НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри

Інформаційних систем і технологій

_____ Швиденко М.З.
(підпис) (ПІБ)

"___" _____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Розробка підходів до створення адаптивного дизайну інтернет магазину з реалізації квітів

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
_____д.е.н., професор_____ _____ __Руденський Р.А.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____к.е.н., доц._____ _____ Мокрієв М.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Дубова Іванна Анатоліївна__
(підпис) (ПІБ студента)

Київ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інформаційних систем і технологій
_____ Швиденко М.З.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
_____ Дубова Іванна Анатоліївна _____
(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи Розробка підходів до створення адаптивного дизайну інтернет магазину з реалізації квітів

затверджена наказом ректора НУБіП України від 16.12.2024 р. №2246с
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи
опис програмного забезпечення

Перелік питань, які потрібно розробити:
1. Системний аналіз предметної області
2. Інформаційне забезпечення системи
3. Розробка прикладного програмного забезпечення
4. Тестування та впровадження програмного забезпечення
Перелік графічних документів (за потреби) _____
_____
_____

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи _____ Мокрієв М.В.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Дубова Іванна Анатоліївна
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Формулювання теми та цілей кваліфікаційної роботи	23.03.2025 – 27.03.2025	
2	Аналіз предметної області та аналогічних систем	27.03.2025 – 01.04.2025	
3	Проектування архітектури інформаційної системи	01.04.2025 – 07.04.2025	
4	Розробка бази даних та бекенд-логіки	07.04.2025 – 25.04.2025	
5	Розробка інтерфейсу користувача (веб-частина)	25.04.2025 – 09.05.2025	
6	Інтеграція компонентів, налагодження	09.05.2025 – 16.05.2025	
7	Тестування, перевірка працездатності системи	16.05.2025 – 18.05.2025	
8	Написання пояснювальної записки	18.05.2025 – 23.05.2025	

Студент _____ _ Дубова Іванна Анатоліївна _
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи __Мокрієв М.В._
(підпис) (прізвище та ініціали)

Анотація

Розроблено адаптивний вебзастосунок для інтернет-магазину «Квіткова Лавка», що спеціалізується на реалізації квітів та супутніх товарів. Система створена на базі сучасного технологічного стеку: React для клієнтської частини, Node.js з фреймворком Express для серверної, SQLite як база даних, Socket.IO для чату в реальному часі, а також JSON Web Tokens і bcryptjs для забезпечення безпеки. Основна предметна область – електронна комерція у сфері продажу квітів. Програма вирішує задачі автоматизації управління каталогом товарів, оформлення замовлень, автентифікації користувачів та підтримки клієнтів.

Функціонал системи включає реєстрацію та автентифікацію користувачів із розподілом ролей (клієнт, менеджер), перегляд та фільтрацію каталогу товарів, управління кошиком, оформлення замовлень, інтерактивний чат, а також адміністративну панель для керування товарами, категоріями та користувачами. Адаптивний дизайн, реалізований через CSS-медіа-запити та гнучкі сітки, забезпечує зручність використання на різних пристроях – від смартфонів до настільних комп’ютерів. Система підтримує українську локалізацію та відповідає стандартам доступності. Розробка сприяє підвищенню ефективності продажів, оптимізації взаємодії з клієнтами та масштабуванню бізнесу в умовах конкурентного ринку.

Annotation

An adaptive web application has been developed for the online store "Kvitkova Lavka," which specializes in the sale of flowers and related products. The system is built using a modern technological stack: React for the client side, Node.js with the Express framework for the server side, SQLite as the database, Socket.IO for real-time chat, as well as JSON Web Tokens and bcryptjs to ensure security. The main domain of focus is e-commerce in the field of flower sales. The application addresses the automation of product catalog management, order processing, user authentication, and customer support.

The system's functionality includes user registration and authentication with role distribution (client, manager), viewing and filtering the product catalog, managing the shopping cart, placing orders, interactive chat, and an administrative panel for managing products, categories, and users. Adaptive design, implemented through CSS media queries and flexible grids, ensures usability on various devices—from smartphones to desktop computers. The system supports Ukrainian localization and adheres to accessibility standards. This development enhances sales efficiency, optimizes customer interaction, and facilitates business scaling in a competitive market environment.

Зміст

ВСТУП

Розробка адаптивного дизайну інтернет-магазину для реалізації квітів є актуальною в сучасних умовах, коли електронна комерція стрімко розвивається, а користувачі все частіше використовують різноманітні пристрої для здійснення покупок. Попит на квіти залишається стабільно високим завдяки їх універсальності як подарунка чи елемента декору, але конкуренція в цій сфері вимагає від магазинів створення зручних, швидких і візуально привабливих платформ. Адаптивний дизайн дозволяє забезпечити однаково якісний досвід користувача на смартфонах, планшетах і настільних комп'ютерах, що є критично важливим для залучення та утримання клієнтів. Відсутність адаптивності може призводити до втрати потенційних покупців через незручність навігації чи повільне завантаження сторінок. Крім того, автоматизація процесів, таких як управління замовленнями, кошиком і профілем користувача, підвищує ефективність роботи магазину, зменшує навантаження на персонал і сприяє масштабуванню бізнесу.

Метою створення програмного додатку є розробка адаптивного вебзастосунку для інтернет-магазину квітів, який забезпечує зручне оформлення замовлень, управління профілем користувача, перегляд каталогу товарів, а також адміністративні функції для менеджерів. Додаток покликаний оптимізувати взаємодію між клієнтами та магазином, забезпечуючи швидкий доступ до асортименту, безпечну автентифікацію, інтеграцію з кошиком і чатом для підтримки. Адаптивний дизайн гарантує комфортне використання на будь-якому пристрої, що підвищує доступність і сприяє зростанню продажів. Крім того, додаток передбачає інтеграцію з вебсокетами для реального часу спілкування, що є важливим для оперативної обробки запитів клієнтів.

Для реалізації проєкту використано сучасний стек технологій, що забезпечує надійність, масштабованість і зручність розробки. На стороні клієнта застосовано бібліотеку React для створення динамічного та адаптивного інтерфейсу користувача, а також React Router DOM для управління

маршрутизацією. Для стилізації використано CSS з підтримкою адаптивних стилів через медіа-запити, що забезпечує коректне відображення на різних пристроях. На стороні сервера використано Node.js з фреймворком Express для обробки HTTP-запитів, а також Socket.IO для реалізації чату в реальному часі. Для управління даними застосовано SQLite як легку базу даних, що підходить для прототипу, з можливістю масштабування до більш потужних СУБД у майбутньому. Безпека забезпечується через JSON Web Tokens (JWT) для автентифікації та bcryptjs для шифрування паролів. Для збирання клієнтського коду використано Webpack із підтримкою Babel для транспіляції сучасного JavaScript. Архітектура додатку базується на принципах компонентного підходу та контекстного управління станом у React, що сприяє модульності та легкості підтримки коду.

На момент написання записки апробація програмного додатку на конференціях чи у вигляді публікацій не проводилася, оскільки проєкт перебуває на стадії розробки та внутрішнього тестування. Однак вебзастосунок був протестований у лабораторних умовах, що підтвердило його працездатність, відповідність функціональним вимогам і коректну роботу адаптивного дизайну на різних пристроях. У майбутньому планується презентація результатів на профільних конференціях і публікація тез для популяризації розробки.

Пояснювальна записка складається з 66 сторінок, містить 35 використаних джерел, 22 таблиці, 27 рисунків та 2 додатки. Вона структурована таким чином, щоб послідовно розкрити всі етапи розробки програмного додатку:

- Системний аналіз предметної області – включає постановку завдання, аналіз аналогів (наприклад, популярних платформ для продажу квітів), а також моделювання предметної області через діаграми UML.
- Інформаційне забезпечення системи – охоплює створення логічної моделі даних, обґрунтування вибору SQLite як СУБД, а також опис реалізації бази даних для зберігання інформації про товари, користувачів і замовлення.

- Розробка прикладного програмного забезпечення – детально описує архітектуру вебзастосунку, вибір інструментів (React, Node.js, Socket.IO тощо) і реалізацію ключових модулів, таких як каталог, кошик, профіль і чат.
- Тестування та впровадження системи – присвячений тестуванню функціональності та адаптивності, опису вимог до апаратного й програмного забезпечення, а також підготовці до розгортання системи.

Така структура забезпечує комплексне висвітлення всіх етапів розробки, від теоретичного аналізу до практичної реалізації та рекомендацій щодо використання системи.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Постановка завдання

Інтернет-магазин «Квіткова Лавка» призначений для автоматизації продажу квітів та супутніх товарів. Основні функціональні вимоги до системи описані в табл. 1.1.

Таблиця 1.1

Основні функціональні вимоги

№	Вимоги	Пояснення
1	Реєстрація та аутентифікація користувачів	Система підтримує створення облікових записів для клієнтів і менеджерів із можливістю входу через email та пароль. Реалізовано захист даних за допомогою JSON Web Token (JWT) та шифрування паролів (bcryptjs) [1]. Користувачі можуть оновлювати профіль, змінювати контактну інформацію та пароль
2	Каталог товарів	Магазин надає структурований каталог із можливістю фільтрації за категоріями, пошуку за назвою та сортування (за ціною, новизною, назвою). Кожен товар має детальну сторінку з описом, ціною, зображенням та інформацією про наявність
3	Кошик та оформлення замовлень	Клієнти можуть додавати товари до кошика, змінювати їх кількість, видаляти позиції та оформлювати замовлення. Система підтримує симуляцію оплати та автоматично очищає кошик після успішного оформлення
4	Керування замовленнями	Користувачі можуть переглядати історію замовлень, а менеджери мають доступ до адмін-панелі для керування товарами, категоріями та користувачами
5	Інтерактивний чат	Реалізовано чат на основі Socket.IO для комунікації між клієнтами та менеджерами в реальному часі, що підвищує рівень обслуговування [2]
6	Адміністративна панель	Менеджери можуть додавати, редагувати та видаляти товари, категорії, а також керувати обліковими записами користувачів
7	Локалізація та інтерактивність	Інтерфейс підтримує українську мову, включає адаптивне меню, сповіщення про дії (наприклад, додавання до кошика) та інтерактивні елементи, такі як випадаючі списки в адмін-панелі

Адаптивність дизайну є ключовою вимогою для забезпечення зручності використання магазину на різних пристроях (десктопи, планшети, смартфони). Основні вимоги до адаптивності показані в табл. 1.2.

Таблиця 1.2

Основні вимоги до адаптивного дизайну

№	Вимоги	Пояснення
1	Респонсивна верстка	Інтерфейс повинен коректно відображатися на екранах різної роздільної здатності. Це досягається за допомогою CSS-медіазапитів та гнучких макетів (flexbox, grid), які адаптують розташування елементів (наприклад, сітку товарів, навігаційне меню) [3]
2	Мобільна навігація	На мобільних пристроях меню трансформується в гамбургер-меню, що забезпечує економію простору та зручність. Елементи, такі як кнопки та поля введення, мають достатній розмір для сенсорного керування
3	Оптимізація зображень	Зображення товарів автоматично масштабуються залежно від розміру екрана, використовуючи атрибути srcset або динамічне завантаження, щоб зменшити час завантаження на мобільних пристроях
4	Швидкодія та оптимізація	Код оптимізовано за допомогою Webpack для мінімізації та об’єднання ресурсів (JS, CSS) [4]. Використання React забезпечує ефективне оновлення компонентів без перезавантаження сторінки, що важливо для мобільних користувачів із обмеженим інтернет-з’єднанням
5	Кросбраузерність та доступність	Дизайн сумісний із сучасними браузерами (Chrome, Firefox, Safari) і відповідає стандартам доступності (ARIA-атрибути для навігації, контрастність тексту)
6	Динамічне завантаження контенту	Сторінки, такі як каталог і деталі товару, завантажують дані асинхронно через API, що зменшує навантаження на клієнтський пристрій і забезпечує плавний перехід між розділами

Постановка завдання для інтернет-магазину «Квіткова Лавка» охоплює створення функціональної платформи з інтуїтивним інтерфейсом, безпечною аутентифікацією, зручним каталогом і можливістю керування замовленнями. Адаптивний дизайн забезпечує однаково комфортне використання на всіх пристроях, що відповідає сучасним вимогам до e-commerce платформ. Реалізована система, як видно з коду, відповідає цим вимогам, використовуючи

сучасні технології (React, Socket.IO, Express) для забезпечення функціональності та адаптивності.

1.2. Огляд інформаційних джерел та існуючих рішень

Сфера онлайн-продажу квітів активно розвивається в Європі та Україні, що зумовлено зростанням попиту на зручні цифрові платформи для придбання квітів та подарунків. Сучасні рішення в цій галузі характеризуються інтеграцією адаптивного дизайну, інтуїтивних інтерфейсів, а також підтримкою додаткових функцій, таких як персоналізація замовлень, швидка доставка та інтеграція з платіжними системами. Розглянуто чотири популярні платформи, які є прикладом успішних рішень у сфері онлайн-продажу квітів.

Bloom & Wild (Велика Британія, Європа) є одним із провідних європейських сервісів для замовлення квітів онлайн (рис. 1.1) [5]. Платформа пропонує адаптивний вебсайт та мобільний додаток, які забезпечують зручний доступ до каталогу букетів, персоналізованих листівок та подарунків. Особливістю є технологія доставки через поштову скриньку, що дозволяє уникнути необхідності особистої присутності отримувача. Сайт використовує мінімалістичний дизайн із чіткою навігацією, що відповідає сучасним UI/UX стандартам. Bloom & Wild також інтегрує інструменти для відстеження замовлень та рекомендаційні алгоритми на основі попередніх покупок.

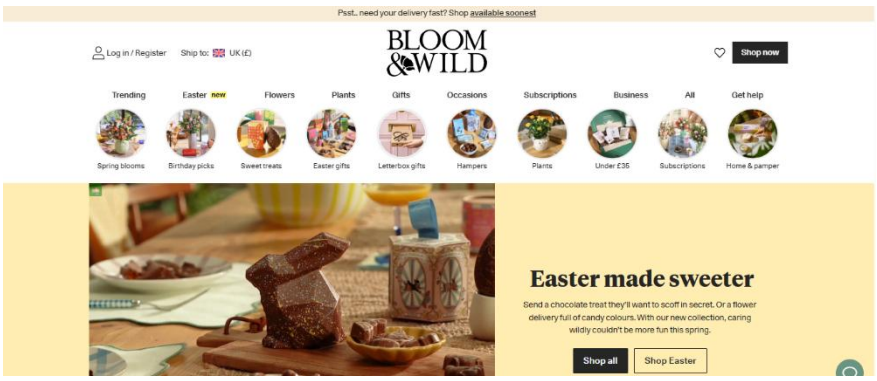

Рис.1.1 Сайт Bloom & Wild [5]

Interflora (Європа) — це міжнародна мережа доставки квітів із сильною присутністю в Європі (рис. 1.2) [6]. Їх вебсайт адаптований для різних пристроїв, із підтримкою як десктопних, так і мобільних версій. Платформа дозволяє користувачам обирати букети за категоріями (наприклад, для весілля чи ювілею), додавати подарунки та вказувати точний час доставки. Interflora використовує яскраві візуальні елементи та чіткі заклики до дії (CTA), що полегшують процес оформлення замовлення. Система також підтримує локалізацію для різних країн, що забезпечує зручність для міжнародних клієнтів.

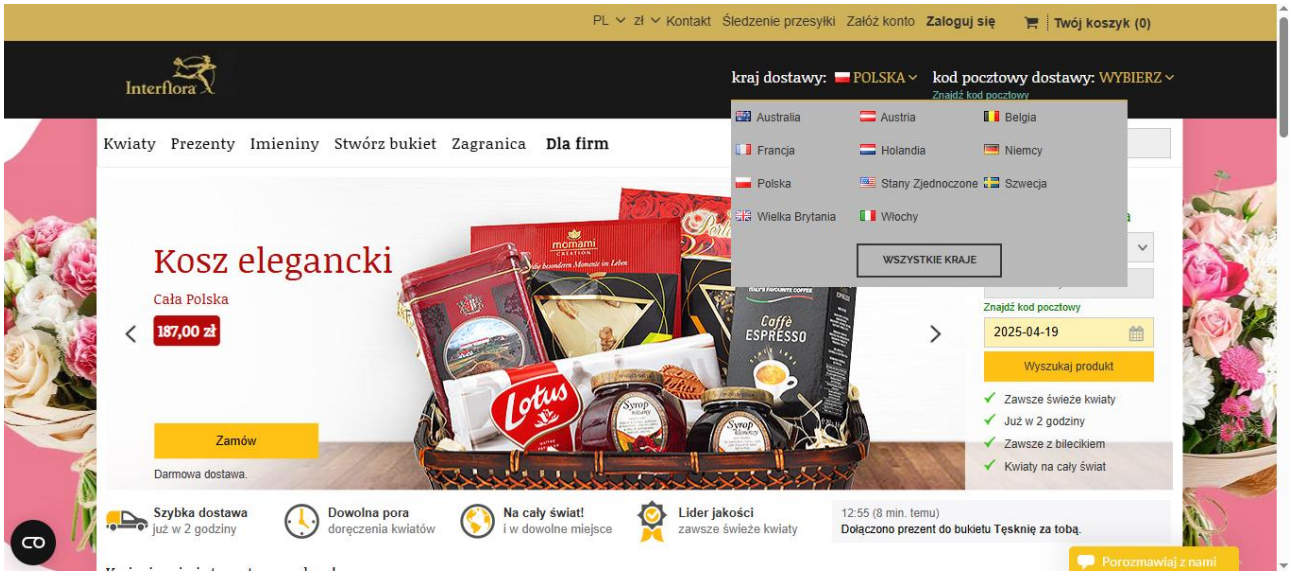

Рис. 1.2 Сайт Interflora [6]

UFL — українська платформа для замовлення квітів із доставкою по Україні та за кордон (рис. 1.3) [7]. Вебсайт UFL має адаптивний дизайн, який забезпечує коректне відображення на смартфонах, планшетах і комп’ютерах. Ключовими функціями є можливість вибору букетів за ціною, типом квітів чи подією, а також інтеграція з популярними платіжними системами, такими як LiqPay та Visa/Mastercard. UFL активно використовує відгуки клієнтів та фотографії реальних букетів, що підвищує довіру користувачів. Дизайн сайту простий, з акцентом на візуальну привабливість та швидке оформлення замовлення.

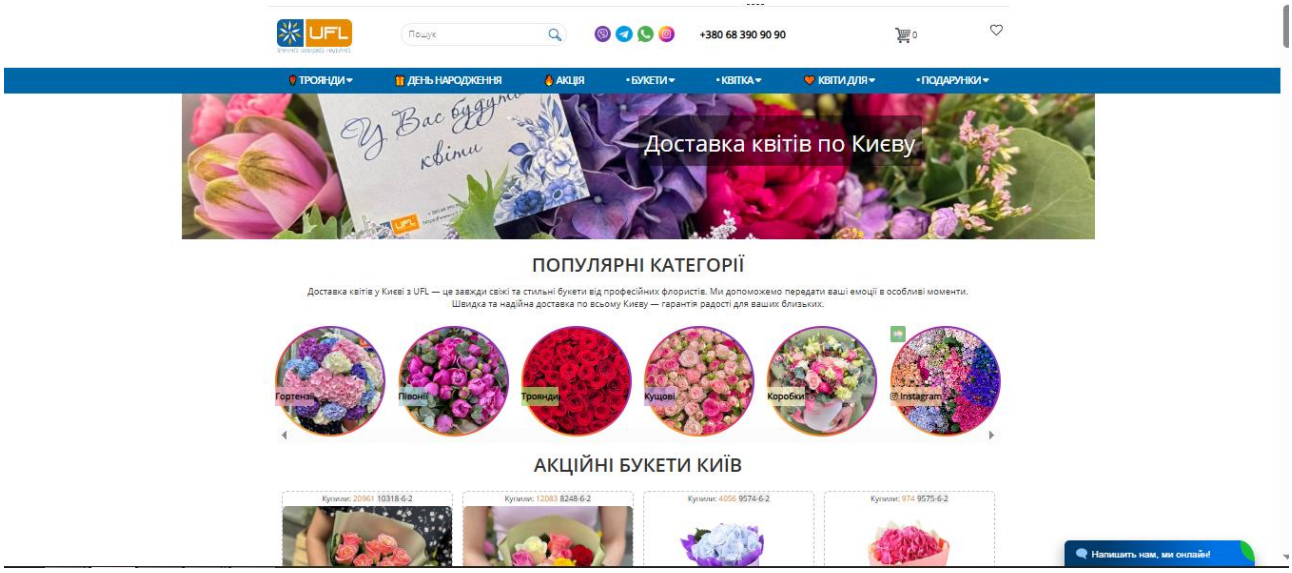

Рис. 1.3 Сайт UFL [7]

Kvitochka — ще один популярний український сервіс доставки квітів (рис. 1.4) [8]. Платформа пропонує адаптивний вебсайт із підтримкою кількох мов, що робить її доступною для іноземних клієнтів. Kvitochka акцентує увагу на локальних флористах, що дозволяє гарантувати свіжість квітів. Сайт має зручну систему фільтрів (за ціною, типом квітів, подією), а також підтримує чат для консультацій із менеджерами. Дизайн виконаний у світлих тонах із чіткою структурою, що сприяє легкому сприйняттю інформації.

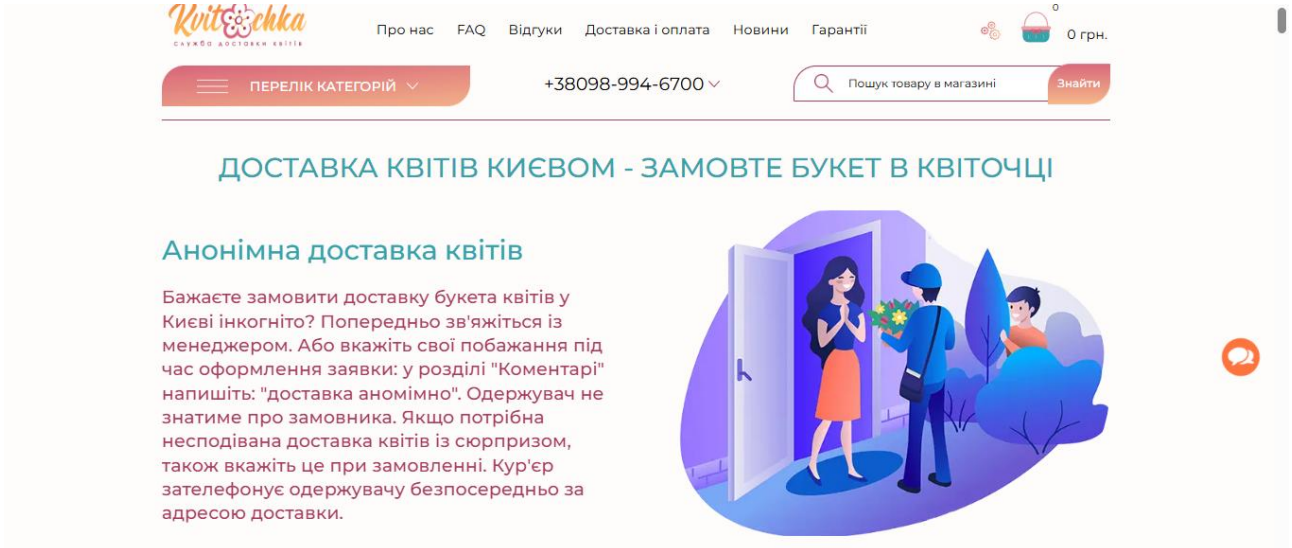

Рис. 1.4 Сайт Kvitochka [8]

Адаптивний вебдизайн (Responsive Web Design, RWD) є ключовим елементом сучасних онлайн-магазинів, включаючи платформи для продажу квітів. Він забезпечує коректне відображення контенту на пристроях із різними розмірами екранів, що особливо важливо, враховуючи зростання частки мобільного трафіку. Основні підходи до адаптивного дизайну, які застосовуються в розглянутих рішеннях, показані в табл.1.3.

Таблиця 1.3

Основні підходи до адаптивного дизайну

№	Підходи	Пояснення
1	Fluid Grid Layouts	Використання гнучких сіток, де розміри елементів задаються у відсотках або відносних одиницях (vw, vh, rem, em), дозволяє контенту адаптуватися до розміру екрана [9]. Наприклад, Bloom & Wild використовує сітку, яка плавно змінює кількість колонок у каталозі залежно від ширини екрана
2	Media Queries	CSS-медіазапити дозволяють застосовувати різні стилі залежно від характеристик пристрою, таких як роздільна здатність чи орієнтація екрана [10]. UFL активно використовує медіазапити для оптимізації відображення великих зображень букетів на мобільних пристроях
3	Mobile-First Design	Розробка починається з базового дизайну для мобільних пристроїв, який потім розширюється для більших екранів [11]. Kvitochka застосовує цей підхід, забезпечуючи швидке завантаження сторінок на смартфонах за рахунок мінімалізації важких графічних елементів
4	Progressive Enhancement	Базова функціональність сайту доступна навіть на старих пристроях чи браузерах, а додаткові функції (анімації, складні переходи) додаються для сучасних платформ [12]. Interflora використовує цей підхід, щоб забезпечити доступність для широкої аудиторії

Аналіз коду проєкту показує, що проєкт "Квіткова Лавка" також використовує адаптивний підхід. Наприклад, компонент Header включає бургер-меню для мобільних пристроїв, а CSS-класи, такі як container і nav, імовірно, використовують гнучкі стилі для адаптації до різних екранів. Однак для повної

відповідності сучасним стандартам варто додати підтримку медіазапитів і оптимізацію зображень для швидшого завантаження.

UI/UX принципи відіграють ключову роль у створенні зручних і привабливих інтерфейсів для онлайн-магазинів квітів, де емоційна складова покупки є важливою [13]. Основні принципи, застосовані в розглянутих рішеннях, показані в табл. 1.4.

Таблиця 1.4

Основні принципи UI/UX дізайну

№	Принципи	Пояснення
1	Простота та чіткість	Усі платформи (Bloom & Wild, Interflora, UFL, Kvitochka) мають інтуїтивну навігацію з чітко виділеними категоріями, кнопками CTA та формами замовлення. Наприклад, UFL використовує великі кнопки "Замовити" та мінімалістичні форми для швидкого оформлення
2	Візуальна ієрархія	Яскраві зображення букетів, контрастні кольори та чіткі заголовки допомагають користувачам швидко знаходити потрібну інформацію. У Bloom & Wild застосовується мінімалістичний фон, щоб підкреслити якість зображень квітів
3	Персоналізація	Платформи дозволяють користувачам додавати персоналізовані листівки чи обирати букети за подіями. У проєкті "Квіткова Лавка" це частково реалізовано через фільтри в компоненті ProductList, які дозволяють сортувати товари за категоріями чи ціною
4	Зворотний зв’язок	Повідомлення про успішне оформлення замовлення чи помилки є важливими для користувацького досвіду. У реалізованому коді компоненти Login і Register показують повідомлення про помилки (error), що відповідає цьому принципу
5	Доступність	Використання семантичного HTML, alt-текстів для зображень та контрастних кольорів забезпечує доступність для користувачів із обмеженими можливостями. Interflora, наприклад, використовує alt-тексти для всіх зображень букетів

У проєкті "Квіткова Лавка" UI/UX принципи частково реалізовані через чітку структуру компонентів (Header, Footer, ProductList), використання React Router для навігації та контексту аутентифікації (AuthContext) для

персоналізації. Однак для покращення досвіду користувачів рекомендується додати анімації для переходів між сторінками, покращити доступність (наприклад, додати підтримку клавіатурної навігації) та оптимізувати час завантаження зображень.

Сучасні рішення в сфері онлайн-продажу квітів, такі як Bloom & Wild, Interflora, UFL та Kvitochka, демонструють високий рівень адаптивності та відповідність UI/UX стандартам. Вони використовують гнучкі сітки, медіазапити та mobile-first підходи для забезпечення зручності на всіх пристроях. Проєкт "Квіткова Лавка", представлений у реалізованому коді, має міцну основу для створення адаптивного дизайну, але потребує додаткової оптимізації для відповідності найкращим практикам. Впровадження сучасних UI/UX принципів, таких як персоналізація, зворотний зв’язок та доступність, може значно підвищити конкурентоспроможність платформи.

1.3. Моделювання предметної області

Моделювання предметної області є ключовим етапом системного аналізу, що дозволяє формалізувати структуру та поведінку системи, а також визначити взаємодію між її компонентами та користувачами. У контексті розробки адаптивного дизайну інтернет-магазину з реалізації квітів предметна область охоплює процеси купівлі-продажу квіткової продукції, управління асортиментом, взаємодію з клієнтами та адміністрування системи. На основі реалізованих кодів, що реалізують функціонал магазину, виконано моделювання предметної області з використанням UML-діаграм (діаграми прецедентів та діаграми класів) та розроблено сценарії користувачів для основних ролей: покупця та адміністратора.

Діаграма прецедентів (рис. 1.5) відображає основні функціональні можливості системи та взаємодію користувачів із нею [14]. У системі інтернет-магазину квітів визначено два основних актори: Покупець (клієнт) та Адміністратор (менеджер).

Рис. 1.5 Діаграма прецедентів

Ключові прецеденти описані в табл. 1.5.

Таблиця 1.5

Основні прецеденти

№	Прецеденти	Опис прецедентів
1	Покупець	
1.1	Реєстрація в системі	створення облікового запису з вказівкою імені, email, пароля та ролі (клієнт)
1.2	Авторизація	вхід у систему за допомогою email та пароля
1.3	Перегляд каталогу товарів	ознайомлення з асортиментом квітів, фільтрація за категоріями, сортування за ціною чи назвою
1.4	Перегляд деталей товару	отримання інформації про конкретний продукт (ціна, опис, наявність)
1.5	Додавання товару до кошика	вибір кількості товару та додавання до кошика
1.6	Оформлення замовлення	вибір товарів із кошика, підтвердження замовлення та оплата
1.7	Перегляд історії замовлень	доступ до списку попередніх замовлень
1.8	Оновлення профілю	редагування особистих даних (ім’я, телефон, адреса)
1.9	Використання чату	спілкування з менеджером для консультацій
2	Адміністратор	
2.1	Управління товарами	додавання, редагування, видалення товарів у каталозі
2.2	Управління категоріями	створення, редагування, видалення категорій товарів
2.3	Управління користувачами	перегляд списку користувачів, зміна їх ролей чи видалення
2.4	Обробка замовлень	перегляд, підтвердження або скасування замовлень клієнтів

Діаграма прецедентів відображає взаємозв’язки між акторами та прецедентами, де, наприклад, прецедент «Оформлення замовлення» залежить від «Авторизації» та «Додавання товару до кошика».

Діаграма класів (рис. 1.6) описує статичну структуру системи, включаючи основні сутності та їх взаємозв’язки [15].

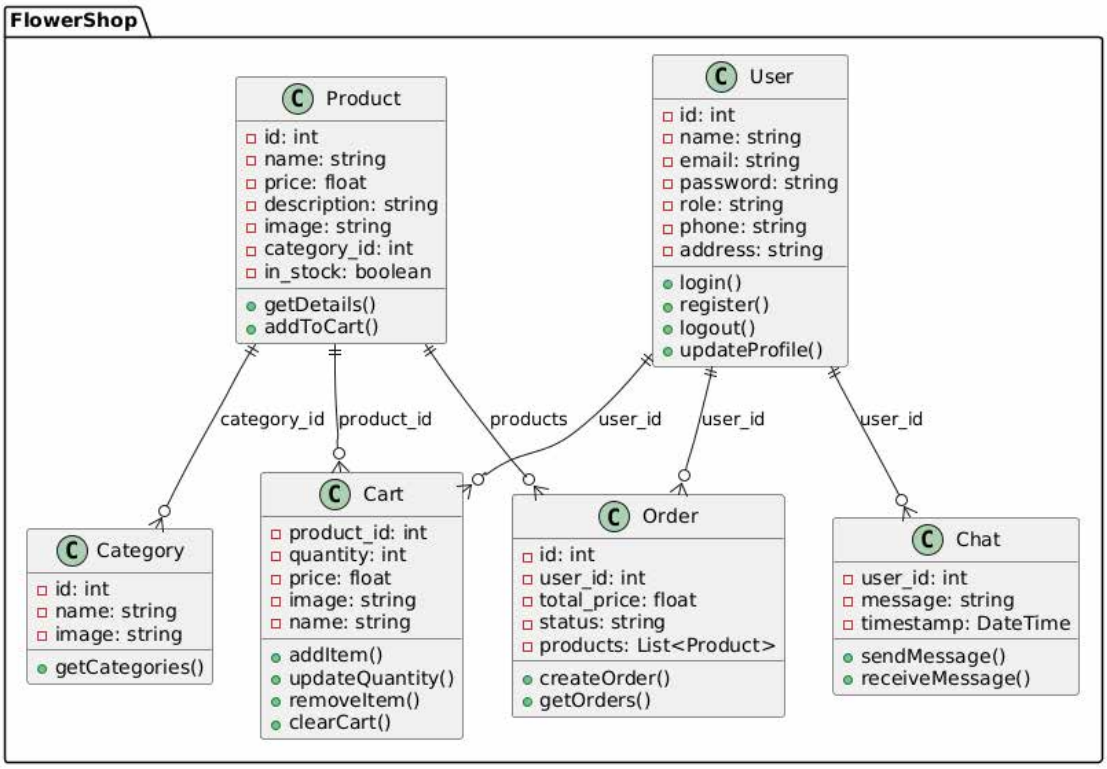

Рис. 1.6 Діаграма класів

На основі аналізу кодів проєкту виділено ключові класи, описані в табл. 1.6.

Таблиця 1.6

Основні класи

№	Класи	Опис класів
1	2	3
1	User	Атрибути: id, name, email, password, role (client або manager), phone, address
		Операції: login(), register(), logout(), updateProfile()
		Асоціації: пов’язаний із класами Order та Cart
2	Product	Атрибути: id, name, price, description, image, category_id, in_stock
		Операції: getDetails(), addToCart()
		Асоціації: належить до Category, входить до Cart та Order
3	Category	Атрибути: id, name, image
		Операції: getCategories()
		Асоціації: містить Product

Продовження табл. 1.6

1	2	3
4	Cart	Атрибути: product_id, quantity, price, image, name
		Операції: addItem(), updateQuantity(), removeItem(), clearCart()
		Асоціації: пов’язаний із User та Product
5	Order	Атрибути: id, user_id, total_price, status, products
		Операції: createOrder(), getOrders()
		Асоціації: пов’язаний із User та Product
6	Chat	Атрибути: user_id, message, timestamp
		Операції: sendMessage(), receiveMessage()
		Асоціації: пов’язаний із User

Класи пов’язані асоціаціями, агрегаціями та композиціями. Наприклад, User має композицію з Cart (кошик належить конкретному користувачу), а Product агрегується в Order (замовлення містить товари).

Сценарії користувачів описані в табл. 1.7.

Таблиця 1.7

Сценарії користувачів

№	Сценарій	Дії сценарію
1	2	3
1. Сценарій користувача: Покупець		
1.1	Перегляд каталогу та вибір товару	Покупець відкриває головну сторінку магазину
		Переходить до розділу «Каталог» через меню
		Використовує фільтри (категорія, пошук, сортування) для вибору товару
		Клікає на товар, щоб переглянути деталі (ціна, опис, наявність)
1.2	Додавання до кошика та оформлення замовлення	Покупець обирає кількість товару та натискає «Додати до кошика»
		Переходить до кошика, перевіряє товари та загальну суму
		Натискає «Оформити замовлення», проходить авторизацію (якщо не авторизований)
		Підтверджує замовлення та виконує оплату через симульовану платіжну систему
1.3	Управління профілем	Покупець переходить до розділу «Профіль»
		Редагує особисті дані (ім’я, телефон, адреса)
		Переглядає історію замовлень або переглядів товарів

Продовження табл. 1.7

1	2	3
2. Сценарій користувача: Адміністратор		
2.1	Управління асортиментом	Адміністратор авторизується в системі
		Переходить до розділу «Адмін» → «Товари»
		Додає новий товар (вказує назву, ціну, категорію, зображення) або редагує/видаляє існуючий
2.2	Управління категоріями	У розділі «Адмін» → «Категорії» створює нову категорію або редагує існуючу
		Завантажує зображення для категорії, якщо необхідно
2.3	Обробка замовлень	У розділі «Замовлення» переглядає список активних замовлень
		Підтверджує або скасовує замовлення, оновлює статус

Моделювання предметної області інтернет-магазину квітів дозволило чітко визначити функціональні вимоги, структуру даних та взаємодію користувачів із системою. Діаграма прецедентів відобразила ключові сценарії використання для покупців та адміністраторів, а діаграма класів формалізувала основні сутності та їх зв’язки. Сценарії користувачів деталізували типові дії, що забезпечують зручність та ефективність роботи з системою. Отримані моделі є основою для реалізації адаптивного дизайну, який враховує потреби різних категорій користувачів та забезпечує інтуїтивно зрозумілий інтерфейс.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1. Логічна модель даних

Логічна модель даних є ключовим елементом інформаційного забезпечення системи інтернет-магазину з реалізації квітів, оскільки вона визначає структуру та взаємозв’язки між основними об’єктами системи [16]. У контексті розробки адаптивного дизайну інтернет-магазину, логічна модель даних забезпечує основу для організації інформації, яка відображається на клієнтському інтерфейсі, а також для взаємодії між фронтендом і бекендом. У даному розділі розглядається структура даних для основних сутностей системи: товарів, замовлень і клієнтів, що відповідає функціональним вимогам системи, реалізованим у коді проєкту.

Структура даних системи зображена на рис. 2.1.

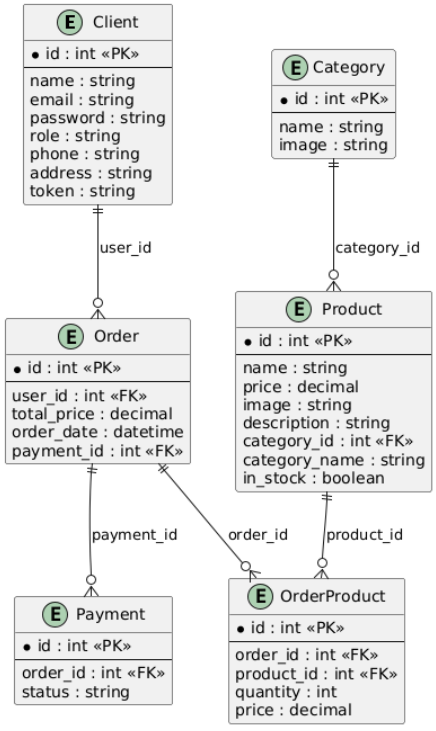

Рис. 2.1 ER-діаграма

Сутність "Товар" є центральною для функціонування інтернет-магазину, оскільки вона описує продукцію, доступну для продажу. На основі аналізу коду,

зокрема компонентів ProductList і ProductDetail, структура даних товару включає атрибути, описані в табл. 2.1.

Таблиця 2.1

Атрибути сутності "Товар"

№	Атрибути	Опис атрибутів
1	id (унікальний ідентифікатор)	первинний ключ, що забезпечує унікальність кожного товару в базі даних. Використовується для звернення до конкретного товару (наприклад, у маршруті /products/:id).
2	name (назва)	текстовий атрибут, що відображає назву товару, наприклад, "Троянди червоні". Відображається в каталозі та на сторінці детального перегляду
3	price (ціна)	числовий атрибут, який вказує вартість товару в гривнях. Форматується з двома знаками після коми для відображення (наприклад, product.price.toFixed(2))
4	image (зображення)	текстовий атрибут, що містить URL або шлях до зображення товару. У разі відсутності зображення використовується заглушка (https://via.placeholder.com/300)
5	description (опис)	текстовий атрибут, що містить детальний опис товару. Може бути відсутнім, у такому випадку відображається повідомлення "Опис відсутній"
6	category_id (ідентифікатор категорії)	зовнішній ключ, що пов’язує товар із відповідною категорією (наприклад, "Букети" або "Кімнатні рослини")
7	category_name (назва категорії)	похідний атрибут, що відображає назву категорії для зручності користувача
8	in_stock (наявність)	логічний атрибут, що вказує, чи є товар у наявності. Впливає на можливість додавання товару до кошика

Ця структура забезпечує гнучкість відображення товарів у каталозі, фільтрацію за категоріями та сортування за різними критеріями (наприклад, за ціною або назвою), як це реалізовано у компоненті ProductList. Взаємозв’язок із категоріями дозволяє організувати товари в ієрархічну структуру, що полегшує навігацію користувачів.

Сутність "Замовлення" відображає транзакції, здійснені клієнтами, і є основою для управління продажами. На основі коду, зокрема компонента Cart і логіки оформлення замовлення (handleCheckout), структура даних замовлення описана в табл. 2.2,

Таблиця 2.2

Атрибути сутності "Замовлення"

№	Атрибути	Опис атрибутів
1	id (унікальний ідентифікатор)	первинний ключ для ідентифікації замовлення
2	user_id (ідентифікатор користувача)	зовнішній ключ, що пов’язує замовлення з клієнтом, який його здійснив. Забезпечує персоналізацію та історію покупок
3	products (список товарів)	масив об’єктів, де кожен об’єкт містить
3.1	product_id (ідентифікатор товару)	посилання на товар
3.2	quantity (кількість)	числовий атрибут, що вказує кількість одиниць товару в замовленні
3.2	price (ціна)	ціна товару на момент замовлення, що фіксується для уникнення змін при оновленні каталогу
4	total_price (загальна сума)	числовий атрибут, що відображає сумарну вартість замовлення, розраховану як сума добутків ціни та кількості для всіх товарів
5	order_date (дата замовлення)	атрибут типу дата/час, що фіксує момент створення замовлення (хоча в коді явно не вказано, це стандартна практика)
6	payment_id (ідентифікатор платежу)	зовнішній ключ, що пов’язує замовлення з інформацією про оплату, як це реалізовано в симуляції платежу (/payments/simulate)

Ця структура підтримує процес оформлення замовлення, включаючи розрахунок вартості, збереження даних у локальному сховищі (localStorage) та відправлення на сервер через API (/orders). Взаємозв’язок із сутністю "Клієнт" дозволяє відстежувати історію замовлень, а з таблицею платежів — статус оплати.

Сутність "Клієнт" описує користувачів системи, які можуть бути як покупцями, так і адміністраторами (менеджерами). На основі коду, зокрема компонентів AuthProvider, Login, Register і Profile, структура даних клієнта описана в табл. 2.3.

Таблиця 2.3

Атрибути сутності "Клієнт"

№	Атрибути	Опис атрибутів
1	id (унікальний ідентифікатор)	первинний ключ для унікальної ідентифікації користувача
2	name (ім’я)	текстовий атрибут, що вказує ім’я користувача, введене під час реєстрації
3	email (електронна пошта)	унікальний текстовий атрибут, що використовується для входу та ідентифікації
4	password (пароль)	зашифрований текстовий атрибут для аутентифікації (з використанням bcryptjs, як зазначено в package.json)
5	role (роль)	текстовий атрибут, що визначає тип користувача (client або manager). Впливає на доступ до функціоналу, наприклад, адмін-панелі
6	phone (телефон)	текстовий атрибут, необов’язковий, для контактної інформації
7	address (адреса)	текстовий атрибут, необов’язковий, для доставки замовлень
8	token (токен аутентифікації)	тимчасовий атрибут, що генерується за допомогою jsonwebtoken і зберігається в localStorage для авторизації запитів

Ця структура забезпечує управління аутентифікацією, авторизацією та профілем користувача. Взаємозв’язок із замовленнями дозволяє зберігати історію покупок, а з ролями — розмежовувати доступ до функціоналу, наприклад, керування товарами для менеджерів.

Логічна модель даних передбачає наступні зв’язки:

- Товар — Категорія має зв’язок "багато до одного", де кожен товар належить до однієї категорії, а одна категорія може містити багато товарів.

- Замовлення — Клієнт має зв’язок "багато до одного", де клієнт може мати багато замовлень, а кожне замовлення належить одному клієнту.

- Замовлення — Товар має зв’язок "багато до багатьох", який реалізується через проміжну таблицю. Замовлення може містити багато товарів, а кожен товар може бути в багатьох замовленнях.

Ці зв’язки реалізовані через зовнішні ключі та API-запити, що забезпечують цілісність даних і підтримують функціонал, такий як фільтрація товарів за категоріями чи відображення історії замовлень.

Логічна модель даних інтернет-магазину "Квіткова Лавка" забезпечує структуроване зберігання та обробку інформації про товари, замовлення та клієнтів. Вона відповідає вимогам адаптивного дизайну, дозволяючи ефективно відображати дані на різних пристроях через API та клієнтський інтерфейс. Використання унікальних ідентифікаторів, зовнішніх ключів і атрибутів, таких як роль користувача чи наявність товару, забезпечує гнучкість і масштабованість системи, що є основою для її подальшого розвитку.

2.2. Вибір системи управління базою даних

Система управління базами даних (СУБД) є ключовим компонентом інформаційного забезпечення будь-якого інтернет-магазину, зокрема такого, що спеціалізується на реалізації квітів [17]. Вибір оптимальної СУБД визначає ефективність обробки даних, швидкість виконання запитів, масштабованість системи та зручність її адміністрування. У контексті розробки інтернет-магазину «Квіткова Лавка» було проведено порівняльний аналіз кількох популярних СУБД, а саме MySQL, PostgreSQL, MongoDB та SQLite, з урахуванням вимог до проєкту, таких як адаптивний дизайн, підтримка транзакцій, обробка даних користувачів, каталогів товарів та замовлень. На основі цього аналізу було обґрунтовано вибір SQLite як основної СУБД для проєкту.

2.2.1. Порівняння альтернатив

MySQL є однією з найпоширеніших реляційних СУБД із відкритим кодом, яка широко використовується для вебдодатків [15]. Вона підтримує швидке виконання запитів, має розвинену екосистему інструментів для адміністрування та оптимізації, а також забезпечує надійну підтримку транзакцій за стандартом ACID. MySQL добре підходить для проєктів із великою кількістю одночасних користувачів і складними запитами, однак потребує окремого серверного розгортання, що може ускладнити початкову розробку та розгортання для невеликих проєктів. Крім того, адміністрування MySQL вимагає певного рівня технічної експертизи, що може бути недоцільним для проєктів із обмеженим бюджетом.

PostgreSQL також є реляційною СУБД із відкритим кодом, яка вирізняється розширеними можливостями, такими як підтримка складних типів даних (JSON, масиви), розвинена система тригерів і процедур, а також висока надійність і масштабованість [19]. Ця СУБД ідеально підходить для складних систем із великими обсягами даних і потребою в аналітичних запитах. Проте, як і MySQL, PostgreSQL вимагає серверного розгортання та значних ресурсів для налаштування й оптимізації, що може бути надмірним для проєкту з обмеженим масштабом, такого як «Квіткова Лавка».

MongoDB є документоорієнтованою NoSQL СУБД, яка зберігає дані у форматі JSON-подібних документів [20]. Вона забезпечує високу гнучкість у роботі з неструктурованими або напівструктурованими даними, що робить її привабливою для проєктів із динамічними схемами даних. MongoDB добре масштабується горизонтально, що корисно для систем із великою кількістю запитів. Однак для реляційних даних, таких як каталог товарів, замовлення чи профілі користувачів, MongoDB може бути менш ефективною через відсутність строгих зв’язків між таблицями та складність реалізації транзакцій. Крім того, MongoDB потребує додаткових зусиль для забезпечення консистентності даних, що може ускладнити розробку.

SQLite є вбудованою реляційною СУБД, яка не потребує окремого серверного процесу, оскільки працює безпосередньо з файлом бази даних [21]. Вона легка, проста в налаштуванні та ідеально підходить для проєктів із невеликим або середнім обсягом даних. SQLite підтримує більшість стандартних SQL-функцій, включаючи транзакції за стандартом ACID, що забезпечує надійність операцій [22]. Основним недоліком SQLite є обмежена масштабованість і продуктивність при великій кількості одночасних записів, що робить її менш придатною для високонавантажених систем. Проте для проєктів із помірним навантаженням, таких як інтернет-магазин із локальним або регіональним охопленням, SQLite є ефективним рішенням.

2.2.2. Обґрунтування вибору СУБД

Для інтернет-магазину «Квіткова Лавка» було обрано SQLite як основну СУБД. Це рішення базується на кількох ключових факторах (табл. 2.4.), які відповідають вимогам проєкту.

Таблиця 2.4

Причини вибору SQLite

№	Причини	Пояснення
1	2	3
1	Простота розгортання та адміністрування	SQLite не потребує окремого серверного розгортання, що значно спрощує процес налаштування як на етапі розробки, так і під час розгортання. Для проєкту, який передбачає швидке створення прототипу та подальше тестування, це є суттєвою перевагою. У коді проєкту (див. package.json) зазначено використання sqlite3 як залежності, що підтверджує інтеграцію SQLite у серверну частину
2	Достатня функціональність для реляційних даних	Структура даних інтернет-магазину включає чітко визначені сутності, такі як користувачі, товари, категорії, замовлення та історія переглядів, які найкраще моделюються за допомогою реляційної моделі. SQLite підтримує SQL-запити, транзакції та зв’язки між таблицями, що забезпечує ефективну роботу з цими даними.

3	Низькі вимоги до ресурсів	SQLite є легкою СУБД, яка не потребує значних апаратних ресурсів. Це робить її ідеальною для проєктів із обмеженим бюджетом або для розгортання на недорогих хостингах. Для інтернет-магазину з локальним охопленням, де кількість одночасних користувачів обмежена, продуктивність SQLite є достатньою

Продовження табл. 2.4

1	2	3
4	Кросплатформна сумісність	SQLite працює на всіх основних операційних системах без необхідності додаткового налаштування, що полегшує розробку та тестування. Це особливо важливо для проєкту, який використовує сучасні інструменти, такі як Node.js і React, як зазначено в package.json і webpack.config.js
5	Відповідність вимогам безпеки	У проєкті реалізовано автентифікацію та авторизацію користувачів (див. AuthProvider у client.js), що передбачає безпечне зберігання даних, таких як паролі (зашифровані за допомогою bcryptjs) і токени JWT. SQLite забезпечує достатній рівень безпеки для зберігання таких даних, а її вбудована природа зменшує ризик зовнішніх атак на сервер бази даних

Хоча SQLite має обмеження в масштабованості, ці недоліки не є критичними для поточного етапу розвитку проєкту. У разі зростання навантаження в майбутньому можливий перехід на більш потужну СУБД, таку як PostgreSQL, із мінімальними змінами в структурі даних завдяки стандартизованому SQL. Наразі SQLite забезпечує оптимальний баланс між простотою, функціональністю та продуктивністю для реалізації адаптивного дизайну інтернет-магазину «Квіткова Лавка».

Таким чином, вибір SQLite як СУБД для проєкту є обґрунтованим рішенням, яке відповідає технічним, економічним і функціональним вимогам, забезпечуючи надійну основу для інформаційного забезпечення системи.

2.3. Проєктування та реалізація інформаційної бази

Проєктування та реалізація інформаційної бази є ключовим етапом у створенні адаптивного інтернет-магазину для реалізації квітів. Інформаційна база забезпечує ефективне зберігання, обробку та управління даними, необхідними для функціонування системи. У даному підрозділі розглядається фізична модель даних, структура SQL-таблиць, а також організація зв’язків між ними, що відповідає вимогам до функціональності та адаптивності системи.

Фізична модель даних являє собою деталізоване представлення структури бази даних, яке враховує технічні показники реалізації, такі як типи даних, індекси, ключі та обмеження цілісності [23]. Для інтернет-магазину квітів фізична модель даних розроблена з урахуванням основних сутностей, що відображають бізнес-процеси: користувачі, товари, категорії, замовлення, платежі, кошик, історія переглядів та чат. Кожна сутність представлена окремою таблицею, що забезпечує модульність і масштабованість системи [24].

Основні сутності та їх призначення описані в табл. 2.5.

Таблиця 2.5

Основні сутності та їх призначення

№	Сутності	Призначення
1	Користувачі	Зберігає інформацію про клієнтів і менеджерів, включаючи персональні дані (ім’я, електронна пошта, пароль) та роль у системі
2	Товари	Містить дані про квіткові товари, такі як назва, опис, ціна, зображення, наявність на складі та категорія
3	Категорії	Використовується для класифікації товарів, включаючи назву категорії та зображення
4	Замовлення	Зберігає інформацію про замовлення клієнтів, включаючи перелік товарів, загальну суму та статус
5	Платежі	Відображає дані про фінансові транзакції, пов’язані з оплатою замовлень
6	Кошик	Тимчасово зберігає товари, які користувач додав для подальшого оформлення замовлення
7	Історія переглядів	Фіксує дії клієнтів щодо перегляду товарів для персоналізації пропозицій
8	Чат	Забезпечує збереження повідомлень між клієнтами та менеджерами для підтримки комунікації

Фізична модель враховує вимоги до адаптивності, зокрема швидкий доступ до даних для відображення на різних пристроях, оптимізацію запитів для зменшення часу відповіді та підтримку високого рівня безпеки даних користувачів. Для цього застосовуються індекси на часто використовувані поля, такі як ідентифікатори та категорії, а також обмеження цілісності для забезпечення консистентності даних.

Для реалізації фізичної моделі даних використовується реляційна база даних SQLite, яка забезпечує легкість інтеграції та достатню продуктивність для невеликих і середніх інтернет-магазинів. SQL-структури таблиць (рис. 2.2) розроблені з урахуванням типів даних, що оптимально відповідають їх вмісту, та обмежень, які гарантують цілісність і унікальність записів (табл. 2.6).

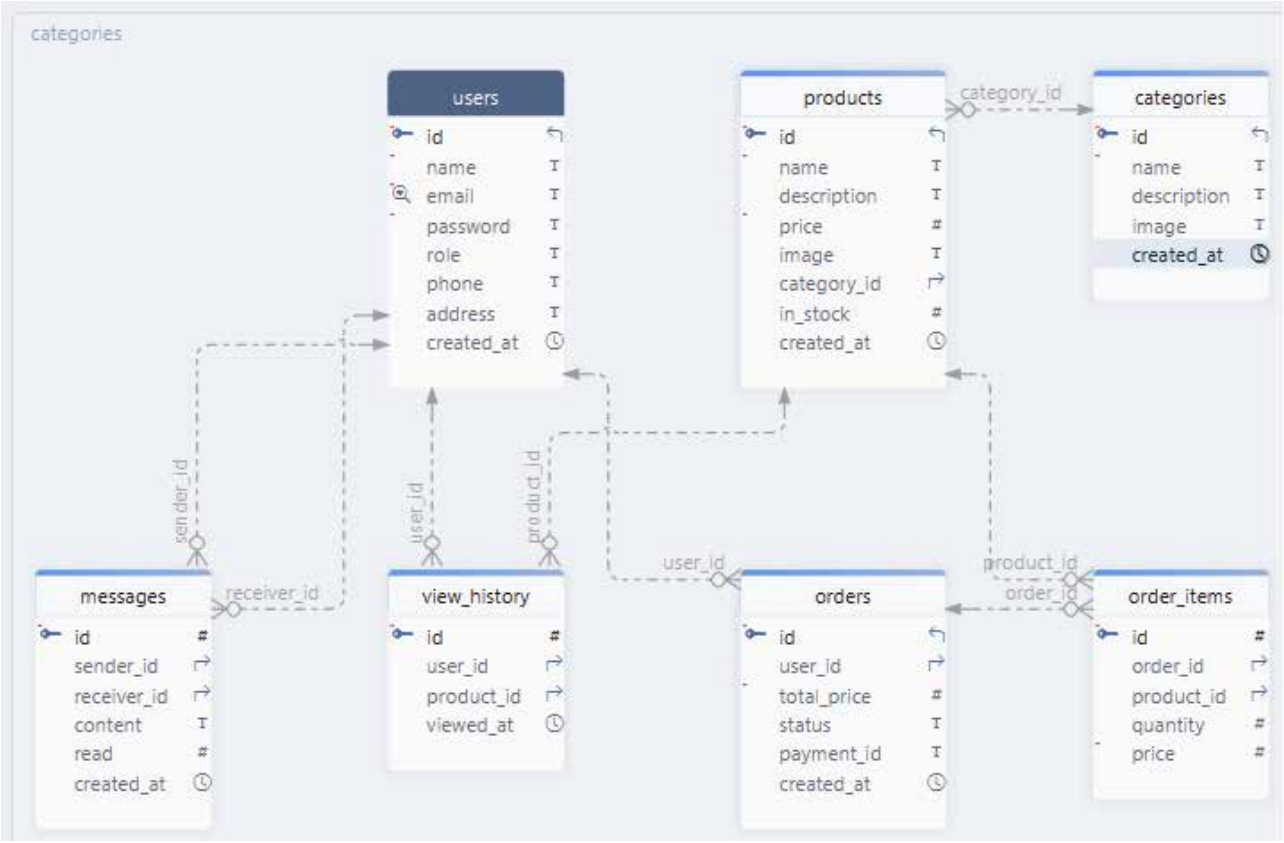

Рис. 2.2 Структура бази даних

Таблиця 2.6

Основні таблиці бази даних

№	Таблиці	Опис таблиць
1	2	3
1	Таблиця користувачів	Містить поля для зберігання унікального ідентифікатора, імені, електронної пошти, хешованого пароля, номера телефону, адреси та ролі (клієнт або менеджер). Поле електронної пошти має обмеження унікальності, щоб уникнути дублювання облікових записів, а поле ролі використовує тип даних для фіксованих значень

Продовження табл. 2.6

1	2	3
2	Таблиця товарів	Включає унікальний ідентифікатор, назву, опис, ціну (з точністю до двох знаків після коми), URL зображення, логічний прапорець наявності на складі та ідентифікатор категорії. Ціна зберігається як числове значення для точних розрахунків
3	Таблиця категорій	Містить ідентифікатор, назву та опціональне поле для URL зображення. Назва категорії є обов’язковим полем
4	Таблиця замовлень	Зберігає ідентифікатор замовлення, ідентифікатор користувача, дату створення, загальну суму та статус (наприклад, "в обробці", "оплачено"). Для зв’язку з товарами використовується допоміжна таблиця, яка містить ідентифікатори замовлення, товару та кількість
5	Таблиця платежів	Включає ідентифікатор платежу, ідентифікатор замовлення, суму, статус платежу та дату. Статус платежу може мати значення, такі як "успішно" або "очікує"
6	Таблиця історії переглядів	Фіксує ідентифікатор користувача, товару та дату перегляду для аналізу поведінки клієнтів
7	Таблиця чату	Містить ідентифікатор повідомлення, відправника, одержувача, текст повідомлення та час відправлення

Кожна таблиця має первинний ключ (зазвичай автоінкрементний ідентифікатор), що забезпечує унікальність записів. Обмеження NOT NULL застосовуються до обов’язкових полів, таких як назва товару чи електронна пошта користувача. Для підвищення продуктивності створюються індекси на поля, що часто використовуються у запитах, наприклад, ідентифікатор категорії в таблиці товарів.

Зв’язки між таблицями реалізуються за допомогою зовнішніх ключів, що забезпечують реляційну цілісність даних. Основні типи зв’язків у базі даних [25]:

- Один до багатьох передбачає зв’язок між таблицею категорій і таблицею товарів, де одна категорія може містити багато товарів, але кожен товар належить лише до однієї категорії. Зовнішній ключ у таблиці товарів посилається на ідентифікатор категорії.

- Один до багатьох також охоплює зв’язок між таблицею користувачів і таблицею замовлень, де один користувач може мати багато замовлень, а кожне замовлення належить одному користувачу.
- Багато до багатьох реалізується через допоміжну таблицю для зв’язку між таблицями замовлень і товарів. Ця таблиця містить ідентифікатори замовлення і товару, а також кількість, що дозволяє одному замовленню включати кілька товарів, а одному товару бути частиною кількох замовлень.
- Один до одного описує зв’язок між таблицею замовлень і таблицею платежів, де кожне замовлення може мати лише один платіж, а кожен платіж відповідає одному замовленню.

Зовнішні ключі налаштовані з обмеженнями ON DELETE і ON UPDATE для контролю поведінки при видаленні або оновленні записів [26]. Наприклад, при видаленні категорії товари, пов’язані з нею, можуть бути позначені як некатегоризовані, щоб уникнути втрати даних.

Проєктування та реалізація інформаційної бази для інтернет-магазину квітів забезпечують надійне зберігання та ефективне управління даними. Фізична модель даних враховує всі ключові сутності бізнес-процесів, а SQL-структури таблиць оптимізовано для швидкого доступу та безпеки. Зв’язки між таблицями, реалізовані через зовнішні ключі, гарантують цілісність і консистентність даних. Такий підхід дозволяє створити гнучку та масштабовану базу даних, яка відповідає вимогам адаптивного дизайну та забезпечує зручність використання системи як для клієнтів, так і для менеджерів.

3. РОЗРОБКА ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура вебзастосунку

Розробка вебзастосунку для інтернет-магазину «Квіткова Лавка» базується на клієнт-серверній моделі архітектури, яка є однією з найпоширеніших у створенні сучасних вебдодатків. Ця модель передбачає чіткий розподіл функціональних обов’язків між клієнтською та серверною частинами, що забезпечує модульність, масштабованість і зручність підтримки системи. У даному розділі буде детально описано архітектуру вебзастосунку, включаючи розподіл на фронтенд і бекенд, а також ключові технології, використані для реалізації.

Клієнт-серверна архітектура вебзастосунку «Квіткова Лавка» побудована на взаємодії двох основних компонентів: клієнтської частини, яка виконується в браузері користувача, та серверної частини, що обробляє запити й забезпечує доступ до даних [27]. Клієнтська частина відповідає за відображення інтерфейсу користувача, обробку взаємодії з ним і надсилання запитів до сервера. Серверна частина, у свою чергу, обробляє ці запити, виконує бізнес-логіку, взаємодіє з базою даних і повертає відповідні результати клієнту. Для обміну даними між клієнтом і сервером використовується протокол HTTP/HTTPS, а для асинхронної передачі даних у реальному часі застосовується технологія WebSocket через бібліотеку Socket.IO [28].

Клієнт-серверна модель (рис. 3.1) забезпечує централізоване управління даними та логікою на сервері, що підвищує безпеку та спрощує оновлення функціоналу [29]. Водночас клієнтська частина, побудована з використанням сучасних JavaScript-фреймворків, забезпечує високу інтерактивність і адаптивність інтерфейсу [30]. Такий підхід дозволяє створювати гнучкі та масштабовані системи, які можуть адаптуватися до різних пристроїв і потреб користувачів.

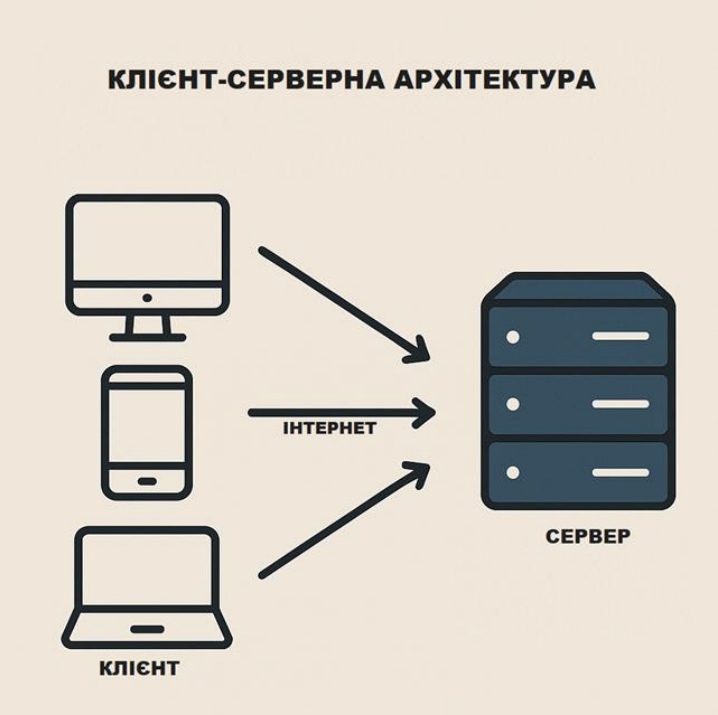

Рис. 3.1 Шаблон клієнт-серверної архітектури

Архітектура вебзастосунку чітко розподілена на фронтенд і бекенд, що відповідає принципам поділу відповідальностей (Separation of Concerns). Варто розглянути детальний опис кожної складової.

Фронтенд вебзастосунку реалізовано з використанням бібліотеки React, яка є однією з провідних технологій для створення динамічних і адаптивних інтерфейсів користувача [31]. React забезпечує компонентно-орієнтовану архітектуру, що дозволяє розбити інтерфейс на незалежні модулі (компоненти), такі як Header, Footer, ProductList, ProductDetail тощо. Кожен компонент відповідає за певну частину функціоналу, що спрощує розробку, тестування та підтримку.

Для управління маршрутизацією між сторінками застосовано бібліотеку React Router DOM, яка забезпечує навігацію без перезавантаження сторінки (Single Page Application, SPA) [32]. Це дозволяє створювати плавний користувацький досвід, коли перехід між сторінками, наприклад, з каталогу товарів до сторінки окремого продукту, відбувається швидко й без затримок.

Адаптивний дизайн фронтенду досягається завдяки використанню CSS-модулів і сучасних підходів до стилізації, таких як Flexbox і Grid [3]. Для обробки

стилів застосовуються бібліотеки style-loader і css-loader, що інтегруються з Webpack [33]. Це забезпечує коректне відображення інтерфейсу на різних пристроях, включаючи настільні комп'ютери, планшети та смартфони. Крім того, фронтенд підтримує асинхронну взаємодію з сервером через API-запити (використовується Fetch API) та реалізує двосторонній зв'язок у реальному часі через Socket.IO для таких функцій, як чат.

Контекст аутентифікації (AuthContext) використовується для управління станом користувача, що дозволяє централізовано обробляти авторизацію, профіль і доступ до захищених маршрутів. Захищені маршрути (PrivateRoute) та публічні маршрути (PublicRoute) забезпечують контроль доступу до сторінок залежно від ролі користувача (клієнт або менеджер).

Бекенд вебзастосунку побудовано на основі фреймворку Express.js, який працює на платформі Node.js. Express.js забезпечує створення RESTful API для обробки запитів від клієнтської частини, таких як авторизація, реєстрація, управління товарами, замовленнями та категоріями [34]. API-ендпоінти, визначені за адресою http://localhost:3001/api, включають такі маршрути, як /login, /register, /products, /orders тощо.

Для зберігання даних використовується реляційна база даних SQLite, яка є легкою та ефективною для такого типу застосунків. SQLite забезпечує швидкий доступ до даних про користувачів, товари, замовлення та категорії. Безпека даних реалізується через бібліотеку bcryptjs для хешування паролів і jsonwebtoken (JWT) для аутентифікації користувачів. JWT-токени використовуються для перевірки авторизації при доступі до захищених ресурсів, таких як профіль користувача чи адміністративні функції.

Для реалізації зв'язку в реальному часі між клієнтом і сервером застосовується бібліотека Socket.IO. Вона дозволяє створювати інтерактивні функції, такі як чат між клієнтами та менеджерами, а також повідомлення про оновлення статусу замовлень. Socket.IO інтегрується з Express.js, що забезпечує єдину точку входу для HTTP-запитів і WebSocket-з'єднань.

Для зборки та оптимізації фронтенду використовується Webpack, який компілює JavaScript, CSS і інші ресурси в єдиний бандл (bundle.js). Webpack також підтримує гаряче перезавантаження (Hot Module Replacement) через devServer, що прискорює процес розробки. Babel забезпечує транспіляцію сучасного JavaScript (ES6+) для сумісності зі старими браузерами.

Лінтер ESLint інтегровано для забезпечення якості коду, а nodemon використовується для автоматичного перезапуску серверної частини під час розробки [35]. Проектна структура включає файли конфігурації, такі як package.json, webpack.config.js і eslint.config.js, що забезпечують стандартизацію та автоматизацію процесів розробки.

Архітектура вебзастосунку «Квіткова Лавка» побудована на клієнт-серверній моделі з чітким розподілом на фронтенд і бекенд. Фронтенд, реалізований за допомогою React і React Router, забезпечує адаптивний і інтерактивний інтерфейс користувача, тоді як бекенд на базі Express.js і SQLite обробляє бізнес-логіку та дані. Використання сучасних технологій, таких як Webpack, Socket.IO і JWT, дозволяє створити гнучку, безпечну та масштабовану систему, яка відповідає вимогам сучасного інтернет-магазину. Такий підхід забезпечує зручність для користувачів, ефективність розробки та можливість подальшого розширення функціоналу.

3.2. Вибір інструментів та технологій

Розробка адаптивного дизайну інтернет-магазину з реалізації квітів вимагає ретельного вибору інструментів і технологій, які забезпечують ефективну реалізацію функціоналу, зручність користувацького інтерфейсу та гнучкість для подальшого масштабування. У процесі створення програмного забезпечення для даного проєкту акцент робився на сучасних технологіях, які відповідають вимогам адаптивності, продуктивності та безпеки. У цьому підрозділі обґрунтовується вибір фреймворків для фронтенду, бекенд-

інструментарію, а також супутніх технологій, використаних для реалізації проєкту.

Для розробки фронтенду інтернет-магазину було обрано бібліотеку React, яка є однією з найпоширеніших і найефективніших технологій для створення сучасних веб-інтерфейсів. React забезпечує компонентно-орієнтований підхід до розробки, що дозволяє створювати модульні та повторно використовувані елементи інтерфейсу. Це особливо важливо для адаптивного дизайну, оскільки компоненти можна легко адаптувати до різних розмірів екрана за допомогою CSS-медіа-запитів або бібліотек, таких як CSS Grid чи Flexbox.

Використання React у проєкті обґрунтовується кількома ключовими перевагами, що описані в табл. 3.1.

Таблиця 3.1

Переваги React

№	Переваги	Пояснення
1	Висока продуктивність	завдяки віртуальному DOM, React мінімізує кількість операцій із реальним DOM, що забезпечує швидке оновлення інтерфейсу навіть при динамічних змінах даних, таких як оновлення кошика чи каталогу товарів
2	Гнучкість	бібліотека дозволяє інтегрувати додаткові інструменти, такі як React Router DOM для управління маршрутизацією, що використано в проєкті для навігації між сторінками (наприклад, каталог, кошик, профіль користувача)
3	Адаптивність	React підтримує створення адаптивних інтерфейсів через використання CSS-модулів або препроцесорів, що забезпечує коректне відображення на різних пристроях — від смартфонів до настільних комп’ютерів

Для обробки стилів у проєкті використано CSS із підтримкою style-loader і css-loader у конфігураVELOції Webpack, що дозволяє динамічно підключати стилі до компонентів. Це забезпечує модульність і полегшує підтримку адаптивного дизайну, оскільки стилі можна застосовувати вибірково залежно від контексту компонента. Крім того, використання Babel із пресетами @babel/preset-env і @babel/preset-react забезпечує сумісність із сучасними

браузерами та дозволяє писати код у JSX-форматі, що спрощує розробку компонентів.

Для управління маршрутизацією в додатку застосовано React Router DOM, що забезпечує плавну навігацію без перезавантаження сторінки. Це особливо важливо для створення зручного користувацького досвіду, оскільки користувачі можуть швидко переходити між розділами магазину, такими як головна сторінка, каталог чи сторінка оформлення замовлення. У коді видно використання компонентів BrowserRouter, Routes, Route, Link і Navigate, що підтверджує інтеграцію цього інструменту для реалізації клієнтської маршрутизації.

Для реалізації серверної частини проєкту було обрано Node.js у поєднанні з фреймворком Express. Цей вибір зумовлений кількома факторами (табл. 3.2), які відповідають вимогам проєкту.

Таблиця 3.2

Причини вибору Node.js у поєднанні з фреймворком Express

№	Причини	Пояснення
1	Продуктивність	Node.js базується на асинхронній моделі обробки запитів, що забезпечує високу швидкість обробки великої кількості одночасних запитів, наприклад, при перегляді каталогу товарів чи оформленні замовлень
2	Екосистема	Node.js має розвинену екосистему пакетів, доступних через npm, що спрощує інтеграцію додаткових інструментів, таких як jsonwebtoken для аутентифікації, bcryptjs для шифрування паролів чи socket.io для реалізації чату в реальному часі
3	Гнучкість	Express як легкий і мінімалістичний фреймворк дозволяє швидко налаштувати REST API для взаємодії фронтенду з бекендом. У проєкті API-ендпоінти, такі як /api/login, /api/register, /api/products, використовуються для обробки запитів користувачів, що підтверджує ефективність цього інструменту

Для зберігання даних обрано SQLite у поєднанні з модулем sqlite3. SQLite є легкою реляційною базою даних, яка ідеально підходить для проєктів середнього масштабу, таких як інтернет-магазин. Її переваги включають

простоту налаштування, відсутність необхідності в окремому сервері бази даних і достатню продуктивність для обробки запитів, пов’язаних із товарами, користувачами та замовленнями. У коді видно, що SQLite використовується для зберігання інформації про користувачів, товари, категорії та замовлення, що забезпечує стабільну роботу системи.

Для забезпечення безпеки аутентифікації та авторизації використано JSON Web Tokens (JWT). JWT дозволяє створювати токени для ідентифікації користувачів, які передаються в заголовках HTTP-запитів (наприклад, Authorization: Bearer <token>). Це забезпечує безпечний доступ до захищених маршрутів, таких як профіль користувача чи адмін-панель. Шифрування паролів здійснюється за допомогою bcryptjs, що гарантує захист конфіденційних даних користувачів.

Для забезпечення зв’язку в реальному часі, зокрема для функціоналу чату, використано Socket.IO. Ця бібліотека дозволяє встановлювати двостороннє з’єднання між клієнтом і сервером, що забезпечує миттєве оновлення повідомлень у чаті для клієнтів і менеджерів. У коді видно, що Socket.IO ініціалізується з токеном аутентифікації, що забезпечує безпечне підключення лише для авторизованих користувачів.

Для збирання та оптимізації фронтенд-коду використано Webpack. Webpack дозволяє об’єднувати JavaScript-файли, CSS-стилі та інші ресурси в єдиний бандл (bundle.js), що зменшує час завантаження сторінки. Конфігурація Webpack у проєкті включає підтримку Babel для транспіляції сучасного JavaScript і JSX, а також обробку CSS через відповідні лоадери. Крім того, devServer у Webpack забезпечує зручне середовище розробки з автоматичним перезавантаженням сторінки при зміні коду.

Для контролю якості коду використано ESLint, що допомагає виявляти потенційні помилки та забезпечує єдиний стиль кодування. Файл eslint.config.js налаштовано для перевірки синтаксису та відповідності стандартам, що підвищує читабельність і підтримуваність коду.

Вибір інструментів і технологій для розробки інтернет-магазину “Квіткова Лавка” базується на їх здатності забезпечити адаптивний дизайн, високу продуктивність і безпеку. React як фронтенд-фреймворк забезпечує модульність і гнучкість інтерфейсу, тоді як Node.js із Express і SQLite формують надійну серверну інфраструктуру. Додаткові інструменти, такі як Socket.IO, JWT, Webpack і ESLint, доповнюють технологічний стек, забезпечуючи повноцінну функціональність і зручність розробки. Цей набір технологій дозволяє створити сучасний, адаптивний і безпечний інтернет-магазин, який відповідає потребам користувачів і бізнес-вимогам.

3.3. Реалізація модулів вебзастосунку

Розробка вебзастосунку інтернет-магазину «Квіткова Лавка» передбачала створення модульної структури, яка забезпечує зручну взаємодію користувачів із платформою, адаптивність інтерфейсу та інтеграцію з базою даних. У цьому підрозділі розглянуто реалізацію ключових модулів вебзастосунку, зокрема головної сторінки, каталогу, картки товару, а також підходи до створення адаптивного дизайну та інтеграцію форми замовлення з базою даних.

Головна сторінка вебзастосунку, реалізована в компоненті Home (файл client.js), є центральним елементом інтерфейсу, який презентує основну інформацію про магазин (рис. 3.2) та його асортимент. Компонент побудовано з використанням бібліотеки React, що забезпечує динамічне оновлення вмісту без перезавантаження сторінки. Сторінка складається з трьох основних секцій, що описані в табл. 3.3.

Рис. 3.2 Блок з основною інформацією про магазин

Таблиця 3.3

Основні секції головної сторінки

№	Секції	Опис секцій
1	Секція Меню (рис. 3.3, рис. 3.4, рис. 3.5)	містить привітальний заголовок, короткий опис магазину та кнопку для переходу до каталогу товарів. Використовується компонент Link з бібліотеки react-router-dom для навігації
2	Секція категорій (рис. 3.6)	відображає перелік категорій товарів, отриманих через API-запит до ендпоінту /categories. Кожна категорія представлена карткою з назвою та зображенням, що формується динамічно за допомогою методу map
3	Секція популярних товарів (рис. 3.7)	демонструє до шести рекомендованих продуктів, отриманих через API-запит до /products?limit=6. Картки товарів містять зображення, назву та ціну, з можливістю переходу до детальної сторінки товару

Рис. 3.3 Секція меню для неавторизованого користувача

Рис. 3.4 Секція меню для авторизованого користувача

Рис. 3.5 Секція меню для адміністратора

Рис. 3.6 Секція категорій

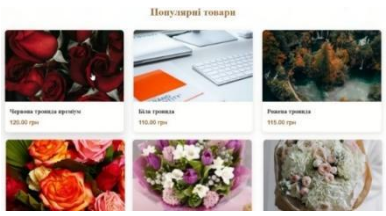

Рис. 3.7 Секція популярних товарів

Для забезпечення швидкого завантаження даних застосовується хук useEffect, який виконує асинхронні запити до сервера під час монтування компонента. Обробка помилок реалізована через конструкцію try-catch, що забезпечує стабільність роботи сторінки.

Модуль каталогу (рис. 3.8), представлений компонентом ProductList, забезпечує зручний перегляд та фільтрацію асортименту товарів.

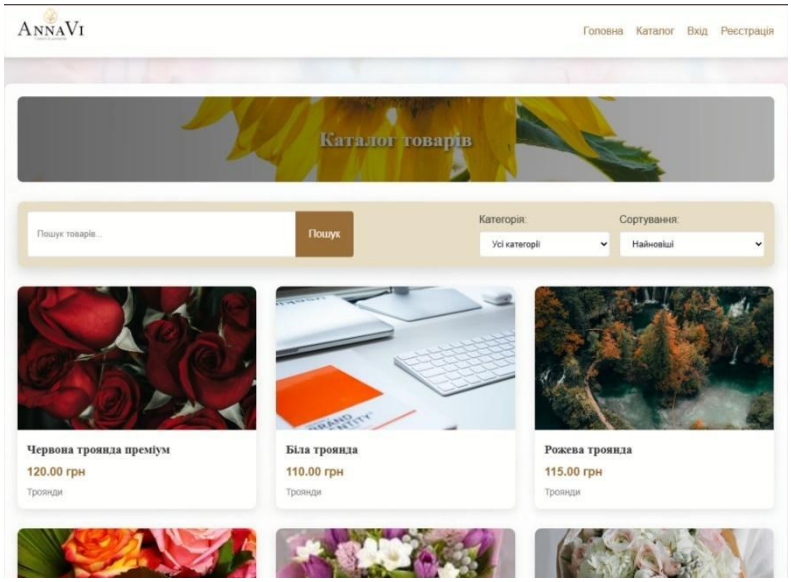

Рис.3.8 Каталог

Основні функціональні можливості каталогу включають:

- Фільтрація за категоріями (рис. 3.9), що дозволяє користувачу обирати категорію зі списку, отриманого через API-запит до /categories. Фільтр реалізується через елемент <select>, який оновлює стан компонента за допомогою хука useState.

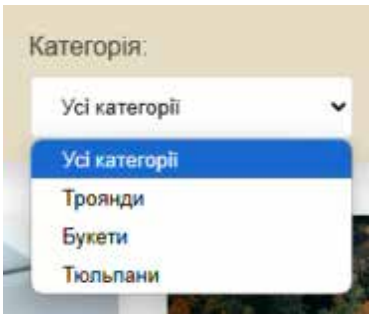

Рис. 3.9 Параметри фільтрації за категоріями

- Пошук товарів (рис. 3.10), що здійснюється за допомогою форми, що дозволяє вводити ключові слова. Введені дані передаються до API як параметр search, а запит формується динамічно через об’єкт URLSearchParams.

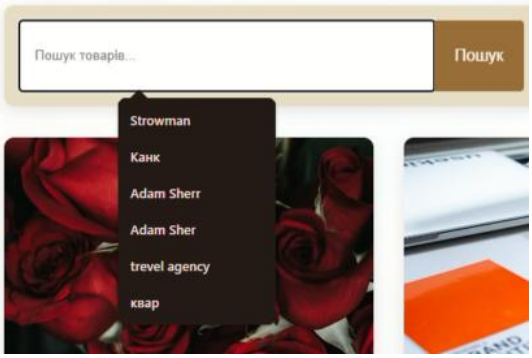

Рис. 3.10 Пошук товарів

- Сортування (рис. 3.11), що дозволяє користувачу впорядковувати товари за новизною, ціною (за зростанням чи спаданням) або назвою. Параметр сортування передається до API через запит.

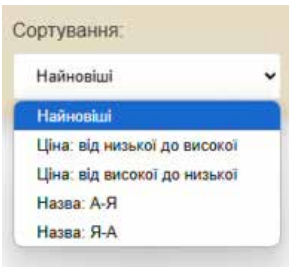

Рис. 3.11 Параметри сортування

Товари відображаються у вигляді сітки (products-grid), де кожна картка містить зображення, назву, ціну та категорію. Для адаптивності сітка використовує CSS-властивості grid-template-columns із медіа-запитами, що дозволяють змінювати кількість стовпців залежно від розміру екрана. Завантаження даних супроводжується відображенням індикатора «Завантаження...», що покращує користувацький досвід.

Картка товару (рис. 3.12), реалізована в компоненті ProductDetail, відображає детальну інформацію про конкретний продукт, отриману через API-запит до /products/:id.

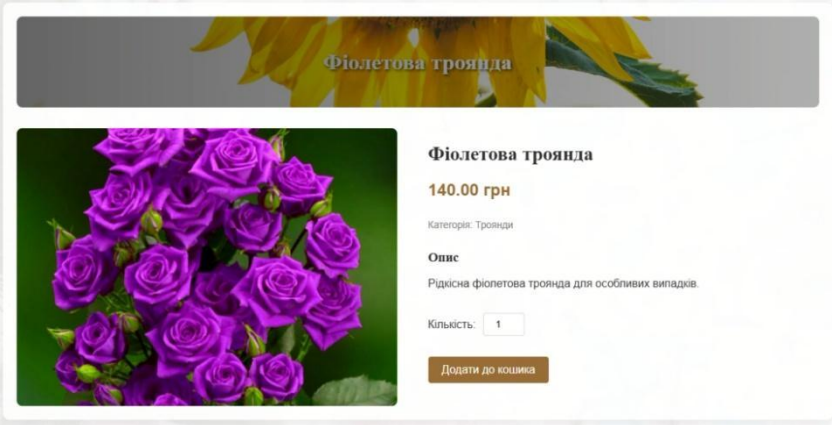

Рис. 3.12 Картка товару

Компонент включає:

- Зображення товару, що відображається через тег <img>, із використанням резервного зображення-заповнювача у випадку відсутності основного.

- Інформаційний блок, що містить назву, ціну, категорію та опис товару, а для форматування ціни застосовується метод toFixed(2), який відображає два знаки після коми.

- Функціонал додавання до кошика, що доступний лише для авторизованих користувачів (рис. 3.13, рис. 3.14) із роллю «client». Кількість товару регулюється через елемент <input type="number">, а оновлення кошика реалізується через localStorage і тригер події cartUpdated.

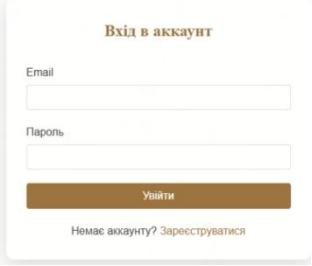

Рис. 3.13 Авторизація користувача для додавання товару до кошика

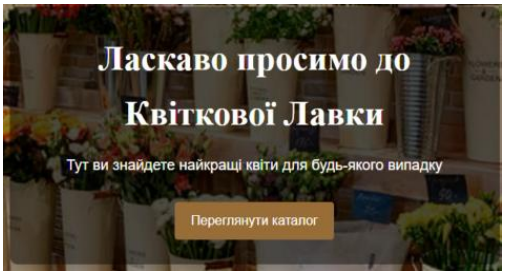

Рис. 3.14 Підтвердження авторизації

Для обробки помилок (наприклад, якщо товар не знайдено) передбачено відображення відповідного повідомлення. Адаптивність картки забезпечується за допомогою CSS-медіа-запитів, які змінюють розташування елементів (зображення та інформації) на вертикальне для мобільних пристроїв.

Адаптивний дизайн вебзастосунку реалізовано за допомогою CSS-медіа-запитів, які дозволяють адаптувати інтерфейс до різних розмірів екранів (десктоп, планшет, мобільний). Основні підходи описані в табл.3.4.

Таблиця 3.4

Основні підходи до адаптивного дизайну

№	Підходи	Пояснення
1	Гнучка сітка	для компонентів каталогу та головної сторінки використовується CSS Grid із динамічною кількістю стовпців. Наприклад, на десктопах відображається 3–4 стовпці, на планшетах — 2, а на мобільних — 1.
2	Медіа-запити	у CSS-файлах визначено правила для різних діапазонів ширини екрана (наприклад, @media (max-width: 768px)), які змінюють розміри шрифтів, відступи та розташування елементів
3	Мобільне меню	у компоненті Header реалізовано бургер-меню, яке активується на малих екранах. Стан меню керується через хук useState, а стилі для його відображення/приховування задано через CSS-клас open

Хоча фреймворки типу Bootstrap або Tailwind не використовувалися, підхід із власними CSS-стилями дозволив досягти гнучкості та оптимізувати

розмір стилів. Для підвищення доступності застосовано семантичну розмітку HTML5 та атрибути ARIA.

Форма замовлення реалізована в компоненті Cart, який відображає товари, додані до кошика, та дозволяє оформити замовлення. Основні елементи:

- Відображення кошика (рис. 3.15), що здійснюється через отримання товарів з localStorage, які відображаються у списку з можливістю зміни кількості або видалення. Загальна сума розраховується динамічно.

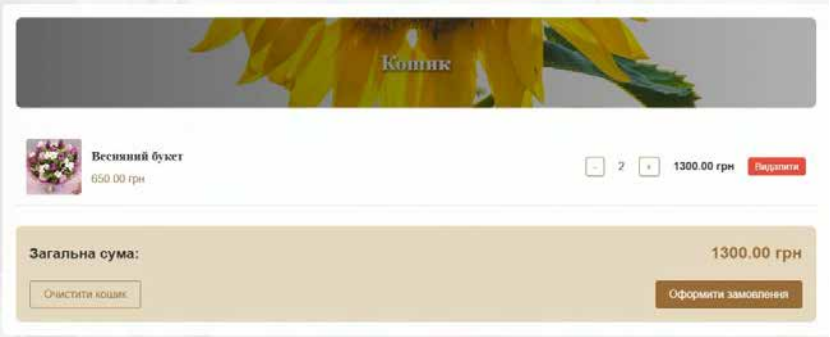

Рис. 3.15 Відображення кошика

- Оформлення замовлення, що виконується при натисканні кнопки «Оформити замовлення», що формує POST-запит до ендпоінту /orders із даними про товари, їх кількість та загальну суму. Запит включає токен авторизації для перевірки користувача.

- Інтеграція з базою даних, що реалізована через серверну частину на Express.js із використанням бази даних SQLite для обробки запитів на створення замовлення. Після успішного створення замовлення (рис. 3.16) виконується симуляція оплати через ендпоінт /payments/simulate, кошик очищається, а користувач перенаправляється на сторінку замовлень (рис. 3.17).

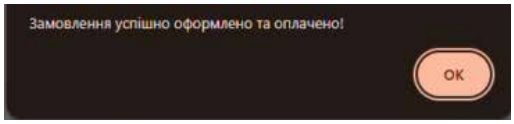

Рис. 3.16 Підтвердження успіху створення замовлення

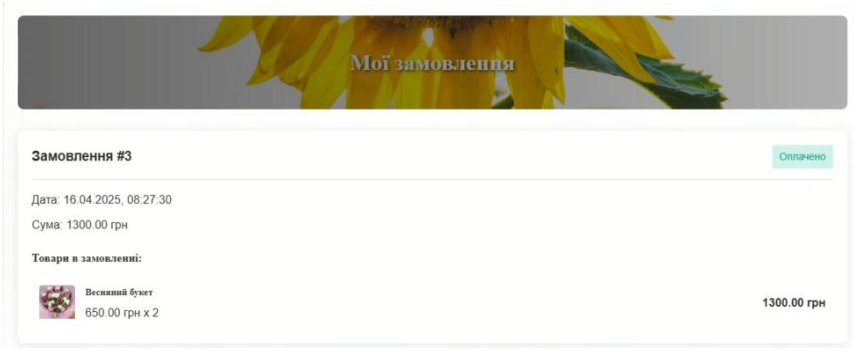

Рис. 3.17 Сторінка замовлень

- Обробка помилок, що відбувається через відображення відповідних повідомлень за допомогою елемента у випадку помилок, таких як порожній кошик або проблеми з оплатою.

Інтеграція з базою даних забезпечує збереження даних про замовлення, що дозволяє користувачам переглядати історію покупок у компоненті Orders.

Реалізація модулів вебзастосунку «Квіткова Лавка» охоплює створення функціональних компонентів для головної сторінки, каталогу, картки товару, а також інтеграцію форми замовлення з базою даних. Адаптивний дизайн, реалізований за допомогою CSS-медіа-запитів, забезпечує зручність використання на різних пристроях. Застосування сучасних технологій, таких як React та Express.js, дозволило створити ефективний та користувацьки орієнтований вебзастосунок, який відповідає вимогам сучасного e-commerce.

4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1. Тестування вебзастосунку

Тестування вебзастосунку є критично важливим етапом розробки, що дозволяє забезпечити його надійність, функціональність та відповідність вимогам користувачів. Для інтернет-магазину «Квіткова Лавка», реалізованого на основі коду проєкту, було проведено комплексне тестування, яке включало функціональне тестування та перевірку адаптивності дизайну на різних пристроях. Ці заходи спрямовані на гарантування якісного користувацького досвіду та стабільної роботи системи в реальних умовах.

Функціональне тестування вебзастосунку «Квіткова Лавка» проводилося з метою перевірки коректності роботи всіх основних функцій, передбачених вимогами до системи. Тестування охоплювало ключові модулі, такі як аутентифікація користувачів, управління каталогом товарів, кошик, оформлення замовлень, чат для комунікації та адміністративний інтерфейс. Для кожного модуля було розроблено набір тестових сценаріїв, що відображають типові дії користувачів, а також граничні випадки (табл. 4.1).

Таблиця 4.1

Тестові сценарії для функціонального тестування

№	Сценарій	Опис дій сценарію
1	2	3
1	Аутентифікація та авторизація	Перевірено можливість реєстрації нового користувача з ролями «клієнт» та «менеджер». Тестування включало введення коректних та некоректних даних (наприклад, невідповідність паролів або невалідний email). Успішна реєстрація завершувалася створенням токена та збереженням його в localStorage
		Перевірено функціонал входу в систему: успішний вхід із правильними обліковими даними, обробка помилок при введенні невірного пароля чи email, а також автоматичне перенаправлення на головну сторінку після успішного входу
		Тестувалася функція виходу з системи, що передбачала видалення токена та роз’єднання сокет-з’єднання, якщо воно було активним

Продовження табл. 4.1

1	2	3
2	Каталог товарів та фільтрація	Перевірено відображення списку товарів із застосуванням фільтрів за категоріями, пошуком та сортуванням (за ціною, назвою, новизною). Тестові сценарії включали перевірку коректності відображення товарів при порожньому пошуковому запиті та при відсутності товарів у певній категорії
		Перевірено сторінку деталей товару, зокрема відображення зображень, опису, ціни та доступності. Для авторизованих клієнтів тестувалася можливість додавання товару до кошика з різною кількістю
3	Кошик та оформлення замовлення	Перевірено додавання, оновлення кількості та видалення товарів із кошика. Тестові сценарії охоплювали перевірку правильності розрахунку загальної вартості та оновлення кількості товарів у заголовку (Header)
		Тестувалося оформлення замовлення, включаючи створення замовлення на сервері та симуляцію оплати. Перевірено обробку помилок, таких як порожній кошик або відсутність токена авторизації
4	Чат та адміністративний інтерфейс	Для чату перевірено встановлення сокет-з'єднання, відправлення та отримання повідомлень у реальному часі, а також коректність відображення повідомлень для клієнтів і менеджерів
		Для адміністративного інтерфейсу тестувалися функції управління товарами, категоріями та користувачами, доступні лише для ролі «менеджер». Перевірено обмеження доступу для неавторизованих користувачів або клієнтів

Функціональне тестування проводилося вручну з використанням тестових акаунтів для різних ролей. Для автоматизації частини тестів застосовувалися інструменти, такі як Jest для модульного тестування React-компонентів та Cypress для end-to-end тестування. Усі виявлені помилки, такі як некоректна обробка граничних значень у формах або тимчасові збої сокет-з'єднання, були задокументовані та виправлені.

Адаптивний дизайн є ключовою вимогою для сучасних вебзастосунків, оскільки користувачі взаємодіють із сайтом через різноманітні пристрої: смартфони, планшети та персональні комп'ютери. Для «Квіткової Лавки» перевірка адаптивності проводилася з урахуванням різних роздільних

здатностей екранів та типів пристроїв, щоб забезпечити зручність використання та візуальну цілісність інтерфейсу.

Методологія тестування передбачала використання інструментів розробника браузерів (Chrome DevTools) для симуляції різних розмірів екранів, таких як 360x640 для мобільних, 768x1024 для планшетів і 1920x1080 для ПК.

Тестування проводилось на фізичних пристроях, включаючи смартфони (iPhone 12, Samsung Galaxy S21), планшети (iPad Air) та настільні комп'ютери з браузерами Chrome, Firefox і Safari.

Відповідність дизайну принципам mobile-first перевірялася через використання в CSS гнучких одиниць вимірювання (%, vw, rem, em) та медіа-запитів для адаптації стилів.

Результати тестування на адаптивність приведені в табл. 4.2.

Таблиця 4.2

Результати тестування на адаптивність

№	Пристрій	Результати
1	Мобільні пристрої (рис. 4.1)	На екранах із шириною до 576 пікселів активується мобільне меню (гамбургер-меню), яке коректно відкривається та закривається. Елементи, такі як картки товарів та форми, адаптуються до вертикального розташування, забезпечуючи зручне прокручування. Виявлено незначну проблему з відображенням великих зображень у деталях товару, яка була виправлена шляхом оптимізації розмірів зображень
2	Планшети	На екранах із шириною 768–992 пікселів інтерфейс відображається у двоколонковому або одноколонковому форматі залежно від компонента. Наприклад, каталог товарів використовує сітку з двома колонками, що забезпечує оптимальне використання простору. Перевірено коректність роботи сенсорного введення, зокрема у формах та фільтрах
3	Персональні комп'ютери (рис. 4.2)	На великих екранах (понад 1200 пікселів) усі елементи відображаються у повнорозмірному вигляді з горизонтальним меню та багатоколоночною сіткою для каталогу. Перевірено, що шрифти, відступи та зображення масштабуються пропорційно, зберігаючи читабельність і естетичність

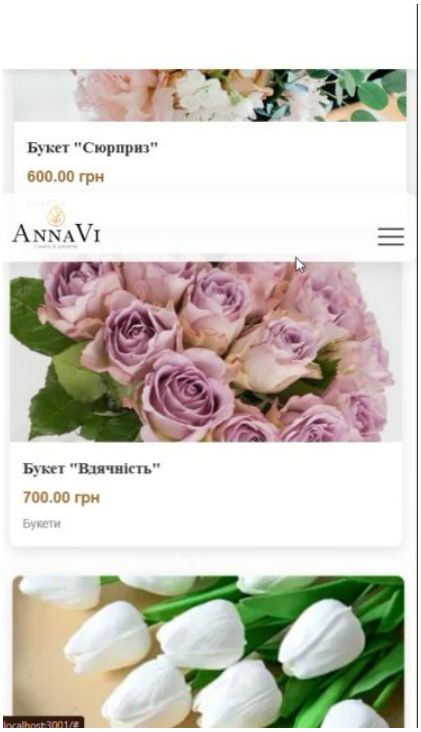

Рис. 4.1 Адаптивність під мобільні пристрої

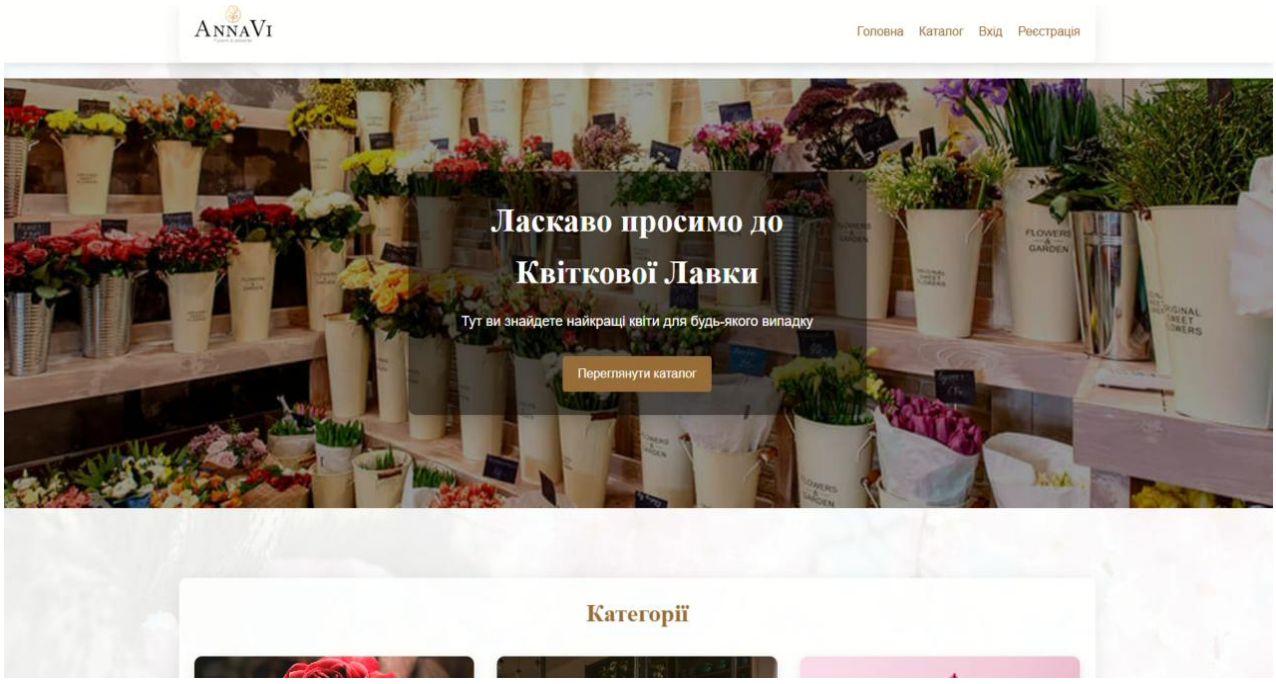

Рис. 4.2 Адаптивність під ПК

Для оцінки адаптивності використовувалися інструменти, такі як Lighthouse, які аналізували продуктивність, доступність та SEO. Зокрема, показник адаптивності склав 92/100 після оптимізації стилів.

Перевірено відповідність стандартам WCAG 2.1 для забезпечення доступності, зокрема контрастності тексту та навігації за допомогою клавіатури на всіх пристроях.

У процесі тестування адаптивності виявлено та усунуто проблеми, пов’язані з неправильним розташуванням елементів на малих екранах та надмірним масштабуванням зображень. Після внесення коригувань дизайн вебзастосунку став повністю адаптивним, забезпечуючи комфортну взаємодію для всіх категорій користувачів.

Проведене тестування вебзастосунку «Квіткова Лавка» підтвердило його функціональну стабільність та адаптивність до різних пристроїв. Функціональне тестування дозволило перевірити коректність роботи всіх ключових модулів, включаючи аутентифікацію, каталог, кошик та адміністративні функції. Перевірка адаптивності засвідчила, що інтерфейс коректно відображається на мобільних пристроях, планшетах і ПК, відповідаючи сучасним стандартам веброзробки. Усі виявлені недоліки були виправлені, що забезпечило високу якість користувацького досвіду та готовність системи до впровадження. Подальші етапи можуть включати стрес-тестування для оцінки продуктивності при високому навантаженні та розширення автоматизованих тестів для підвищення ефективності підтримки проєкту.

4.2. Вимоги до апаратного та програмного середовища

Для забезпечення стабільної роботи інтернет-магазину «Квіткова Лавка», розробленого з використанням кодів проєкту, необхідно чітко визначити вимоги до апаратного та програмного середовища. Ці вимоги охоплюють мінімальні технічні характеристики для хостингу системи, а також необхідне програмне забезпечення для запуску, розробки та тестування програми. Важливо розглянути ключові показники, які гарантують коректне функціонування системи, її масштабування та зручність для розробників і кінцевих користувачів.

Для хостингу інтернет-магазину «Квіткова Лавка» необхідно забезпечити серверне обладнання, яке відповідає потребам системи з урахуванням її клієнт-серверної архітектури, використання бази даних SQLite та інтеграції WebSocket через Socket.IO для реального часу. Основні мінімальні вимоги до апаратного забезпечення показані в табл. 4.3.

Таблиця 4.3

Мінімальні вимоги до апаратного забезпечення

№	Компонент	Вимоги
1	Процесор	Багатоядерний процесор (наприклад, 2 ядра з частотою 2.0 ГГц або вище). Це необхідно для обробки HTTP-запитів, управління WebSocket-з'єднаннями та виконання операцій із базою даних
2	Оперативна пам'ять (RAM)	Мінімум 4 ГБ для забезпечення стабільної роботи сервера Node.js, бази даних SQLite та одночасного обслуговування кількох користувачів. Для масштабування системи (наприклад, при збільшенні кількості відвідувачів) рекомендується 8 ГБ або більше
3	Дисковий простір	Не менше 20 ГБ вільного місця на SSD для зберігання операційної системи, програмного забезпечення, бази даних, зображень товарів та логів. SSD забезпечує швидший доступ до даних порівняно з HDD, що критично для швидкодії системи
4	Мережеве підключення	Стабільне інтернет-з'єднання зі швидкістю не менше 100 Мбіт/с для забезпечення швидкого обміну даними між клієнтами та сервером, особливо при використанні WebSocket для чату та оновлення даних у реальному часі
5	Операційна система	Сервер на базі Linux (наприклад, Ubuntu 20.04 або новішої версії) або Windows Server 2019. Linux є кращим вибором через його стабільність, безпеку та широку підтримку Node.js

Ці вимоги є мінімальними для хостингу системи на початковому етапі з невеликою кількістю користувачів (до 100 одночасних підключень). Для масштабування системи при зростанні аудиторії рекомендується використовувати хмарні платформи (наприклад, AWS, Google Cloud або DigitalOcean), які дозволяють динамічно розширювати ресурси, такі як обчислювальна потужність, пам'ять і дисковий простір.

Для запуску інтернет-магазину «Квіткова Лавка» необхідно встановити та налаштувати відповідне програмне забезпечення, яке забезпечує функціонування серверної та клієнтської частин системи. Основні компоненти ПЗ описані в табл. 4.4.

Таблиця 4.4

Вимоги до програмного забезпечення для запуску інтернет-магазину

№	Компонент	Вимоги
1	Node.js (версія 18.x або новіша)	Використовується як основне середовище виконання для серверної частини (файл server.js) та клієнтських скриптів. Node.js забезпечує асинхронну обробку запитів і підтримку WebSocket через бібліотеку Socket.IO
2	NPM (Node Package Manager)	Входить до складу Node.js і використовується для встановлення залежностей, зазначених у файлі package.json, таких як express, socket.io, jsonwebtoken, bcryptjs, sqlite3 тощо
3	SQLite (версія 3.x)	Легка реляційна база даних, яка використовується для зберігання інформації про користувачів, товари, замовлення та категорії. SQLite не вимагає окремого серверного процесу, що спрощує розгортання
4	Веб-сервер	Express.js, який є частиною залежностей проєкту, виступає в ролі веб-сервера для обробки HTTP-запитів і маршрутизації
5	Браузер	Для клієнтської частини системи (React-додаток) потрібен сучасний веб-браузер, такий як Google Chrome, Mozilla Firefox, Microsoft Edge або Safari (версії не старші за 2023 рік), із підтримкою JavaScript та WebSocket

Для розробки, тестування та підтримки системи розробникам необхідно встановити додаткове програмне забезпечення, яке забезпечує зручну роботу з кодом, налагодження та контроль якості. На основі кодів проєкту і конфігураційних файлів можна визначити вимоги, що описані в табл. 4.5.

Таблиця 4.5

Вимоги до програмного забезпечення для розробки

№	Компонент	Вимоги
1	Текстові редактори або IDE	Рекомендується використовувати Visual Studio Code, WebStorm або аналогічні середовища з підтримкою JavaScript, React і Node.js. Visual Studio Code, зокрема, підтримує інтеграцію з ESLint для перевірки якості коду
2	Webpack (версія 5.x)	Використовується для збирання клієнтської частини додатка, як зазначено у файлі webpack.config.js. Webpack забезпечує модульне збирання JavaScript, CSS і інших ресурсів у єдиний бандл (bundle.js)
3	Babel	Необхідний для трансформації сучасного JavaScript-коду (ES6+) у формат, сумісний зі старими браузерами. Налаштування Babel визначено в webpack.config.js через пресети @babel/preset-env і @babel/preset-react
4	ESLint	Використовується для статичного аналізу коду та забезпечення відповідності стандартам кодування. Конфігурація ESLint міститься у файлі eslint.config.js
5	Concurrently	Дозволяє одночасно запускати кілька процесів (наприклад, сервер Node.js і клієнтський дев-сервер), як зазначено в скрипті dev у package.json
6	Nodemon	Забезпечує автоматичний перезапуск сервера при зміні вихідного коду, що прискорює процес розробки
7	Git	Система контролю версій для управління кодом і спільної роботи команди розробників
8	Браузерні інструменти розробника	Вбудовані інструменти браузерів (наприклад, Chrome DevTools) для налагодження клієнтської частини, аналізу мережевих запитів і тестування WebSocket-з’єднань

Для забезпечення безпеки та стабільності системи рекомендується використовувати зворотний проксі-сервер, такий як Nginx, для балансування навантаження, обробки статичних файлів і захисту від DDoS-атак. Також слід налаштувати HTTPS за допомогою сертифікатів SSL/TLS (наприклад, через Let’s Encrypt) для шифрування даних, що передаються між клієнтом і сервером. Для моніторингу продуктивності та логування можна використовувати інструменти, такі як PM2 для управління процесами Node.js або Prometheus для збору метрик.

Описані вимоги до апаратного та програмного середовища забезпечують коректне розгортання, запуск і розробку інтернет-магазину «Квіткова Лавка».

Мінімальні апаратні вимоги дозволяють запустити систему на бюджетних серверах або хмарних платформах, тоді як рекомендовані параметри забезпечують масштабування при зростанні навантаження. Програмне забезпечення, включаючи Node.js, SQLite, Webpack і ESLint, створює гнучке та ефективне середовище для розробки й експлуатації. Дотримання цих вимог гарантує стабільність, безпеку та зручність використання системи як для користувачів, так і для розробників.

4.3. Підготовка системи до впровадження

Підготовка системи до впровадження є ключовим етапом життєвого циклу розробки програмного забезпечення, що забезпечує безперебійне розгортання, стабільну роботу та зручність використання кінцевим користувачам і адміністраторам. Для інтернет-магазину «Квіткова Лавка» підготовка до впровадження включає створення інсталяційного пакету або розгортання на хостингу, а також розробку детальних інструкцій для користувачів і адміністраторів. Слід детально розглянути ці показники, враховуючи специфіку системи та її адаптивний дизайн.

Інтернет-магазин «Квіткова Лавка» побудовано з використанням сучасного технологічного стеку, що включає фронтенд на базі React.js із React Router для навігації, бекенд на основі Express.js, базу даних SQLite та WebSocket для реального часу взаємодії через Socket.IO. Для забезпечення гнучкості розгортання системи передбачено два основні підходи: створення інсталяційного пакету для локального розгортання та деплой на хостинг для хмарного використання.

Для локального розгортання підготовлено інсталяційний пакет, який включає всі необхідні компоненти системи. Процес створення пакету передбачає наступні кроки:

- Збірка фронтенду виконується за допомогою Webpack, який компілює React-компоненти у статичний файл bundle.js, як описано у файлі

webpack.config.js. . Команда npm run build генерує оптимізовані статичні файли в директорії dist.

- Підготовка бекенду охоплює реалізацію у server.js, упаковану разом із залежностями з package.json, а також створення шаблону бази даних SQLite із початковими таблицями для користувачів, продуктів, категорій і замовлень.

- Автоматизація інсталяції забезпечується через скрипт для автоматичного встановлення залежностей (npm install) та ініціалізації бази даних. Інсталяційний пакет містить документацію з вимогами до середовища (Node.js версії 18+, npm) і покрокові інструкції.

- Конфігурація включає можливість налаштування базового URL API (http://localhost:3001/api) та параметрів підключення до Socket.IO через відповідний файл конфігурації.

Для хмарного розгортання система адаптована до платформ типу Heroku, Vercel або AWS. Процес деплою включає:

- Фронтенд передбачає розміщення статичних файлів, згенерованих Webpack, на Vercel або Netlify, що гарантує швидке завантаження та адаптивний дизайн для різних пристроїв.

- Бекенд реалізується через сервер Express.js, розгорнутий на Heroku або AWS EC2, із використанням Dockerfile для контейнеризації або Heroku Procfile. SQLite замінюється на PostgreSQL для забезпечення масштабованості у хмарному середовищі.

- Синхронізація включає налаштування CORS та Socket.IO для безпечної взаємодії між фронтендом і бекендом, а змінна середовища використовується для динамічного визначення API_URL.

- Безпека забезпечується впровадженням HTTPS, захистом JWT-токенів та обмеженням доступу до адміністративних маршрутів (/admin/*) для ролі «manager».

Обидва підходи протестовано для забезпечення стабільності. Локальний пакет перевірено на Windows та Linux, а хмарне розгортання — на Vercel

(фронтенд) та Heroku (бекенд). Адаптивний дизайн, реалізований через CSS-медіа-запити, забезпечує коректне відображення на пристроях із роздільною здатністю від 320 пікселів (мобільні) до 1920 пікселів (десктопи).

Для забезпечення зручності використання системи розроблено детальні інструкції, адаптовані до потреб двох основних груп: кінцевих користувачів (клієнтів) та адміністраторів (менеджерів). Інструкції надаються у вигляді PDF-документів та вбудованих підказок у інтерфейсі користувача.

Клієнти інтернет-магазину «Квіткова Лавка» отримують доступ до каталогу продуктів, кошика, профілю та чату. Інструкція включає:

- Реєстрація та вхід передбачають покроковий опис процесу створення облікового запису (/register) та авторизації (/login), з акцентом на використанні валідної електронної пошти та пароля довжиною не менше 8 символів.

- Навігація каталогом включає опис фільтрів у компоненті ProductList, таких як пошук, категорії та сортування, а також перегляд деталей товару (/products/:id), де адаптивний дизайн забезпечує зручність на мобільних пристроях.

- Оформлення замовлення охоплює додавання товарів до кошика (Cart) і оформлення замовлення з імітацією оплати, із поясненням перевірки статусу замовлення в розділі «Замовлення» (/orders).

- Чат описує функціонал звернення до менеджера через сторінку (/chat) з відображенням повідомлень у реальному часі за допомогою Socket.IO.

- Оновлення профілю передбачає редагування особистих даних (/profile), таких як ім’я, телефон та адреса.

Адміністратори мають розширений доступ до управління продуктами, категоріями та користувачами. Інструкція охоплює:

- Доступ до адмін-панелі описує процес входу з роллю «manager» та навігацію до маршрутів /admin/products, /admin/categories, /admin/users.

- Управління продуктами передбачає покроковий процес додавання, редагування та видалення товарів із завантаженням зображень і встановленням цін.

- Управління категоріями включає інструкції зі створення та редагування категорій для структуризації каталогу.

- Модерація користувачів охоплює функції перегляду списку користувачів, зміну їх ролей і блокування облікових записів.

- Моніторинг чату пояснює, як відповідати на запити клієнтів у реальному часі за допомогою Socket.IO.

Інструкції містять скріншоти інтерфейсу, що полегшують орієнтацію, та приклади типових сценаріїв використання. Для клієнтів акцентовано на простоті та інтуїтивності, для адміністраторів — на безпеці та ефективності управління. Адаптивний дизайн інтерфейсу забезпечує зручність роботи як на настільних комп’ютерах, так і на мобільних пристроях.

Підготовка системи «Квіткова Лавка» до впровадження проведена з урахуванням сучасних вимог до розгортання та експлуатації веб-додатків. Інсталяційний пакет для локального використання та деплой на хостинг забезпечують гнучкість вибору середовища. Детальні інструкції для клієнтів і адміністраторів сприяють швидкому освоєнню системи, а адаптивний дизайн гарантує зручність використання на різних пристроях. Ці заходи створюють міцну основу для успішного впровадження та подальшої експлуатації інтернет-магазину.

ВИСНОВКИ

Проведена розробка адаптивного вебзастосунку для інтернет-магазину «Квіткова Лавка» стала важливим кроком у створенні сучасної платформи для реалізації квітів, що відповідає актуальним вимогам електронної комерції. Основним результатом проєкту є функціональний вебзастосунок, який забезпечує зручну взаємодію користувачів із платформою через інтуїтивно зрозумілий інтерфейс, адаптивний до різних пристроїв, включаючи смартфони, планшети та настільні комп'ютери. Реалізовані ключові модулі, такі як каталог товарів, кошик, оформлення замовлень, профіль користувача, інтерактивний чат і адміністративна панель, дозволяють ефективно автоматизувати процеси продажу та управління. Використання сучасного технологічного стеку, зокрема React для клієнтської частини, Node.js із Express для серверної, SQLite як бази даних, а також Socket.IO для зв'язку в реальному часі, забезпечило високу продуктивність, модульність і безпеку системи. Адаптивний дизайн, реалізований через CSS-медіазапити, гнучкі сітки та mobile-first підхід, гарантує комфортне використання платформи незалежно від розміру екрана, що підтверджено тестуванням на різноманітних пристроях.

Ефективність запропонованих підходів до створення адаптивного дизайну та функціоналу підтверджується результатами тестування, які засвідчили стабільну роботу всіх модулів і відповідність системи заявленим вимогам. Використання React дозволило досягти швидкого оновлення інтерфейсу завдяки віртуальному DOM, що особливо важливо для динамічних компонентів, таких як каталог і кошик. Node.js із Express забезпечив асинхронну обробку запитів, що сприяло високій швидкості відгуку системи навіть при одночасній роботі кількох користувачів. SQLite, попри свою простоту, виявилася оптимальним вибором для прототипу, забезпечуючи достатню функціональність для реляційних даних із мінімальними витратами на розгортання. Інтеграція JSON Web Tokens і bcryptjs гарантувала безпечну аутентифікацію, а Socket.IO додав цінну функціональність у вигляді чату в реальному часі, що підвищує якість

обслуговування клієнтів. Адаптивність дизайну, підтверджена тестами на різних роздільних здатностях, відповідає сучасним стандартам веброзробки, включаючи принципи WCAG для доступності. Водночас аналіз аналогів, таких як Bloom & Wild, Interflora, UFL і Kvitochka, показав, що розроблена система має міцну основу для конкуренції, хоча потребує додаткової оптимізації, зокрема в плані швидкості завантаження зображень і розширення UI/UX-функціоналу.

Перспективи подальшого розвитку системи пов’язані з її масштабуванням і вдосконаленням для ширшої аудиторії. У майбутньому доцільно розглянути перехід на потужнішу СУБД, наприклад PostgreSQL, для підтримки більшого обсягу даних і одночасних підключень. Інтеграція з платіжними системами, такими як LiqPay чи Stripe, дозволить реалізувати реальні транзакції, підвищуючи комерційну цінність платформи. Розширення функціоналу, наприклад, додавання рекомендаційних алгоритмів на основі попередніх покупок або персоналізованих пропозицій, може покращити користувацький досвід. Впровадження анімацій, оптимізація зображень і підтримка клавіатурної навігації сприятимуть підвищенню доступності та привабливості інтерфейсу. Презентація результатів на профільних конференціях і публікація тез, заплановані на майбутнє, сприятимуть популяризації проєкту та залученню нових користувачів. Таким чином, створений вебзастосунок є надійною основою для розвитку конкурентоспроможного інтернет-магазину, який має потенціал для розширення функціональності та ринкової присутності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Як використовувати JSON Web Tokens (JWT) для автентифікації URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi
2. Створіть свій чат за допомогою Node.js та Socket.IO (найпростіший спосіб) URL: https://javascript.org.ua/stvorit-svij-chat-za-dopomogoyu-node-js-ta-socket-io-najprostishij-sposib/
3. Огляд CSS Grid та Flexbox: коли та як використовувати URL: https://mate.academy/blog/front-end-and-js/css-grid-and-flexbox/
4. Що таке Webpack? URL: https://foxminded.ua/webpack/
5. Bloom & Wild URL: https://www.bloomandwild.com/
6. Interflora URL: https://www.interflora.pl/
7. UFL URL: https://www.u-f-l.net/
8. Kvitochka URL: https://kvitochka.ua/uk
9. What are the best practices for fluid grid layout in responsive web design? URL: https://www.linkedin.com/advice/3/what-best-practices-fluid-grid-layout-responsive-web-6hoyc
10. Responsive Web Design - Media Queries URL: https://www.w3schools.com/css/css_rwd_mediaqueries.asp
11. Mobile First Design: What it is and How to Implement it URL: https://www.browserstack.com/guide/how-to-implement-mobile-first-design
12. Progressive Enhancement URL: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement
13. UX/UI дизайн інтернет-магазину 2025: як створити зручний та привабливий сайт URL: https://coi.ua/blog/DesignCo/ux-ui-design-for-e-commerce-in-2025-how-to-create-a-user-friendly-and-attractive-website/
14. Діаграми Прецедентів (Use Case UML Diagram) URL: https://lvivqaclub.blogspot.com/2008/10/use-case-uml-diagram.html

15. Що таке діаграма класів UML і найкращий творець діаграм класів UML URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/

16. Логічна модель даних URL: https://uk.wikipedia.org/wiki/Логічна_модель_даних

17. Що таке СУБД і для чого вони потрібні URL: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/

18. Що таке MySQL URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-mysql/

19. Що таке PostgreSQL? URL: https://itvdn.com/ua/shares/postgresql-for-free

20. MongoDB URL: https://cyberdev.space/docs/tutorials/databases/NoSQL/mongo/

21. Що таке SQlite? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/

22. Що означає ACID принцип у програмуванні? URL: https://tseivo.com/b/memecode/t/jnz2vv2xjn/shcho-oznachaie-acid-pryntsyp-u-prohramuvanni

23. Моделювання даних (Data Modelling) URL: https://www.maxzosim.com/data-modelling/

24. Оксана Мулеса. ІНФОРМАЦІЙНІ СИСТЕМИ ТА РЕЛЯЦІЙНІ БАЗИ ДАНИХ. Навчальний посібник для студентів математичних спеціальностей. Ужгород. 2018

25. Зв’язки в базах даних: що це і як вони працюють URL: https://foxminded.ua/zviazok-u-bazi-danykh/

26. Foreign key constraints: When to use ON UPDATE and ON DELETE URL: https://stackoverflow.com/questions/6720050/foreign-key-constraints-when-to-use-on-update-and-on-delete

27. Архітектура вебзастосунків 2024: Ультимативний гайд для розробників URL: https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv

28. Socket-IO для початківців URL: https://javascript.org.ua/socket-io-dlya-pochatkivcziv-%F0%9F%94%8C/

29. Клієнт-серверна архітектура URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

30. Огляд популярних JavaScript фреймворків URL: https://foxminded.ua/freimvorky-javascript/

31. Як стати React розробником. Що потрібно знати та вміти – з нуля до рівня спеціаліста URL: https://itvdn.com/ua/blog/article/how-to-become-a-react-developer

32. Роутинг у React URL: https://blog.ithillel.ua/articles/routing-in-react

33. Webpack style-loader vs css-loader URL: https://stackoverflow.com/questions/34039826/webpack-style-loader-vs-css-loader

34. Основи роботи з фреймворком Express.js URL: https://foxminded.ua/express-js/

35. Як ESLint допомагає забезпечити якість коду URL: https://foxminded.ua/eslint-shcho-tse/

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

client.js

```
const { useState, useEffect, createContext, useContext } = React;
// Спробуємо отримати доступ до об'єктів React Router
console.log('ReactRouter доступний:', !!window.ReactRouter);
console.log('ReactRouterDOM доступний:', !!window.ReactRouterDOM);
// Використовуємо безпечно з перевірками
const BrowserRouter = window.ReactRouterDOM?.BrowserRouter;
const Routes = window.ReactRouterDOM?.Routes;
const Route = window.ReactRouterDOM?.Route;
const Link = window.ReactRouterDOM?.Link;
const Navigate = window.ReactRouterDOM?.Navigate;
const useNavigate = window.ReactRouterDOM?.useNavigate;
// Перевіримо, що всі необхідні компоненти доступні
if (!BrowserRouter || !Routes || !Route || !Link || !Navigate || !useNavigate) {
    console.error('React Router DOM компоненти не доступні. Перевірте імпорти в
HTML.');
}
// Для socket.io
const io = window.io;
// Базовий URL API
const API_URL = 'http://localhost:3001/api';
// Контекст аутентифікації
const AuthContext = React.createContext();
// Провайдер аутентифікації
function AuthProvider({ children }) {
    const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);
    const [token, setToken] = useState(localStorage.getItem('token'));
    const [socket, setSocket] = useState(null);
    useEffect(() => {
        // Встановлення з'єднання через сокет
        if (token) {
            const newSocket = io('http://localhost:3001');
            newSocket.emit('authenticate', token);
            setSocket(newSocket);
            return () => {
                newSocket.disconnect();
            };
        }
    }, [token]);
    useEffect(() => {
        // При зміні токена перевіряємо користувача
        if (token) {
            localStorage.setItem('token', token);
            fetchUser();
        } else {
            localStorage.removeItem('token');
            setUser(null);
            setLoading(false);
        }
    }, [token]);
    const fetchUser = async () => {
        try {
            const response = await fetch(`${API_URL}/me`, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
```

```
            if (response.ok) {
                const userData = await response.json();
                setUser(userData);
            } else {
                setToken(null);
            }
        } catch (error) {
            console.error('Помилка при отриманні даних користувача:', error);
            setToken(null);
        } finally {
            setLoading(false);
        }
    };
    const login = async (email, password) => {
        try {
            const response = await fetch(`${API_URL}/login`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ email, password })
            });
            const data = await response.json();
            if (response.ok) {
                setToken(data.token);
                return { success: true };
            } else {
                return { success: false, message: data.message };
            }
        } catch (error) {
            console.error('Помилка при вході:', error);
            return { success: false, message: 'Помилка при вході. Спробуйте пізніше.'
};
        }
    };
    const register = async (name, email, password, role = 'client') => {
        try {
            const response = await fetch(`${API_URL}/register`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ name, email, password, role })
            });
            const data = await response.json();
            if (response.ok) {
                setToken(data.token);
                return { success: true };
            } else {
                return { success: false, message: data.message };
            }
        } catch (error) {
            console.error('Помилка при реєстрації:', error);
            return { success: false, message: 'Помилка при реєстрації. Спробуйте
пізніше.' };
        }
    };    const logout = () => {
        setToken(null);
        if (socket) {
            socket.disconnect();
            setSocket(null);
        }
    };
    const updateProfile = async (userData) => {
        try {
```

```
            const response = await fetch(`${API_URL}/users/profile`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify(userData)
            });
            const data = await response.json();
            if (response.ok) {
                fetchUser(); // Оновлюємо дані користувача
                return { success: true };
            } else {
                return { success: false, message: data.message };
            }
        } catch (error) {
            console.error('Помилка при оновленні профілю:', error);
            return { success: false, message: 'Помилка при оновленні профілю.
Спробуйте пізніше.' };
        }
    };
    return (
        <AuthContext.Provider value={{
            user,
            loading,
            token,
            socket,
            login,
            register,
            logout,
            updateProfile
        }}>
            {children}
        </AuthContext.Provider>
    );
}
// Хук для використання контексту аутентифікації
function useAuth() {
    return React.useContext(AuthContext);
}
// Компонент для захищених маршрутів
function PrivateRoute({ element, roles = [] }) {
    const { user, loading } = useAuth();
    if (loading) {
        return <div className="loading">Завантаження...</div>;
    }
    if (!user) {
        return <Navigate to="/login" />;
    }
    if (roles.length > 0 && !roles.includes(user.role)) {
        return <Navigate to="/" />;
    }
    return element;
}

// Компонент для незахищених маршрутів (для неавторизованих користувачів)
function PublicRoute({ element, restricted = false }) {
    const { user, loading } = useAuth();
   if (loading) {
        return <div className="loading">Завантаження...</div>;
    }
    if (user && restricted) {
        return <Navigate to="/" />;
    }
    return element;
}
```

```
// Головний компонент App
function App() {
    return (
        <Router>
            <AuthProvider>
                <div className="app">
                    <Header />
                    <main className="main-content">
                        <Routes>
                            <Route path="/" element={<Home />} />
                            <Route path="/login" element={<PublicRoute element={<Login
/>} restricted={true} />} />
                            <Route path="/register" element={<PublicRoute
element={<Register />} restricted={true} />} />
                            <Route path="/products" element={<ProductList />} />
                            <Route path="/products/:id" element={<ProductDetail />} />
                            <Route path="/cart" element={<PrivateRoute element={<Cart
/>} roles={['client']} />} />
                            <Route path="/profile" element={<PrivateRoute
element={<Profile />} />} />
                            <Route path="/orders" element={<PrivateRoute
element={<Orders />} roles={['client', 'manager']} />} />
                            <Route path="/chat" element={<PrivateRoute element={<Chat
/>} roles={['client', 'manager']} />} />
                            <Route path="/admin/products" element={<PrivateRoute
element={<AdminProducts />} roles={['manager']} />} />
                            <Route path="/admin/categories" element={<PrivateRoute
element={<AdminCategories />} roles={['manager']} />} />
                            <Route path="/admin/users" element={<PrivateRoute
element={<AdminUsers />} roles={['manager']} />} />
                            <Route path="*" element={<NotFound />} />
                        </Routes>
                    </main>
                    <Footer />
                </div>
            </AuthProvider>
        </Router>
    );
}
// Компонент Header
function Header() {
    const { user, logout } = useAuth();
    const [isOpen, setIsOpen] = useState(false);
    const [cartCount, setCartCount] = useState(0);
    useEffect(() => {
        if (user && user.role === 'client') {
            // Завантажуємо кількість товарів у кошику
            const cart = JSON.parse(localStorage.getItem('cart') || '[]');
            setCartCount(cart.reduce((total, item) => total + item.quantity, 0));
        }
    }, [user]);
    const toggleMenu = () => {
        setIsOpen(!isOpen);
    };
    return (
        <header className="header">
            <div className="container header-container">
                <div className="logo">
                    <Link to="/">Квіткова Лавка</Link>
                </div>
                <button className="menu-toggle" onClick={toggleMenu}>
                    <span></span>
                    <span></span>
                    <span></span>
                </button>
```

```
                <nav className={`nav ${isOpen ? 'open' : ''}`}>
                    <ul className="nav-list">
                        <li className="nav-item">
                            <Link to="/" onClick={() =>
setIsOpen(false)}>Головна</Link>
                        </li>
                        <li className="nav-item">
                            <Link to="/products" onClick={() =>
setIsOpen(false)}>Каталог</Link>
                        </li>
                        {user ? (
                            <>
                                {user.role === 'client' && (
                                    <li className="nav-item">
                                        <Link to="/cart" onClick={() =>
setIsOpen(false)}>
                                            Кошик {cartCount > 0 && <span
className="cart-badge">{cartCount}</span>}
                                        </Link>
                                    </li>
                                )}
                                <li className="nav-item">
                                    <Link to="/profile" onClick={() =>
setIsOpen(false)}>Профіль</Link>
                                </li>
                                <li className="nav-item">
                                    <Link to="/orders" onClick={() =>
setIsOpen(false)}>Замовлення</Link>
                                </li>
                                <li className="nav-item">
                                    <Link to="/chat" onClick={() =>
setIsOpen(false)}>Чат</Link>
                                </li>
                                {user.role === 'manager' && (
                                    <li className="nav-item dropdown">
                                        <span className="dropdown-toggle">Адмін</span>
                                        <ul className="dropdown-menu">
                                            <li>
                                                <Link to="/admin/products" onClick={()
=> setIsOpen(false)}>Товари</Link>
                                            </li>
                                            <li>
                                                <Link to="/admin/categories"
onClick={() => setIsOpen(false)}>Категорії</Link>
                                            </li>
                                            <li>
                                                <Link to="/admin/users" onClick={() =>
setIsOpen(false)}>Користувачі</Link>
                                            </li>
                                        </ul>
                                    </li>
                                )}
                                <li className="nav-item">
                                    <button className="btn-link" onClick={() => {
logout(); setIsOpen(false); }}>Вихід</button>
                                </li>
                            </>
                        ) : (
                            <>
                                <li className="nav-item">
                                    <Link to="/login" onClick={() =>
setIsOpen(false)}>Вхід</Link>
                                </li>
                                <li className="nav-item">
```

```
                                    <Link to="/register" onClick={() =>
setIsOpen(false)}>Реєстрація</Link>
                                </li>
                            </>
                        )}
                    </ul>
                </nav>
            </div>
        </header>
    );
}
// Компонент Footer
function Footer() {
    return (
        <footer className="footer">
            <div className="container">
                <div className="footer-content">
                    <div className="footer-section">
                        <h3>Квіткова Лавка</h3>
                        <p>Ваш надійний партнер у світі квітів. Ми допоможемо зробити
будь-який день особливим!</p>
                    </div>
                    <div className="footer-section">
                        <h3>Контакти</h3>
                        <p>Телефон: +380 50 123 4567</p>
                        <p>Email: info@flower-shop.ua</p>
                        <p>Адреса: м. Київ, вул. Квіткова, 12</p>
                    </div>
                    <div className="footer-section">
                        <h3>Соціальні мережі</h3>
                        <div className="social-links">
                            <a href="#" className="social-link">Facebook</a>
                            <a href="#" className="social-link">Instagram</a>
                            <a href="#" className="social-link">Telegram</a>
                        </div>
                    </div>
                </div>
                <div className="footer-bottom">
                    <p>&copy; {new Date().getFullYear()} Квіткова Лавка. Усі права
захищені.</p>
                </div>
            </div>
        </footer>
    );
}
// Компонент Home
function Home() {
    const [categories, setCategories] = useState([]);
    const [featuredProducts, setFeaturedProducts] = useState([]);
    useEffect(() => {
        // Завантаження категорій
        fetch(`${API_URL}/categories`)
            .then(response => response.json())
            .then(data => setCategories(data))
            .catch(error => console.error('Помилка при завантаженні категорій:',
error));
        // Завантаження вибраних товарів
        fetch(`${API_URL}/products?limit=6`)
            .then(response => response.json())
            .then(data => setFeaturedProducts(data))
            .catch(error => console.error('Помилка при завантаженні товарів:',
error));
    }, []);
    return (
        <div className="home-page">
```

```
            <section className="hero">
                <div className="container">
                    <div className="hero-content">
                        <h1>Ласкаво просимо до Квіткової Лавки</h1>
                        <p>Тут ви знайдете найкращі квіти для будь-якого випадку</p>
                        <Link to="/products" className="btn btn-primary">Переглянути
каталог</Link>
                    </div>
                </div>
            </section>
            <section className="categories section">
                <div className="container">
                    <h2 className="section-title">Категорії</h2>
                    <div className="categories-grid">
                        {categories.map(category => (
                            <div key={category.id} className="category-card">
                                <Link to={`/products?category=${category.id}`}>
                                    <div className="category-image">
                                        <img src={category.image ||
'https://via.placeholder.com/300'} alt={category.name} />
                                    </div>
                                    <h3 className="category-name">{category.name}</h3>
                                </Link>
                            </div>
                        ))}
                    </div>
                </div>
            </section>
            <section className="featured-products section">
                <div className="container">
                    <h2 className="section-title">Популярні товари</h2>
                    <div className="products-grid">
                        {featuredProducts.map(product => (
                            <div key={product.id} className="product-card">
                                <Link to={`/products/${product.id}`}>
                                    <div className="product-image">
                                        <img src={product.image ||
'https://via.placeholder.com/300'} alt={product.name} />
                                    </div>
                                    <div className="product-info">
                                        <h3 className="product-
name">{product.name}</h3>
                                        <p className="product-
price">{product.price.toFixed(2)} грн</p>
                                    </div>
                                </Link>
                            </div>
                        ))}
                    </div>
                    <div className="text-center mt-4">
                        <Link to="/products" className="btn btn-secondary">Переглянути
всі товари</Link>
                    </div>
                </div>
            </section>
        </div>
    );
}

// Компонент Login
function Login() {
    const { login } = useAuth();
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        email: '',
        password: ''
```

```
    });
    const [error, setError] = useState('');
    const [loading, setLoading] = useState(false);
    const handleChange = (e) => {
        setFormData({
            ...formData,
            [e.target.name]: e.target.value
        });
    };
    const handleSubmit = async (e) => {
        e.preventDefault();
        setError('');
        setLoading(true);
        const { email, password } = formData;
        if (!email || !password) {
            setError('Усі поля обов\'язкові');
            setLoading(false);
            return;
        }
        const result = await login(email, password);
        if (result.success) {
            navigate('/');
        } else {
            setError(result.message);
        }
        setLoading(false);
    };
    return (
        <div className="auth-page">
            <div className="container">
                <div className="auth-form-container">
                    <h2>Вхід в аккаунт</h2>
                    {error && <div className="alert alert-danger">{error}</div>}
                    <form className="auth-form" onSubmit={handleSubmit}>
                        <div className="form-group">
                            <label htmlFor="email">Email</label>
                            <input
                                type="email"
                                id="email"
                                name="email"
                                value={formData.email}
                                onChange={handleChange}
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="password">Пароль</label>
                            <input
                                type="password"
                                id="password"
                                name="password"
                                value={formData.password}
                                onChange={handleChange}
                                required
                            />
                        </div>
                        <button type="submit" className="btn btn-primary w-100"
disabled={loading}>
                            {loading ? 'Завантаження...' : 'Увійти'}
                        </button>
                    </form>
                    <div className="auth-links">
                        <p>Немає аккаунту? <Link
to="/register">Зареєструватися</Link></p>
                    </div>
```

```
                </div>
            </div>
        </div>
    );
}
// Компонент Register
function Register() {
    const { register } = useAuth();
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        name: '',
        email: '',
        password: '',
        confirmPassword: '',
        role: 'client'
    });
    const [error, setError] = useState('');
    const [loading, setLoading] = useState(false);
    const handleChange = (e) => {
        setFormData({
            ...formData,
            [e.target.name]: e.target.value
        });
    };
    const handleSubmit = async (e) => {
        e.preventDefault();
        setError('');
        setLoading(true);
        const { name, email, password, confirmPassword, role } = formData;
        if (!name || !email || !password || !confirmPassword) {
            setError('Усі поля обов\'язкові');
            setLoading(false);
            return;
        }
        if (password !== confirmPassword) {
            setError('Паролі не співпадають');
            setLoading(false);
            return;
        }
        const result = await register(name, email, password, role);
        if (result.success) {
            navigate('/');
        } else {
            setError(result.message);
        }
        setLoading(false);
    };
    return (
        <div className="auth-page">
            <div className="container">
                <div className="auth-form-container">
                    <h2>Реєстрація</h2>
                    {error && <div className="alert alert-danger">{error}</div>}
                    <form className="auth-form" onSubmit={handleSubmit}>
                        <div className="form-group">
                            <label htmlFor="name">Ім'я</label>
                            <input
                                type="text"
                                id="name"
                                name="name"
                                value={formData.name}
                                onChange={handleChange}
                                required
                            />
                        </div>
```

```
                        <div className="form-group">
                            <label htmlFor="email">Email</label>
                            <input
                                type="email"
                                id="email"
                                name="email"
                                value={formData.email}
                                onChange={handleChange}
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="password">Пароль</label>
                            <input
                                type="password"
                                id="password"
                                name="password"
                                value={formData.password}
                                onChange={handleChange}
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="confirmPassword">Підтвердіть
пароль</label>
                            <input
                                type="password"
                                id="confirmPassword"
                                name="confirmPassword"
                                value={formData.confirmPassword}
                                onChange={handleChange}
                                required
                            />
                        </div>
                        <div className="form-group">
                            <label htmlFor="role">Роль</label>
                            <select
                                id="role"
                                name="role"
                                value={formData.role}
                                onChange={handleChange}
                            >
                                <option value="client">Клієнт</option>
                                <option value="manager">Менеджер</option>
                            </select>
                        </div>
                        <button type="submit" className="btn btn-primary w-100"
disabled={loading}>
                            {loading ? 'Завантаження...' : 'Зареєструватися'}
                        </button>
                    </form>
                    <div className="auth-links">
                        <p>Вже є аккаунт? <Link to="/login">Увійти</Link></p>
                    </div>
                </div>
            </div>
        </div>
    );
}

// Компонент ProductList (Каталог)
function ProductList() {
    const [products, setProducts] = useState([]);
    const [categories, setCategories] = useState([]);
    const [loading, setLoading] = useState(false);
    const [filters, setFilters] = useState({
```

```
        category: '',
        search: '',
        sort: 'newest'
    });
    useEffect(() => {
        // Завантаження категорій
        fetch(`${API_URL}/categories`)
            .then(response => response.json())
            .then(data => setCategories(data))
            .catch(error => console.error('Помилка при завантаженні категорій:',
error));
        // Завантаження товарів з початковими фільтрами
        fetchProducts();
    }, []);
    useEffect(() => {
        fetchProducts();
    }, [filters]);
    const fetchProducts = () => {
        setLoading(true);
        let queryParams = new URLSearchParams();
        if (filters.category) {
            queryParams.append('category', filters.category);
        }
        if (filters.search) {
            queryParams.append('search', filters.search);
        }
        if (filters.sort) {
            queryParams.append('sort', filters.sort);
        }
        fetch(`${API_URL}/products?${queryParams.toString()}`)
            .then(response => response.json())
            .then(data => {
                setProducts(data);
                setLoading(false);
            })
            .catch(error => {
                console.error('Помилка при завантаженні товарів:', error);
                setLoading(false);
            });
    };
    const handleFilterChange = (e) => {
        setFilters({
            ...filters,
            [e.target.name]: e.target.value
        });
    };
    const handleSearchSubmit = (e) => {
        e.preventDefault();
        fetchProducts();
    };
    return (
        <div className="catalog-page">
            <div className="container">
                <h1 className="page-title">Каталог товарів</h1>
                <div className="filters-panel">
                    <form className="search-form" onSubmit={handleSearchSubmit}>
                        <input
                            type="text"
                            name="search"
                            placeholder="Пошук товарів..."
                            value={filters.search}
                            onChange={handleFilterChange}
                        />
                        <button type="submit" className="btn btn-
primary">Пошук</button>
```

```
                    </form>
                    <div className="filter-controls">
                        <div className="filter-item">
                            <label htmlFor="category">Категорія:</label>
                            <select
                                id="category"
                                name="category"
                                value={filters.category}
                                onChange={handleFilterChange}
                            >
                                <option value="">Усі категорії</option>
                                {categories.map(category => (
                                    <option key={category.id}
value={category.id}>{category.name}</option>
                                ))}
                            </select>
                        </div>
                        <div className="filter-item">
                            <label htmlFor="sort">Сортування:</label>
                            <select
                                id="sort"
                                name="sort"
                                value={filters.sort}
                                onChange={handleFilterChange}
                            >
                                <option value="newest">Найновіші</option>
                                <option value="price_asc">Ціна: від низької до
високої</option>
                                <option value="price_desc">Ціна: від високої до
низької</option>
                                <option value="name_asc">Назва: А-Я</option>
                                <option value="name_desc">Назва: Я-А</option>
                            </select>
                        </div>
                    </div>
                </div>
                {loading ? (
                    <div className="loading">Завантаження товарів...</div>
                ) : products.length === 0 ? (
                    <div className="no-results">Товари не знайдено</div>
                ) : (
                    <div className="products-grid">
                        {products.map(product => (
                            <div key={product.id} className="product-card">
                                <Link to={`/products/${product.id}`}>
                                    <div className="product-image">
                                        <img src={product.image ||
'https://via.placeholder.com/300'} alt={product.name} />
                                    </div>
                                    <div className="product-info">
                                        <h3 className="product-
name">{product.name}</h3>
                                        <p className="product-
price">{product.price.toFixed(2)} грн</p>
                                        <p className="product-
category">{product.category_name}</p>
                                    </div>
                                </Link>
                            </div>
                        ))}
                    </div>
                )}
            </div>
        </div>
    );
```

```
}
// Компонент ProductDetail
function ProductDetail() {
    const { id } = useParams();
    const { user, token } = useAuth();
    const navigate = useNavigate();
    const [product, setProduct] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState('');
    const [quantity, setQuantity] = useState(1);
    useEffect(() => {
        // Завантаження даних про товар
        const headers = token ? { 'Authorization': `Bearer ${token}` } : {};
        fetch(`${API_URL}/products/${id}`, { headers })
            .then(response => {
                if (!response.ok) {
                    throw new Error('Товар не знайдено');
                }
                return response.json();
            })
            .then(data => {
                setProduct(data);
                setLoading(false);
            })
            .catch(error => {
                console.error('Помилка при завантаженні товару:', error);
                setError(error.message || 'Помилка при завантаженні товару');
                setLoading(false);
            });
    }, [id, token]);
    const handleQuantityChange = (e) => {
        const value = parseInt(e.target.value);
        if (value > 0) {
            setQuantity(value);
        }
    };
    const addToCart = () => {
        if (!user) {
            navigate('/login');
            return;
        }
        if (user.role !== 'client') {
            alert('Тільки клієнти можуть додавати товари в кошик');
            return;
        }
        const cart = JSON.parse(localStorage.getItem('cart') || '[]');
        // Перевіряємо, чи товар вже є в кошику
        const existingItemIndex = cart.findIndex(item => item.product_id ===
product.id);
        if (existingItemIndex !== -1) {
            // Оновлюємо кількість, якщо товар вже є в кошику
            cart[existingItemIndex].quantity += quantity;
        } else {
            // Додаємо новий товар
            cart.push({
                product_id: product.id,
                name: product.name,
                price: product.price,
                image: product.image,
                quantity
            });
        }
        localStorage.setItem('cart', JSON.stringify(cart));
        alert(`Товар "${product.name}" додано до кошика`);
        // Тригер оновлення кількості товарів у кошику в Header
```

```
            window.dispatchEvent(new Event('cartUpdated'));
        };
        if (loading) {
            return <div className="loading">Завантаження...</div>;
        }
        if (error) {
            return <div className="error">{error}</div>;
        }
        if (!product) {
            return <div className="not-found">Товар не знайдено</div>;
        }
        return (
            <div className="product-detail-page">
                <div className="container">
                    <div className="product-detail">
                        <div className="product-image">
                            <img src={product.image || 'https://via.placeholder.com/500'}
alt={product.name} />
                        </div>
                        <div className="product-info">
                            <h1 className="product-name">{product.name}</h1>
                            <p className="product-price">{product.price.toFixed(2)}
грн</p>
                            <p className="product-category">Категорія:
{product.category_name}</p>
                            <div className="product-description">
                                <h3>Опис</h3>
                                <p>{product.description || 'Опис відсутній'}</p>
                            </div>
                            {product.in_stock ? (
                                <div className="product-actions">
                                    <div className="quantity-control">
                                        <label htmlFor="quantity">Кількість:</label>
                                        <input
                                            type="number"
                                            id="quantity"
                                            min="1"
                                            value={quantity}
                                            onChange={handleQuantityChange}
                                        />
                                    </div>
                                    <button
                                        className="btn btn-primary btn-add-to-cart"
                                        onClick={addToCart}
                                        disabled={user && user.role !== 'client'}
                                    >
                                        Додати до кошика
                                    </button>
                                </div>
                            ) : (
                                <div className="out-of-stock">
                                    <p>Товар тимчасово відсутній</p>
                                </div>
                            )}
                        </div>
                    </div>
                </div>
            </div>
        );
}
// Компонент Cart
function Cart() {
    const { user, token } = useAuth();
    const navigate = useNavigate();
    const [cart, setCart] = useState([]);
```

```
    const [totalPrice, setTotalPrice] = useState(0);
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState('');
    useEffect(() => {
        // Завантаження кошика з localStorage
        const storedCart = JSON.parse(localStorage.getItem('cart') || '[]');
        setCart(storedCart);
        // Розрахунок загальної вартості
        const total = storedCart.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
        setTotalPrice(total);
    }, []);
    const updateQuantity = (index, newQuantity) => {
        if (newQuantity < 1) return;
        const updatedCart = [...cart];
        updatedCart[index].quantity = newQuantity;
        setCart(updatedCart);
        localStorage.setItem('cart', JSON.stringify(updatedCart));
        // Оновлення загальної вартості
        const total = updatedCart.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
        setTotalPrice(total);
        // Тригер оновлення кількості товарів у кошику в Header
        window.dispatchEvent(new Event('cartUpdated'));
    };
    const removeItem = (index) => {
        const updatedCart = [...cart];
        updatedCart.splice(index, 1);
        setCart(updatedCart);
        localStorage.setItem('cart', JSON.stringify(updatedCart));
        // Оновлення загальної вартості
        const total = updatedCart.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
        setTotalPrice(total);
        // Тригер оновлення кількості товарів у кошику в Header
        window.dispatchEvent(new Event('cartUpdated'));
    };
    const clearCart = () => {
        setCart([]);
        localStorage.setItem('cart', JSON.stringify([]));
        setTotalPrice(0);
        // Тригер оновлення кількості товарів у кошику в Header
        window.dispatchEvent(new Event('cartUpdated'));
    };
    const handleCheckout = async () => {
        if (cart.length === 0) {
            setError('Кошик порожній');
            return;
        }
        setLoading(true);
        setError('');
        try {
            // Форматуємо дані для створення замовлення
            const orderData = {
                products: cart.map(item => ({
                    product_id: item.product_id,
                    quantity: item.quantity,
                    price: item.price
                })),
                total_price: totalPrice
            };
            const response = await fetch(`${API_URL}/orders`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
```

```
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify(orderData)
            });
            const data = await response.json();
            if (response.ok) {
                // Замовлення успішно створено, виконуємо симуляцію оплати
                const paymentResponse = await fetch(`${API_URL}/payments/simulate`, {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                        'Authorization': `Bearer ${token}`
                    },
                    body: JSON.stringify({
                        payment_id: data.payment_id,
                        order_id: data.order.id
                    })
                });
                const paymentData = await paymentResponse.json();
                if (paymentResponse.ok) {
                    // Оплата пройшла успішно
                    clearCart();
                    alert('Замовлення успішно оформлено та оплачено!');
                    navigate('/orders');
                } else {
                    setError(paymentData.message || 'Помилка при оплаті замовлення');
                }
            } else {
                setError(data.message || 'Помилка при створенні замовлення');
            }
        } catch (error) {
            setError('Помилка при оформленні замовлення');
            console.error('Помилка при оформленні замовлення:', error);
        } finally {
            setLoading(false);
        }
    };
    return (
        <div className="cart-page">
            <div className="container">
                <h1 className="page-title">Кошик</h1>
                {error && <div className="alert alert-danger">{error}</div>}
                {cart.length === 0 ? (
                    <div className="empty-cart">
                        <p>Ваш кошик порожній</p>
                        <Link to="/products" className="btn btn-primary">Перейти до
каталогу</Link>
                    </div>
                ) : (
                    <>
                        <div className="cart-items">
                            {cart.map((item, index) => (
                                <div key={index} className="cart-item">
                                    <div className="item-image">
                                        <img src={item.image ||
'https://via.placeholder.com/100'} alt={item.name} />
                                    </div>
                                    <div className="item-details">
                                        <h3 className="item-name">{item.name}</h3>
                                        <p className="item-
price">{item.price.toFixed(2)} грн</p>
                                    </div>
                                    <div className="item-quantity">
                                        <button
                                            className="btn btn-sm btn-outline"
```

```
                                        onClick={() => updateQuantity(index,
item.quantity - 1)}
                                    >
                                        -
                                    </button>
                                    <span>{item.quantity}</span>
                                    <button
                                        className="btn btn-sm btn-outline"
                                        onClick={() => updateQuantity(index,
item.quantity + 1)}
                                    >
                                        +
                                    </button>
                                </div>
                                <div className="item-total">
                                    <p>{(item.price * item.quantity).toFixed(2)}
грн</p>
                                </div>
                                <button
                                    className="btn btn-sm btn-danger"
                                    onClick={() => removeItem(index)}
                                >
                                    Видалити
                                </button>
                            </div>
                        ))}
                    </div>
                    <div className="cart-summary">
                        <div className="cart-total">
                            <h3>Загальна сума:</h3>
                            <p>{totalPrice.toFixed(2)} грн</p>
                        </div>
                        <div className="cart-actions">
                            <button
                                className="btn btn-outline"
                                onClick={clearCart}
                            >
                                Очистити кошик
                            </button>
                            <button
                                className="btn btn-primary"
                                onClick={handleCheckout}
                                disabled={loading}
                            >
                                {loading ? 'Обробка...' : 'Оформити замовлення'}
                            </button>
                        </div>
                    </div>
                </>
            )}
        </div>
    </div>
  );
}
// Компонент Profile
function Profile() {
    const { user, updateProfile } = useAuth();
    const [formData, setFormData] = useState({
        name: '',
        phone: '',
        address: '',
        password: '',
        confirmPassword: ''
    });
    const [viewHistory, setViewHistory] = useState([]);
```

```
const [loading, setLoading] = useState(false);
const [message, setMessage] = useState('');
const [error, setError] = useState('');
const [activeTab, setActiveTab] = useState('profile');
useEffect(() => {
    if (user) {
        setFormData({
            name: user.name || '',
            phone: user.phone || '',
            address: user.address || '',
            password: '',
            confirmPassword: ''
        });
    }
    // Завантаження історії переглядів для клієнтів
    if (user && user.role === 'client') {
        fetchViewHistory();
    }
}, [user]);
const fetchViewHistory = async () => {
    try {
        const response = await fetch(`${API_URL}/view-history`, {
            headers: {
                'Authorization': `Bearer ${localStorage.getItem('token')}`
            }
        });
        if (response.ok) {
            const data = await response.json();
            setViewHistory(data);
        }
    } catch (error) {
        console.error('Помилка при завантаженні історії переглядів:', error);
    }
};
const handleChange = (e) => {
    setFormData({
        ...formData,
        [e.target.name]: e.target.value
    });
};
const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage('');
    setError('');
    // Перевірка паролів, якщо вони вказані
    if (formData.password && formData.password !== formData.confirmPassword) {
        setError('Паролі не співпадають');
        return;
    }
    setLoading(true);
    // Підготовка даних для оновлення
    const updateData = {
        name: formData.name,
        phone: formData.phone,
        address: formData.address
    };
    // Додаємо пароль, якщо він був введений
    if (formData.password) {
        updateData.password = formData.password;
    }
    const result = await updateProfile(updateData);
    if (result.success) {
        setMessage('Профіль успішно оновлено');
        // Очищаємо поля пароля
        setFormData({
```

```
                ...formData,
                password: '',
                confirmPassword: ''
            });
        } else {
            setError(result.message || 'Помилка при оновленні профілю');
        }
        setLoading(false);
    };
    return (
        <div className="profile-page">
            <div className="container">
                <h1 className="page-title">Мій профіль</h1>
                <div className="profile-tabs">
                    <div
                        className={`profile-tab ${activeTab === 'profile' ? 'active' :
''}`}
                        onClick={() => setActiveTab('profile')}
                    >
                        Особисті дані
                    </div>
                    {user && user.role === 'client' && (
                        <div
                            className={`profile-tab ${activeTab === 'history' ?
'active' : ''}`}
                            onClick={() => setActiveTab('history')}
                        >
                            Історія переглядів
                        </div>
                    )}
                </div>
                <div className="profile-content">
                    {activeTab === 'profile' && (
                        <div className="profile-form-container">
                            {message && <div className="alert alert-
success">{message}</div>}
                            {error && <div className="alert alert-
danger">{error}</div>}
                            <form className="profile-form" onSubmit={handleSubmit}>
                                <div className="form-group">
                                    <label htmlFor="name">Ім'я</label>
                                    <input
                                        type="text"
                                        id="name"
                                        name="name"
                                        value={formData.name}
                                        onChange={handleChange}
                                    />
                                </div>
                                <div className="form-group">
                                    <label htmlFor="email">Email</label>
                                    <input
                                        type="email"
                                        id="email"
                                        value={user?.email || ''}
                                        disabled
                                    />
                                    <small className="form-text text-muted">Email не
можна змінити</small>
                                </div>
                                <div className="form-group">
                                    <label htmlFor="phone">Телефон</label>
                                    <input
                                        type="tel"
                                        id="phone"
```

```
                                        name="phone"
                                        value={formData.phone}
                                        onChange={handleChange}
                                    />
                                </div>
                                <div className="form-group">
                                    <label htmlFor="address">Адреса</label>
                                    <textarea
                                        id="address"
                                        name="address"
                                        value={formData.address}
                                        onChange={handleChange}
                                        rows="3"
                                    ></textarea>
                                </div>
                                <div className="form-group">
                                    <label htmlFor="password">Новий пароль</label>
                                    <input
                                        type="password"
                                        id="password"
                                        name="password"
                                        value={formData.password}
                                        onChange={handleChange}
                                    />
                                    <small className="form-text text-muted">Залиште
пустим, якщо не хочете змінювати пароль</small>
                                </div>
                                <div className="form-group">
                                    <label htmlFor="confirmPassword">Підтвердження
нового пароля</label>
                                    <input
                                        type="password"
                                        id="confirmPassword"
                                        name="confirmPassword"
                                        value={formData.confirmPassword}
                                        onChange={handleChange}
                                    />
                                </div>
                                <button type="submit" className="btn btn-primary"
disabled={loading}>
                                    {loading ? 'Збереження...' : 'Зберегти зміни'}
                                </button>
                            </form>
                        </div>
                    )}

                    {activeTab === 'history' && (
                        <div className="view-history">
                            <h2>Історія переглядів</h2>
                            {viewHistory.length === 0 ? (
                                <p className="no-history">Ви ще не переглядали
товари</p>
                            ) : (
                                <div className="products-grid">
                                    {viewHistory.map((item, index) => (
                                        <div key={index} className="product-card">
                                            <Link to={`/products/${item.product_id}`}>
                                                <div className="product-image">
                                                    <img src={item.image ||
'https://via.placeholder.com/300'} alt={item.name} />
                                                </div>
                                                <div className="product-info">
                                                    <h3 className="product-
name">{item.name}</h3>
                                                    <p className="product-
price">{item.price.toFixed(2)} грн</p>
```

```
                                                    <p className="product-
category">{item.category_name}</p>
                                                    <p className="view-date">
                                                        Переглянуто: {new
Date(item.viewed_at).toLocaleString('uk-UA')}
                                                    </p>
                                                </div>
                                            </Link>
                                        </div>
                                    ))}
                                </div>
                            )}
                        </div>
                    )}
                </div>
            </div>
        );
}
// Компонент Orders
function Orders() {
    const { user, token } = useAuth();
    const [orders, setOrders] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState('');
    useEffect(() => {
        fetchOrders();
    }, [token]);
    const fetchOrders = async () => {
        try {
            const response = await fetch(`${API_URL}/orders`, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            if (response.ok) {
                const data = await response.json();
                setOrders(data);
            } else {
                throw new Error('Помилка при завантаженні замовлень');
            }
        } catch (error) {
            setError(error.message || 'Помилка при завантаженні замовлень');
        } finally {
            setLoading(false);
        }
    };
    const updateOrderStatus = async (orderId, status) => {
        try {
            const response = await fetch(`${API_URL}/orders/${orderId}/status`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify({ status })
            });
            if (response.ok) {
                // Оновлюємо список замовлень
                fetchOrders();
            } else {
                const data = await response.json();
                throw new Error(data.message || 'Помилка при оновленні статусу
замовлення');
            }
```

```
        } catch (error) {
            alert(error.message);
        }
    };
    // Функція для відображення статусу замовлення
    const getStatusText = (status) => {
        switch (status) {
            case 'pending': return 'В обробці';
            case 'paid': return 'Оплачено';
            case 'shipped': return 'Відправлено';
            case 'delivered': return 'Доставлено';
            case 'cancelled': return 'Скасовано';
            default: return 'Невідомий статус';
        }
    };
    // Функція для визначення класу статусу
    const getStatusClass = (status) => {
        switch (status) {
            case 'pending': return 'status-pending';
            case 'paid': return 'status-paid';
            case 'shipped': return 'status-shipped';
            case 'delivered': return 'status-delivered';
            case 'cancelled': return 'status-cancelled';
            default: return '';
        }
    };
    if (loading) {
        return <div className="loading">Завантаження замовлень...</div>;
    }
    if (error) {
        return <div className="error">{error}</div>;
    }
    return (
        <div className="orders-page">
            <div className="container">
                <h1 className="page-title">
                    {user.role === 'manager' ? 'Всі замовлення' : 'Мої замовлення'}
                </h1>
                {orders.length === 0 ? (
                    <div className="no-orders">
                        <p>Замовлень не знайдено</p>
                    </div>
                ) : (
                    <div className="orders-list">
                        {orders.map(order => (
                            <div key={order.id} className="order-card">
                                <div className="order-header">
                                    <h3 className="order-id">
                                        Замовлення #{order.id}
                                    </h3>
                                    <span className={`order-status
${getStatusClass(order.status)}`}>
                                        {getStatusText(order.status)}
                                    </span>
                                </div>
                                <div className="order-info">
                                    <p className="order-date">
                                        Дата: {new
Date(order.created_at).toLocaleString('uk-UA')}
                                    </p>
                                    <p className="order-total">
                                        Сума: {order.total_price.toFixed(2)} грн
                                    </p>
                                    {user.role === 'manager' && (
                                        <p className="order-client">
```

```
                                        Клієнт: {order.user_name}
({order.user_email})
                                    </p>
                                )}
                                {user.role === 'manager' && (
                                    <div className="order-status-control">
                                        <label>Змінити статус:</label>
                                        <select
                                            value={order.status}
                                            onChange={(e) =>
updateOrderStatus(order.id, e.target.value)}
                                        >
                                            <option value="pending">В
обробці</option>
                                            <option value="paid">Оплачено</option>
                                            <option
value="shipped">Відправлено</option>
                                            <option
value="delivered">Доставлено</option>
                                            <option
value="cancelled">Скасовано</option>
                                        </select>
                                    </div>
                                )}
                            </div>

                            <div className="order-items">
                                <h4>Товари в замовленні:</h4>
                                {order.items && order.items.map(item => (
                                    <div key={item.id} className="order-item">
                                        <div className="item-image">
                                            <img src={item.image ||
'https://via.placeholder.com/50'} alt={item.name} />
                                        </div>
                                        <div className="item-info">
                                            <h5 className="item-
name">{item.name}</h5>
                                            <p className="item-
price">{item.price.toFixed(2)} грн x {item.quantity}</p>
                                        </div>
                                        <div className="item-total">
                                            {(item.price *
item.quantity).toFixed(2)} грн
                                        </div>
                                    </div>
                                ))}
                            </div>
                        </div>
                    ))}
                </div>
            )}
        </div>
    </div>
    );
}
// Компонент Chat
function Chat() {
    const { user, socket } = useAuth();
    const [contacts, setContacts] = useState([]);
    const [selectedContact, setSelectedContact] = useState(null);
    const [messages, setMessages] = useState([]);
    const [newMessage, setNewMessage] = useState('');
    const [loading, setLoading] = useState(true);
    const messagesEndRef = React.useRef(null);
    useEffect(() => {
        if (socket) {
```

```
            // Завантаження контактів в залежності від ролі
            if (user.role === 'manager') {
                // Менеджер отримує список всіх клієнтів
                fetch(`${API_URL}/users`, {
                    headers: {
                        'Authorization': `Bearer ${localStorage.getItem('token')}`
                    }
                })
                    .then(response => response.json())
                    .then(data => {
                        setContacts(data);
                        setLoading(false);
                    })
                    .catch(error => {
                        console.error('Помилка при завантаженні контактів:', error);
                        setLoading(false);
                    });
            } else {
                // Клієнт отримує список всіх менеджерів
                fetch(`${API_URL}/users?role=manager`, {
                    headers: {
                        'Authorization': `Bearer ${localStorage.getItem('token')}`
                    }
                })
                    .then(response => response.json())
                    .then(data => {
                        setContacts(data);
                        setLoading(false);
                        // Якщо є менеджери, вибираємо першого за замовчуванням
                        if (data.length > 0) {
                            setSelectedContact(data[0]);
                        }
                    })
                    .catch(error => {
                        console.error('Помилка при завантаженні контактів:', error);
                        setLoading(false);
                    });
            }
            // Прослуховування нових повідомлень
            socket.on('message', (message) => {
                setMessages(prevMessages => [...prevMessages, message]);
                // Відзначаємо повідомлення як прочитане, якщо воно для нас
                if (message.receiver_id === user.id) {
                    socket.emit('mark_read', { message_ids: [message.id] });
                }
            });
            // Отримання історії чату при виборі контакту
            socket.on('chat_history', (history) => {
                setMessages(history);
            });
            // Отримання непрочитаних повідомлень
            socket.on('unread_messages', (messages) => {
                console.log('Отримано непрочитані повідомлення:', messages);
            });
            // Очищення при розмонтуванні
            return () => {
                socket.off('message');
                socket.off('chat_history');
                socket.off('unread_messages');
            };
        }
    }, [socket, user]);
    useEffect(() => {
        // При виборі контакту завантажуємо історію чату
        if (socket && selectedContact) {
```

```
            socket.emit('get_chat_history', { receiver_id: selectedContact.id });
        }
    }, [socket, selectedContact]);
    useEffect(() => {
        // Прокручування до останнього повідомлення
        if (messagesEndRef.current) {
            messagesEndRef.current.scrollIntoView({ behavior: 'smooth' });
        }
    }, [messages]);
    const handleContactSelect = (contact) => {
        setSelectedContact(contact);
    };
    const handleSendMessage = (e) => {
        e.preventDefault();
        if (!newMessage.trim() || !selectedContact) return;
        socket.emit('send_message', {
            receiver_id: selectedContact.id,
            content: newMessage
        });
        setNewMessage('');
    };
    if (loading) {
        return <div className="loading">Завантаження чату...</div>;
    }

    return (
        <div className="chat-page">
            <div className="container">
                <h1 className="page-title">Чат</h1>
                <div className="chat-container">
                    <div className="chat-sidebar">
                        <h3>Контакти</h3>
                        {contacts.length === 0 ? (
                            <p className="no-contacts">Контактів не знайдено</p>
                        ) : (
                            <ul className="contacts-list">
                                {contacts.map(contact => (
                                    <li
                                        key={contact.id}
                                        className={`contact-item ${selectedContact &&
selectedContact.id === contact.id ? 'active' : ''}`}
                                        onClick={() => handleContactSelect(contact)}
                                    >
                                        <div className="contact-info">
                                            <h4 className="contact-
name">{contact.name}</h4>
                                            <p className="contact-
email">{contact.email}</p>
                                        </div>
                                    </li>
                                ))}
                            </ul>
                        )}
                    </div>
                    <div className="chat-main">
                        {!selectedContact ? (
                            <div className="no-contact-selected">
                                <p>Виберіть контакт для початку чату</p>
                            </div>
                        ) : (
                            <>
                                <div className="chat-header">
                                    <h3>{selectedContact.name}</h3>
                                </div>
                                <div className="chat-messages">
                                    {messages.length === 0 ? (
```

```
                                        <div className="no-messages">
                                            <p>Немає повідомлень. Почніть розмову!</p>
                                        </div>
                                    ) : (
                                        messages.map(message => (
                                            <div
                                                key={message.id}
                                                className={`message
${message.sender_id === user.id ? 'own-message' : 'other-message'}`}
                                            >
                                                <div className="message-content">
                                                    {message.content}
                                                </div>
                                                <div className="message-info">
                                                    <span className="message-time">
                                                        {new
Date(message.created_at).toLocaleTimeString('uk-UA')}
                                                    </span>
                                                    {message.sender_id === user.id &&
(
                                                        <span className="message-
status">
                                                            {message.read ? '✓✓' :
'✓'}
                                                        </span>
                                                    )}
                                                </div>
                                            </div>
                                        ))
                                    )}
                                    <div ref={messagesEndRef} />
                                </div>
                                <form className="chat-form"
onSubmit={handleSendMessage}>
                                    <input
                                        type="text"
                                        placeholder="Введіть повідомлення..."
                                        value={newMessage}
                                        onChange={(e) =>
setNewMessage(e.target.value)}
                                    />
                                    <button type="submit" className="btn btn-
primary">Надіслати</button>
                                </form>
                            </>
                        )}
                    </div>
                </div>
            </div>
        </div>
    );
}
// Компонент AdminProducts
function AdminProducts() {
    const { token } = useAuth();
    const [products, setProducts] = useState([]);
    const [categories, setCategories] = useState([]);
    const [loading, setLoading] = useState(true);
    const [currentProduct, setCurrentProduct] = useState(null);
    const [isModalOpen, setIsModalOpen] = useState(false);
    const [formData, setFormData] = useState({
        name: '',
        description: '',
        price: '',
        image: '',
```

```
            category_id: '',
            in_stock: true
        });
        useEffect(() => {
            // Завантаження категорій
            fetch(`${API_URL}/categories`)
                .then(response => response.json())
                .then(data => setCategories(data))
                .catch(error => console.error('Помилка при завантаженні категорій:',
error));
            // Завантаження товарів
            fetchProducts();
        }, [token]);
        const fetchProducts = () => {
            setLoading(true);
            fetch(`${API_URL}/products`, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            })
                .then(response => response.json())
                .then(data => {
                    setProducts(data);
                    setLoading(false);
                })
                .catch(error => {
                    console.error('Помилка при завантаженні товарів:', error);
                    setLoading(false);
                });
        };
        const openModal = (product = null) => {
            if (product) {
                // Редагування існуючого товару
                setCurrentProduct(product);
                setFormData({
                    name: product.name,
                    description: product.description || '',
                    price: product.price.toString(),
                    image: product.image || '',
                    category_id: product.category_id || '',
                    in_stock: product.in_stock
                });
            } else {
                // Створення нового товару
                setCurrentProduct(null);
                setFormData({
                    name: '',
                    description: '',
                    price: '',
                    image: '',
                    category_id: '',
                    in_stock: true
                });
            }
            setIsModalOpen(true);
        };
        const closeModal = () => {
            setIsModalOpen(false);
        };
        const handleChange = (e) => {
            const { name, value, type, checked } = e.target;
            setFormData({
                ...formData,
                [name]: type === 'checkbox' ? checked : value
            });
```

```
    };
    const handleSubmit = async (e) => {
        e.preventDefault();
        // Валідація форми
        if (!formData.name || !formData.price) {
            alert('Назва та ціна є обов\'язковими полями');
            return;
        }
        try {
            const method = currentProduct ? 'PUT' : 'POST';
            const url = currentProduct
                ? `${API_URL}/products/${currentProduct.id}`
                : `${API_URL}/products`;
            const response = await fetch(url, {
                method,
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify({
                    ...formData,
                    price: parseFloat(formData.price)
                })
            });
            if (response.ok) {
                fetchProducts();
                closeModal();
            } else {
                const data = await response.json();
                throw new Error(data.message || 'Помилка при збереженні товару');
            }
        } catch (error) {
            alert(error.message);
        }
    };
const handleDelete = async (productId) => {
    if (!window.confirm('Ви впевнені, що хочете видалити цей товар?')) {
        return;
    }
    try {
        const response = await fetch(`${API_URL}/products/${productId}`, {
            method: 'DELETE',
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });
        if (response.ok) {
            fetchProducts();
        } else {
            const data = await response.json();
            throw new Error(data.message || 'Помилка при видаленні товару');
        }
    } catch (error) {
        alert(error.message);
    }
};
return (
    <div className="admin-products-page">
        <div className="container">
            <div className="admin-header">
                <h1 className="page-title">Управління товарами</h1>
                <button className="btn btn-primary" onClick={() => openModal()}>
                    Додати товар
                </button>
            </div>
```

```
            {loading ? (
                <div className="loading">Завантаження товарів...</div>
            ) : products.length === 0 ? (
                <div className="no-items">Товари не знайдено</div>
            ) : (
                <div className="admin-table-container">
                    <table className="admin-table">
                        <thead>
                            <tr>
                                <th>ID</th>
                                <th>Зображення</th>
                                <th>Назва</th>
                                <th>Категорія</th>
                                <th>Ціна</th>
                                <th>Наявність</th>
                                <th>Дії</th>
                            </tr>
                        </thead>
                        <tbody>
                            {products.map(product => (
                                <tr key={product.id}>
                                    <td>{product.id}</td>
                                    <td>
                                        <img
                                            src={product.image ||
'https://via.placeholder.com/50'}
                                            alt={product.name}
                                            className="table-image"
                                        />
                                    </td>
                                    <td>{product.name}</td>
                                    <td>{product.category_name}</td>
                                    <td>{product.price.toFixed(2)} грн</td>
                                    <td>
                                        <span className={`status ${product.in_stock ?
'status-active' : 'status-inactive'}`}>
                                            {product.in_stock ? 'В наявності' : 'Немає
в наявності'}
                                        </span>
                                    </td>
                                    <td>
                                        <div className="table-actions">
                                            <button
                                                className="btn btn-sm btn-edit"
                                                onClick={() => openModal(product)}
                                            >
                                                Редагувати
                                            </button>
                                            <button
                                                className="btn btn-sm btn-delete"
                                                onClick={() =>
handleDelete(product.id)}
                                            >
                                                Видалити
                                            </button>
                                        </div>
                                    </td>
                                </tr>
                            ))}
                        </tbody>
                    </table>
                </div>
            )}
            {isModalOpen && (
                <div className="modal">
```

```
                    <div className="modal-content">
                        <div className="modal-header">
                            <h2>{currentProduct ? 'Редагувати товар' : 'Додати
товар'}</h2>
                            <button className="close-btn"
onClick={closeModal}>&times;</button>
                        </div>
                        <form className="product-form" onSubmit={handleSubmit}>
                            <div className="form-group">
                                <label htmlFor="name">Назва</label>
                                <input
                                    type="text"
                                    id="name"
                                    name="name"
                                    value={formData.name}
                                    onChange={handleChange}
                                    required
                                />
                            </div>

                                <div className="form-group">
                                    <label htmlFor="description">Опис</label>
                                    <textarea
                                        id="description"
                                        name="description"
                                        value={formData.description}
                                        onChange={handleChange}
                                        rows="4"
                                    ></textarea>
                                </div>
                                <div className="form-group">
                                    <label htmlFor="price">Ціна</label>
                                    <input
                                        type="number"
                                        id="price"
                                        name="price"
                                        value={formData.price}
                                        onChange={handleChange}
                                        step="0.01"
                                        min="0"
                                        required
                                    />
                                </div>
                                <div className="form-group">
                                    <label htmlFor="image">URL зображення</label>
                                    <input
                                        type="text"
                                        id="image"
                                        name="image"
                                        value={formData.image}
                                        onChange={handleChange}
                                    />
                                </div>
                                <div className="form-group">
                                    <label htmlFor="category_id">Категорія</label>
                                    <select
                                        id="category_id"
                                        name="category_id"
                                        value={formData.category_id}
                                        onChange={handleChange}
                                    >
                                        <option value="">Виберіть категорію</option>
                                        {categories.map(category => (
                                            <option key={category.id}
value={category.id}>
                                                {category.name}
```

```
                                        </option>
                                    ))}
                                </select>
                            </div>
                            <div className="form-check">
                                <input
                                    type="checkbox"
                                    id="in_stock"
                                    name="in_stock"
                                    checked={formData.in_stock}
                                    onChange={handleChange}
                                />
                                <label htmlFor="in_stock">В наявності</label>
                            </div>
                            <div className="form-actions">
                                <button type="button" className="btn btn-
secondary" onClick={closeModal}>
                                    Скасувати
                                </button>
                                <button type="submit" className="btn btn-primary">
                                    {currentProduct ? 'Оновити' : 'Створити'}
                                </button>
                            </div>
                        </form>
                    </div>
                </div>
            )}
        </div>
    </div>
    );
}
// Компонент AdminCategories
function AdminCategories() {
    const { token } = useAuth();
    const [categories, setCategories] = useState([]);
    const [loading, setLoading] = useState(true);
    const [currentCategory, setCurrentCategory] = useState(null);
    const [isModalOpen, setIsModalOpen] = useState(false);
    const [formData, setFormData] = useState({
        name: '',
        description: '',
        image: ''
    });
    useEffect(() => {
        fetchCategories();
    }, [token]);
    const fetchCategories = () => {
        setLoading(true);
        fetch(`${API_URL}/categories`)
            .then(response => response.json())
            .then(data => {
                setCategories(data);
                setLoading(false);
            })
            .catch(error => {
                console.error('Помилка при завантаженні категорій:', error);
                setLoading(false);
            });
    };
    const openModal = (category = null) => {
        if (category) {
            // Редагування існуючої категорії
            setCurrentCategory(category);
            setFormData({
                name: category.name,
```

```
                description: category.description || '',
                image: category.image || ''
            });
        } else {
            // Створення нової категорії
            setCurrentCategory(null);
            setFormData({
                name: '',
                description: '',
                image: ''
            });
        }
        setIsModalOpen(true);
    };
    const closeModal = () => {
        setIsModalOpen(false);
    };
    const handleChange = (e) => {
        setFormData({
            ...formData,
            [e.target.name]: e.target.value
        });
    };
    const handleSubmit = async (e) => {
        e.preventDefault();
        // Валідація форми
        if (!formData.name) {
            alert('Назва є обов\'язковим полем');
            return;
        }
        try {
            const method = currentCategory ? 'PUT' : 'POST';
            const url = currentCategory
                ? `${API_URL}/categories/${currentCategory.id}`
                : `${API_URL}/categories`;
            const response = await fetch(url, {
                method,
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${token}`
                },
                body: JSON.stringify(formData)
            });
            if (response.ok) {
                fetchCategories();
                closeModal();
            } else {
                const data = await response.json();
                throw new Error(data.message || 'Помилка при збереженні категорії');
            }
        } catch (error) {
            alert(error.message);
        }
    };
    const handleDelete = async (categoryId) => {
        if (!window.confirm('Ви впевнені, що хочете видалити цю категорію? Усі
пов\'язані товари втратять зв\'язок з категорією.')) {
            return;
        }
        try {
            const response = await fetch(`${API_URL}/categories/${categoryId}`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
```

```
            });
            if (response.ok) {
                fetchCategories();
            } else {
                const data = await response.json();
                throw new Error(data.message || 'Помилка при видаленні категорії');
            }
        } catch (error) {
            alert(error.message);
        }
    };
    return (
        <div className="admin-categories-page">
            <div className="container">
                <div className="admin-header">
                    <h1 className="page-title">Управління категоріями</h1>
                    <button className="btn btn-primary" onClick={() => openModal()}>
                        Додати категорію
                    </button>
                </div>
                {loading ? (
                    <div className="loading">Завантаження категорій...</div>
                ) : categories.length === 0 ? (
                    <div className="no-items">Категорії не знайдено</div>
                ) : (
                    <div className="admin-table-container">
                        <table className="admin-table">
                            <thead>
                                <tr>
                                    <th>ID</th>
                                    <th>Зображення</th>
                                    <th>Назва</th>
                                    <th>Опис</th>
                                    <th>Дії</th>
                                </tr>
                            </thead>
                            <tbody>
                                {categories.map(category => (
                                    <tr key={category.id}>
                                        <td>{category.id}</td>
                                        <td>
                                            <img
                                                src={category.image ||
'https://via.placeholder.com/50'}
                                                alt={category.name}
                                                className="table-image"
                                            />
                                        </td>
                                        <td>{category.name}</td>
                                        <td>{category.description || 'Без опису'}</td>
                                        <td>
                                            <div className="table-actions">
                                                <button
                                                    className="btn btn-sm btn-edit"
                                                    onClick={() =>
openModal(category)}
                                                >
                                                    Редагувати
                                                </button>
                                                <button
                                                    className="btn btn-sm btn-delete"
                                                    onClick={() =>
handleDelete(category.id)}
                                                >
                                                    Видалити
```

```
                                                </button>
                                            </div>
                                        </td>
                                    </tr>
                                ))}
                            </tbody>
                        </table>
                    </div>
                )}

                {isModalOpen && (
                    <div className="modal">
                        <div className="modal-content">
                            <div className="modal-header">
                                <h2>{currentCategory ? 'Редагувати категорію' :
'Додати категорію'}</h2>
                                <button className="close-btn"
onClick={closeModal}>&times;</button>
                            </div>
                            <form className="category-form" onSubmit={handleSubmit}>
                                <div className="form-group">
                                    <label htmlFor="name">Назва</label>
                                    <input
                                        type="text"
                                        id="name"
                                        name="name"
                                        value={formData.name}
                                        onChange={handleChange}
                                        required
                                    />
                                </div>
                                <div className="form-group">
                                    <label htmlFor="description">Опис</label>
                                    <textarea
                                        id="description"
                                        name="description"
                                        value={formData.description}
                                        onChange={handleChange}
                                        rows="4"
                                    ></textarea>
                                </div>
                                <div className="form-group">
                                    <label htmlFor="image">URL зображення</label>
                                    <input
                                        type="text"
                                        id="image"
                                        name="image"
                                        value={formData.image}
                                        onChange={handleChange}
                                    />
                                </div>
                                <div className="form-actions">
                                    <button type="button" className="btn btn-
secondary" onClick={closeModal}>
                                        Скасувати
                                    </button>
                                    <button type="submit" className="btn btn-primary">
                                        {currentCategory ? 'Оновити' : 'Створити'}
                                    </button>
                                </div>
                            </form>
                        </div>
                    </div>
                )}
            </div>
        </div>
```

```
    );
}
// Компонент AdminUsers
function AdminUsers() {
    const { token } = useAuth();
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = useState(true);
    useEffect(() => {
        fetchUsers();
    }, [token]);
    const fetchUsers = () => {
        setLoading(true);
        fetch(`${API_URL}/users`, {
            headers: {
                'Authorization': `Bearer ${token}`
            }
        })
            .then(response => response.json())
            .then(data => {
                setUsers(data);
                setLoading(false);
            })
            .catch(error => {
                console.error('Помилка при завантаженні користувачів:', error);
                setLoading(false);
            });
    };
    const handleDelete = async (userId) => {
        if (!window.confirm('Ви впевнені, що хочете видалити цього користувача? Усі
пов\'язані замовлення будуть збережені, але стануть анонімними.')) {
            return;
        }
        try {
            const response = await fetch(`${API_URL}/users/${userId}`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            if (response.ok) {
                fetchUsers();
            } else {
                const data = await response.json();
                throw new Error(data.message || 'Помилка при видаленні користувача');
            }
        } catch (error) {
            alert(error.message);
        }
    };
    return (
        <div className="admin-users-page">
            <div className="container">
                <h1 className="page-title">Управління користувачами</h1>
                {loading ? (
                    <div className="loading">Завантаження користувачів...</div>
                ) : users.length === 0 ? (
                    <div className="no-items">Користувачів не знайдено</div>
                ) : (
                    <div className="admin-table-container">
                        <table className="admin-table">
                            <thead>
                                <tr>
                                    <th>ID</th>
                                    <th>Ім'я</th>
                                    <th>Email</th>
```

```
                                    <th>Телефон</th>
                                    <th>Адреса</th>
                                    <th>Дата реєстрації</th>
                                    <th>Дії</th>
                                </tr>
                            </thead>
                            <tbody>
                                {users.map(user => (
                                    <tr key={user.id}>
                                        <td>{user.id}</td>
                                        <td>{user.name}</td>
                                        <td>{user.email}</td>
                                        <td>{user.phone || 'Не вказано'}</td>
                                        <td>{user.address || 'Не вказано'}</td>
                                        <td>{new
Date(user.created_at).toLocaleString('uk-UA')}</td>
                                        <td>
                                            <div className="table-actions">
                                                <button
                                                    className="btn btn-sm btn-delete"
                                                    onClick={() =>
handleDelete(user.id)}
                                                >
                                                    Видалити
                                                </button>
                                            </div>
                                        </td>
                                    </tr>
                                ))}
                            </tbody>
                        </table>
                    </div>
                )}
            </div>
        </div>
    );
}
// Компонент NotFound
function NotFound() {
    return (
        <div className="not-found-page">
            <div className="container">
                <h1>404</h1>
                <h2>Сторінку не знайдено</h2>
                <p>Сторінка, яку ви шукаєте, не існує або була переміщена</p>
                <Link to="/" className="btn btn-primary">На головну</Link>
            </div>
        </div>
    );
}
// Рендеринг додатку в DOM
ReactDOM.createRoot(document.getElementById('root')).render(<BrowserRouter><App
/></BrowserRouter>);
```

server.js

```
const express = require('express');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const sqlite3 = require('sqlite3').verbose();
const http = require('http');
const { Server } = require('socket.io');
```

```
const path = require('path');
const app = express();
const server = http.createServer(app);
const io = new Server(server, {
    cors: {
        origin: "http://localhost:3000",
        methods: ["GET", "POST"]
    }
});
// Налаштування middleware
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname)));
app.use(express.static(path.join(__dirname, 'public')));
// Додайте правильні заголовки MIME типів
app.get('*.css', function (req, res, next) {
    res.set('Content-Type', 'text/css');
    next();
});
app.get('*.js', function (req, res, next) {
    res.set('Content-Type', 'application/javascript');
    next();
});
// Створення та налаштування бази даних SQLite
const db = new sqlite3.Database('./database.db', (err) => {
    if (err) {
        console.error('Помилка підключення до бази даних:', err);
    } else {
        console.log('Підключено до бази даних SQLite');
        initDatabase();
    }
});
// Ініціалізація бази даних
function initDatabase() {
    db.serialize(() => {
        // Таблиця користувачів
        db.run(`CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      email TEXT UNIQUE NOT NULL,
      password TEXT NOT NULL,
      role TEXT DEFAULT 'client',
      phone TEXT,
      address TEXT,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )`);
        // Таблиця категорій
        db.run(`CREATE TABLE IF NOT EXISTS categories (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      description TEXT,
      image TEXT,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )`);
        // Таблиця товарів
        db.run(`CREATE TABLE IF NOT EXISTS products (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      description TEXT,
      price REAL NOT NULL,
      image TEXT,
      category_id INTEGER,
      in_stock BOOLEAN DEFAULT 1,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (category_id) REFERENCES categories (id)
```

```
    )`);
        // Таблиця замовлень
        db.run(`CREATE TABLE IF NOT EXISTS orders (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id INTEGER,
      total_price REAL NOT NULL,
      status TEXT DEFAULT 'pending',
      payment_id TEXT,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (user_id) REFERENCES users (id)
    )`);
        // Таблиця товарів у замовленні
        db.run(`CREATE TABLE IF NOT EXISTS order_items (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      order_id INTEGER,
      product_id INTEGER,
      quantity INTEGER DEFAULT 1,
      price REAL NOT NULL,
      FOREIGN KEY (order_id) REFERENCES orders (id),
      FOREIGN KEY (product_id) REFERENCES products (id)
    )`);
        // Таблиця історії переглядів
        db.run(`CREATE TABLE IF NOT EXISTS view_history (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id INTEGER,
      product_id INTEGER,
      viewed_at DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (user_id) REFERENCES users (id),
      FOREIGN KEY (product_id) REFERENCES products (id)
    )`);
        // Таблиця повідомлень чату
        db.run(`CREATE TABLE IF NOT EXISTS messages (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      sender_id INTEGER,
      receiver_id INTEGER,
      content TEXT,
      read BOOLEAN DEFAULT 0,
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (sender_id) REFERENCES users (id),
      FOREIGN KEY (receiver_id) REFERENCES users (id)
    )`);
        console.log('База даних ініціалізована успішно');
    });
}
// Middleware для аутентифікації
const auth = (req, res, next) => {
    const token = req.header('Authorization')?.replace('Bearer ', '');
    if (!token) {
        return res.status(401).json({ message: 'Необхідна аутентифікація' });
    }
    try {
        const decoded = jwt.verify(token, 'secretkey');
        req.user = decoded;
        // ВАЖЛИВО: дозволити всім авторизованим користувачам
        next();
    } catch (error) {
        res.status(401).json({ message: 'Недійсний токен' });
    }
};
// Middleware для перевірки ролі
const checkRole = (roles) => {
    return (req, res, next) => {
        if (!roles.includes(req.user.role)) {
            return res.status(403).json({ message: 'Доступ заборонено' });
        }
```

```
        next();
    };
};
// Ендпоінти для користувачів
app.post('/api/register', async (req, res) => {
    try {
        const { name, email, password, role = 'client' } = req.body;
        // Перевірка, чи існує користувач
        db.get('SELECT * FROM users WHERE email = ?', [email], async (err, user) => {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (user) {
                return res.status(400).json({ message: 'Користувач з такою електронною
поштою вже існує' });
            }
            // Хешування пароля
            const hashedPassword = await bcrypt.hash(password, 10);
            // Створення нового користувача
            db.run(
                'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
                [name, email, hashedPassword, role],
                function (err) {
                    if (err) {
                        return res.status(500).json({ message: 'Помилка сервера' });
                    }
                    const userId = this.lastID;
                    // Генерація JWT токена
                    const token = jwt.sign({ id: userId, role }, 'secretkey', {
expiresIn: '1d' });
                    res.status(201).json({
                        token,
                        user: { id: userId, name, email, role }
                    });
                }
            );
        });
    } catch (error) {
        res.status(500).json({ message: 'Помилка сервера' });
    }
});
app.post('/api/login', (req, res) => {
    try {
        const { email, password } = req.body;
        // Пошук користувача за email
        db.get('SELECT * FROM users WHERE email = ?', [email], async (err, user) => {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (!user) {
                return res.status(400).json({ message: 'Невірна електронна пошта або
пароль' });
            }
            // Перевірка пароля
            const isMatch = await bcrypt.compare(password, user.password);
            if (!isMatch) {
                return res.status(400).json({ message: 'Невірна електронна пошта або
пароль' });
            }
            // Генерація JWT токена
            const token = jwt.sign({ id: user.id, role: user.role }, 'secretkey', {
expiresIn: '1d' });
            res.json({
                token,
```

```
                user: { id: user.id, name: user.name, email: user.email, role:
user.role }
            });
        });
    } catch (error) {
        res.status(500).json({ message: 'Помилка сервера' });
    }
});

app.get('/api/me', auth, (req, res) => {
    db.get('SELECT id, name, email, role, phone, address FROM users WHERE id = ?',
[req.user.id], (err, user) => {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        if (!user) {
            return res.status(404).json({ message: 'Користувача не знайдено' });
        }
        res.json(user);
    });
});
app.put('/api/users/profile', auth, async (req, res) => {
    try {
        const { name, phone, address, password } = req.body;
        let updates = [];
        let params = [];
        if (name) {
            updates.push('name = ?');
            params.push(name);
        }
        if (phone) {
            updates.push('phone = ?');
            params.push(phone);
        }
        if (address) {
            updates.push('address = ?');
            params.push(address);
        }
        if (password) {
            updates.push('password = ?');
            params.push(await bcrypt.hash(password, 10));
        }
        if (updates.length === 0) {
            return res.status(400).json({ message: 'Немає даних для оновлення' });
        }
        params.push(req.user.id);
        db.run(
            `UPDATE users SET ${updates.join(', ')} WHERE id = ?`,
            params,
            function (err) {
                if (err) {
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                if (this.changes === 0) {
                    return res.status(404).json({ message: 'Користувача не знайдено'
});
                }
                res.json({ message: 'Профіль оновлено успішно' });
            }
        );
    } catch (error) {
        res.status(500).json({ message: 'Помилка сервера' });
    }
});
// Ендпоінти для категорій
app.get('/api/categories', (req, res) => {
```

```
    db.all('SELECT * FROM categories', (err, categories) => {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        res.json(categories);
    });
});
app.post('/api/categories', auth, checkRole(['manager']), (req, res) => {
    const { name, description, image } = req.body;
    db.run(
        'INSERT INTO categories (name, description, image) VALUES (?, ?, ?)',
        [name, description, image],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            db.get('SELECT * FROM categories WHERE id = ?', [this.lastID], (err,
category) => {
                if (err) {
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                res.status(201).json(category);
            });
        }
    );
});
app.put('/api/categories/:id', auth, checkRole(['manager']), (req, res) => {
    const { name, description, image } = req.body;
    const categoryId = req.params.id;
    db.run(
        'UPDATE categories SET name = ?, description = ?, image = ? WHERE id = ?',
        [name, description, image, categoryId],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (this.changes === 0) {
                return res.status(404).json({ message: 'Категорію не знайдено' });
            }
            db.get('SELECT * FROM categories WHERE id = ?', [categoryId], (err,
category) => {
                if (err) {
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                res.json(category);
            });
        }
    );
});
app.delete('/api/categories/:id', auth, checkRole(['manager']), (req, res) => {
    const categoryId = req.params.id;
    db.run('DELETE FROM categories WHERE id = ?', [categoryId], function (err) {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        if (this.changes === 0) {
            return res.status(404).json({ message: 'Категорію не знайдено' });
        }
        res.json({ message: 'Категорію видалено успішно' });
    });
});
// Ендпоінти для товарів
app.get('/api/products', (req, res) => {
    const { category, search, sort } = req.query;
```

```
    let query = 'SELECT p.*, c.name as category_name FROM products p LEFT JOIN
categories c ON p.category_id = c.id';
    let params = [];
    if (category) {
        query += ' WHERE p.category_id = ?';
        params.push(category);
    }
    if (search) {
        query += category ? ' AND' : ' WHERE';
        query += ' (p.name LIKE ? OR p.description LIKE ?)';
        params.push(`%${search}%`, `%${search}%`);
    }
    if (sort) {
        switch (sort) {
            case 'price_asc':
                query += ' ORDER BY p.price ASC';
                break;
            case 'price_desc':
                query += ' ORDER BY p.price DESC';
                break;
            case 'name_asc':
                query += ' ORDER BY p.name ASC';
                break;
            case 'name_desc':
                query += ' ORDER BY p.name DESC';
                break;
            case 'newest':
                query += ' ORDER BY p.created_at DESC';
                break;
            default:
                query += ' ORDER BY p.created_at DESC';
        }
    } else {
        query += ' ORDER BY p.created_at DESC';
    }
    db.all(query, params, (err, products) => {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        res.json(products);
    });
});
app.get('/api/products/:id',  (req, res) => {
    const productId = req.params.id;
    db.get(
        'SELECT p.*, c.name as category_name FROM products p LEFT JOIN categories c ON
p.category_id = c.id WHERE p.id = ?',
        [productId],
        (err, product) => {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (!product) {
                return res.status(404).json({ message: 'Товар не знайдено' });
            }
            // Зберігаємо історію перегляду для аутентифікованого користувача
            if (req.user && req.user.id && req.user.role === 'client') {
                db.run(
                    'INSERT INTO view_history (user_id, product_id) VALUES (?, ?)',
                    [req.user.id, productId],
                    (err) => {
                        if (err) {
                            console.error('Помилка при збереженні історії перегляду:',
err);
                        }
```

```
                    }
                );
            }
            res.json(product);
        }
    );
});
app.post('/api/products', auth, checkRole(['manager']), (req, res) => {
    const { name, description, price, image, category_id, in_stock } = req.body;
    db.run(
        'INSERT INTO products (name, description, price, image, category_id, in_stock)
VALUES (?, ?, ?, ?, ?, ?)',
        [name, description, price, image, category_id, in_stock ? 1 : 0],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            db.get('SELECT * FROM products WHERE id = ?', [this.lastID], (err,
product) => {
                if (err) {
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                res.status(201).json(product);
            });
        }
    );
});
app.put('/api/products/:id', auth, checkRole(['manager']), (req, res) => {
    const { name, description, price, image, category_id, in_stock } = req.body;
    const productId = req.params.id;
    db.run(
        'UPDATE products SET name = ?, description = ?, price = ?, image = ?,
category_id = ?, in_stock = ? WHERE id = ?',
        [name, description, price, image, category_id, in_stock ? 1 : 0, productId],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (this.changes === 0) {
                return res.status(404).json({ message: 'Товар не знайдено' });
            }
            db.get('SELECT * FROM products WHERE id = ?', [productId], (err, product)
=> {
                if (err) {
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                res.json(product);
            });
        }
    );
});
app.delete('/api/products/:id', auth, checkRole(['manager']), (req, res) => {
    const productId = req.params.id;
    db.run('DELETE FROM products WHERE id = ?', [productId], function (err) {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        if (this.changes === 0) {
            return res.status(404).json({ message: 'Товар не знайдено' });
        }
        res.json({ message: 'Товар видалено успішно' });
    });
});
// Ендпоінти для замовлень
app.post('/api/orders', auth, checkRole(['client']), (req, res) => {
```

```
    const { products, total_price } = req.body;
    if (!products || !Array.isArray(products) || products.length === 0) {
        return res.status(400).json({ message: 'Потрібно вказати товари для
замовлення' });
    }
    // Симуляція ID оплати
    const payment_id = 'payment_' + Date.now() + '_' + Math.floor(Math.random() *
1000);
    db.run(
        'INSERT INTO orders (user_id, total_price, payment_id, status) VALUES (?, ?,
?, ?)',
        [req.user.id, total_price, payment_id, 'pending'],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            const orderId = this.lastID;
            // Додаємо товари до замовлення
            const insertItem = (index) => {
                if (index >= products.length) {
                    // Всі товари додані успішно
                    return db.get('SELECT * FROM orders WHERE id = ?', [orderId],
(err, order) => {
                        if (err) {
                            return res.status(500).json({ message: 'Помилка сервера'
});
                        }
                        res.status(201).json({
                            order,
                            payment_id,
                            message: 'Замовлення створено успішно. Використовуйте
payment_id для оплати.'
                        });
                    });
                }
                const item = products[index];
                db.run(
                    'INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES (?, ?, ?, ?)',
                    [orderId, item.product_id, item.quantity, item.price],
                    (err) => {
                        if (err) {
                            return res.status(500).json({ message: 'Помилка сервера'
});
                        }
                        insertItem(index + 1);
                    }
                );
            };
            insertItem(0);
        }
    );
});
app.post('/api/payments/simulate', auth, (req, res) => {
    const { payment_id, order_id } = req.body;
    if (!payment_id || !order_id) {
        return res.status(400).json({ message: 'Необхідні payment_id та order_id' });
    }
    // Симуляція оплати
    setTimeout(() => {
        db.run(
            'UPDATE orders SET status = ? WHERE id = ? AND payment_id = ?',
            ['paid', order_id, payment_id],
            function (err) {
                if (err) {
```

```
                    return res.status(500).json({ message: 'Помилка сервера' });
                }
                if (this.changes === 0) {
                    return res.status(404).json({ message: 'Замовлення не знайдено'
});
                }
                res.json({
                    success: true,
                    message: 'Оплата пройшла успішно',
                    transaction_id: 'txn_' + Date.now()
                });
            }
        );
    }, 1000); // 1 секунда затримки для імітації процесу оплати
});
app.post('/api/products/:id/view', auth, (req, res) => {
    if (req.user.role === 'client') {
        db.run(
            'INSERT INTO view_history (user_id, product_id) VALUES (?, ?)',
            [req.user.id, req.params.id],
            (err) => {
                if (err) {
                    console.error('Помилка при збереженні історії перегляду:', err);
                }
                res.status(200).json({ success: true });
            }
        );
    } else {
        res.status(200).json({ success: true });
    }
});
app.get('/api/products/:id', (req, res) => {
    const productId = req.params.id;
    db.get(
        'SELECT p.*, c.name as category_name FROM products p LEFT JOIN categories c ON
p.category_id = c.id WHERE p.id = ?',
        [productId],
        (err, product) => {
                        if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (!product) {
                return res.status(404).json({ message: 'Товар не знайдено' });
            }
            res.json(product);
        }
    );
});
// Ендпоінти для отримання користувачів для чату
app.get('/api/chat/users', auth, (req, res) => {
    // Визначаємо запит в залежності від ролі користувача
    let query, params;
    if (req.user.role === 'client') {
        // Клієнт бачить тільки менеджерів
        query = 'SELECT id, name, email, role FROM users WHERE role = "manager"';
        params = [];
    } else if (req.user.role === 'manager') {
        // Менеджер бачить тільки клієнтів
        query = 'SELECT id, name, email, role FROM users WHERE role = "client"';
        params = [];
    } else {
        // Інші користувачі бачать всіх, крім себе
        query = 'SELECT id, name, email, role FROM users WHERE id != ?';
        params = [req.user.id];
    }
```

```
    db.all(query, params, (err, users) => {
        if (err) {
            console.error('Помилка при отриманні користувачів для чату:', err);
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        res.json(users);
    });
});
app.get('/api/orders', auth, (req, res) => {
    let query, params;
    if (req.user.role === 'manager') {
        // Менеджер бачить всі замовлення
        query = `
      SELECT o.*, u.name as user_name, u.email as user_email
      FROM orders o
      JOIN users u ON o.user_id = u.id
      ORDER BY o.created_at DESC
    `;
        params = [];
    } else {
        // Клієнт бачить лише свої замовлення
        query = `
      SELECT o.*
      FROM orders o
      WHERE o.user_id = ?
      ORDER BY o.created_at DESC
    `;
        params = [req.user.id];
    }
    db.all(query, params, (err, orders) => {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        // Для кожного замовлення отримуємо його товари
        const getOrderItems = (index) => {
            if (index >= orders.length) {
                return res.json(orders);
            }
            const order = orders[index];
           db.all(
                `SELECT oi.*, p.name, p.image
         FROM order_items oi
         JOIN products p ON oi.product_id = p.id
         WHERE oi.order_id = ?`,
                [order.id],
                (err, items) => {
                    if (err) {
                        return res.status(500).json({ message: 'Помилка сервера' });
                    }
                    order.items = items;
                    getOrderItems(index + 1);
                }
            );
        };
        getOrderItems(0);
    });
});
app.put('/api/orders/:id/status', auth, checkRole(['manager']), (req, res) => {
    const { status } = req.body;
    const orderId = req.params.id;
    if (!['pending', 'paid', 'shipped', 'delivered', 'cancelled'].includes(status)) {
        return res.status(400).json({ message: 'Недійсний статус замовлення' });
    }
    db.run(
        'UPDATE orders SET status = ? WHERE id = ?',
```

```
        [status, orderId],
        function (err) {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            if (this.changes === 0) {
                return res.status(404).json({ message: 'Замовлення не знайдено' });
            }
            res.json({ message: 'Статус замовлення оновлено успішно' });
        }
    );
});
// Ендпоінти для історії переглядів
app.get('/api/view-history', auth, checkRole(['client']), (req, res) => {
    db.all(
        `SELECT vh.*, p.name, p.price, p.image, c.name as category_name
     FROM view_history vh
     JOIN products p ON vh.product_id = p.id
     LEFT JOIN categories c ON p.category_id = c.id
     WHERE vh.user_id = ?
     ORDER BY vh.viewed_at DESC`,
        [req.user.id],
        (err, history) => {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            res.json(history);
        }
    );
});
app.get('/api/chatusers', auth, (req, res) => {
    // Визначаємо SQL-запит відповідно до ролі користувача
    let query, params;
    if (req.user.role === 'client') {
        // Клієнти бачать тільки менеджерів
        query = 'SELECT id, name, email, role FROM users WHERE role = ?';
        params = ['manager'];
    } else if (req.user.role === 'manager') {
        // Менеджери бачать тільки клієнтів
        query = 'SELECT id, name, email, role FROM users WHERE role = ?';
        params = ['client'];
    } else {
        // Інші ролі бачать всіх, крім себе
        query = 'SELECT id, name, email, role FROM users WHERE id != ?';
        params = [req.user.id];
    }
    // Додаємо логування для відстеження
        console.log('Chat Users API викликаний користувачем:', req.user.id, 'з
роллю:', req.user.role);
    console.log('SQL запит:', query, 'параметри:', params);
    // Виконуємо запит до бази даних
    db.all(query, params, (err, users) => {
        if (err) {
            console.error('Помилка при отриманні користувачів для чату:', err);
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        console.log('Знайдено користувачів для чату:', users.length);
        res.json(users);
    });
});
app.get('/api/categories/:id', (req, res) => {
    const categoryId = req.params.id;
    db.get('SELECT * FROM categories WHERE id = ?', [categoryId], (err, category) => {
        if (err) {
            console.error('Помилка при отриманні категорії:', err);
```

```
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        if (!category) {
            return res.status(404).json({ message: 'Категорію не знайдено' });
        }
        res.json(category);
    });
});
// Ендпоінти для управління користувачами (для менеджера)
app.get('/api/users', auth, checkRole(['manager']), (req, res) => {
    db.all(
        'SELECT id, name, email, role, phone, address, created_at FROM users WHERE
role = "client"',
        (err, users) => {
            if (err) {
                return res.status(500).json({ message: 'Помилка сервера' });
            }
            res.json(users);
        }
    );
});
app.delete('/api/users/:id', auth, checkRole(['manager']), (req, res) => {
    const userId = req.params.id;
    db.run('DELETE FROM users WHERE id = ? AND role = "client"', [userId], function
(err) {
        if (err) {
            return res.status(500).json({ message: 'Помилка сервера' });
        }
        if (this.changes === 0) {
            return res.status(404).json({ message: 'Користувача не знайдено або ви не
можете видалити цього користувача' });
        }
        res.json({ message: 'Користувача видалено успішно' });
    });
});
// Налаштування сокетів для чату
io.on('connection', (socket) => {
    console.log('Користувач підключився до чату:', socket.id);
    // Аутентифікація користувача через сокет
    socket.on('authenticate', (token) => {
        try {
            const decoded = jwt.verify(token, 'secretkey');
            socket.userId = decoded.id;
           socket.userRole = decoded.role;
            // Підключення до кімнати для цього користувача
            socket.join(`user_${decoded.id}`);
            console.log(`Користувач ${decoded.id} автентифікований`);
            // Отримання непрочитаних повідомлень
            if (decoded.role === 'client') {
                // Клієнт отримує повідомлення від всіх менеджерів
                db.all(
                    `SELECT m.*, u.name as sender_name
           FROM messages m
           JOIN users u ON m.sender_id = u.id
           WHERE m.receiver_id = ? AND m.read = 0`,
                    [decoded.id],
                    (err, messages) => {
                        if (err) {
                            return console.error('Помилка при отриманні повідомлень:',
err);
                        }
                        if (messages.length > 0) {
                            socket.emit('unread_messages', messages);
                        }
                    }
```

```
                );
            } else if (decoded.role === 'manager') {
                // Менеджер отримує повідомлення від всіх клієнтів
                db.all(
                    `SELECT m.*, u.name as sender_name
            FROM messages m
            JOIN users u ON m.sender_id = u.id
            WHERE m.receiver_id = ? AND m.read = 0`,
                    [decoded.id],
                    (err, messages) => {
                        if (err) {
                            return console.error('Помилка при отриманні повідомлень:',
err);
                        }
                        if (messages.length > 0) {
                            socket.emit('unread_messages', messages);
                        }
                    }
                );
            }
        } catch (error) {
            console.error('Помилка аутентифікації сокета:', error);
        }
    });
    // Отримання історії чату
    socket.on('get_chat_history', ({ receiver_id }) => {
        if (!socket.userId) {
            return socket.emit('error', { message: 'Необхідна аутентифікація' });
        }
        db.all(
            `SELECT m.*, u.name as sender_name
       FROM messages m
       JOIN users u ON m.sender_id = u.id
       WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND
m.receiver_id = ?)
       ORDER BY m.created_at ASC`,
            [socket.userId, receiver_id, receiver_id, socket.userId],
            (err, messages) => {
                if (err) {
                    return socket.emit('error', { message: 'Помилка сервера' });
                }
                socket.emit('chat_history', messages);
                // Позначаємо повідомлення як прочитані
                db.run(
                    'UPDATE messages SET read = 1 WHERE sender_id = ? AND receiver_id
= ? AND read = 0',
                    [receiver_id, socket.userId],
                    (err) => {
                        if (err) {
                            console.error('Помилка при оновленні статусу
повідомлень:', err);
                        }
                    });
            }
        );
   });
    // Надсилання нового повідомлення
    socket.on('send_message', ({ receiver_id, content }) => {
        if (!socket.userId) {
            return socket.emit('error', { message: 'Необхідна аутентифікація' });
        }
        // Перевірка вхідних даних
        if (!receiver_id || !content) {
            return socket.emit('error', { message: 'Необхідні receiver_id та content'
});
```

```
        }
        // Перевірка, щоб клієнт писав тільки менеджерам і навпаки
        db.get('SELECT role FROM users WHERE id = ?', [receiver_id], (err, user) => {
            if (err || !user) {
                return socket.emit('error', { message: 'Одержувача не знайдено' });
            }
            // Клієнт може писати тільки менеджерам
            if (socket.userRole === 'client' && user.role !== 'manager') {
                return socket.emit('error', { message: 'Ви можете писати повідомлення
тільки менеджерам' });
            }
            // Менеджер може писати тільки клієнтам
            if (socket.userRole === 'manager' && user.role !== 'client') {
                return socket.emit('error', { message: 'Ви можете писати повідомлення
тільки клієнтам' });
            }
            // Зберігаємо повідомлення
            db.run(
                'INSERT INTO messages (sender_id, receiver_id, content) VALUES (?, ?,
?)',
                [socket.userId, receiver_id, content],
                function (err) {
                    if (err) {
                        return socket.emit('error', { message: 'Помилка сервера' });
                    }
                    const messageId = this.lastID;
                    // Отримуємо ім'я відправника
                    db.get('SELECT name FROM users WHERE id = ?', [socket.userId],
(err, sender) => {
                        if (err) {
                            return socket.emit('error', { message: 'Помилка сервера'
});
                        }
                        const message = {
                            id: messageId,
                            sender_id: socket.userId,
                            receiver_id,
                            content,
                            read: 0,
                            created_at: new Date().toISOString(),
                            sender_name: sender.name
                        };
                        // Надсилаємо повідомлення обом сторонам
                        socket.emit('message', message);
                        io.to(`user_${receiver_id}`).emit('message', message);
                    });
                }
            );
        });
    });
    // Позначення повідомлень як прочитаних
    socket.on('mark_read', ({ message_ids }) => {
        if (!socket.userId || !Array.isArray(message_ids) || message_ids.length === 0)
{
            return;
        }
        const placeholders = message_ids.map(() => '?').join(',');
        db.run(
            `UPDATE messages SET read = 1 WHERE id IN (${placeholders}) AND
receiver_id = ?`,
            [...message_ids, socket.userId],
            (err) => {
                if (err) {
                    console.error('Помилка при позначенні повідомлень як прочитаних:',
err);
```

```
                }
            }
        );
    });
    socket.on('disconnect', () => {
        console.log('Користувач відключився від чату:', socket.id);
    });
});
// Запуск сервера
const PORT = process.env.PORT || 3001;
server.listen(PORT, () => {
    console.log(`Сервер запущено на порту ${PORT}`);
});
```

seed.js

```
// seed-database.js - Скрипт для заповнення бази даних тестовими даними
const sqlite3 = require('sqlite3').verbose();
const bcrypt = require('bcryptjs');
const path = require('path');
// Відкриваємо з'єднання з базою даних
const db = new sqlite3.Database('./database.db', (err) => {
    if (err) {
        console.error('Помилка підключення до бази даних:', err);
        process.exit(1);
    }
    console.log('Підключено до бази даних SQLite для заповнення тестовими даними');
});
// Функція для хешування паролів
async function hashPassword(password) {
    return await bcrypt.hash(password, 10);
}
// Основна функція для заповнення БД
async function seedDatabase() {
    try {
        console.log('Початок заповнення бази даних...');
        // Очищаємо таблиці перед заповненням
       await clearTables();
        // Заповнюємо таблиці даними
        await seedUsers();
        await seedCategories();
        await seedProducts();
        await seedMessages();
        console.log('База даних успішно заповнена тестовими даними!');
        process.exit(0);
    } catch (error) {
        console.error('Помилка при заповненні бази даних:', error);
        process.exit(1);
    }
}
// Функція для очищення таблиць
function clearTables() {
    return new Promise((resolve, reject) => {
        const tables = [
            'messages',
            'view_history',
            'order_items',
            'orders',
            'products',
            'categories',
            'users'
```

```
        ];
        // Відключаємо перевірку зовнішніх ключів
        db.run('PRAGMA foreign_keys = OFF', (err) => {
            if (err) return reject(err);
            // Починаємо транзакцію
            db.run('BEGIN TRANSACTION', (err) => {
                if (err) return reject(err);
                // Очищаємо кожну таблицю
                const promises = tables.map(table => {
                    return new Promise((resolve, reject) => {
                        db.run(`DELETE FROM ${table}`, (err) => {
                            if (err) reject(err);
                            else resolve();
                        });
                    });
                });
                // Після очищення всіх таблиць
                Promise.all(promises)
                    .then(() => {
                        // Скидаємо лічильники auto-increment
                        const resetPromises = tables.map(table => {
                            return new Promise((resolve, reject) => {
                                db.run(`DELETE FROM sqlite_sequence WHERE
name='${table}'`, (err) => {
                                    if (err) reject(err);
                                    else resolve();
                                });
                            });
                        });
                        return Promise.all(resetPromises);
                    })
                    .then(() => {
                        // Завершуємо транзакцію
                        db.run('COMMIT', (err) => {
                            if (err) return reject(err);
                            console.log('Всі таблиці очищено');
                            // Включаємо перевірку зовнішніх ключів
                            db.run('PRAGMA foreign_keys = ON', (err) => {
                                if (err) return reject(err);
                                resolve();
                            });
                        });
                    })
                    .catch(err => {
                        // Відкат транзакції в разі помилки
                        db.run('ROLLBACK', () => {
                            reject(err);
                        });
                    });
            });
        });
    });
}
// Функція для заповнення користувачів
async function seedUsers() {
    console.log('Заповнення користувачів...');
    // Створюємо адміністраторів
    const admins = [
        { name: 'Адміністратор 1', email: 'admin1@example.com', password: await
hashPassword('admin123'), role: 'manager' },
        { name: 'Адміністратор 2', email: 'admin2@example.com', password: await
hashPassword('admin123'), role: 'manager' }
    ];
    // Створюємо клієнтів
    const clients = [];
```

```
    for (let i = 1; i <= 15; i++) {
        clients.push({
            name: `Клієнт ${i}`,
            email: `client${i}@example.com`,
            password: await hashPassword('client123'),
            role: 'client',
            phone: `+38050${String(1000000 + i).substring(1)}`,
            address: `вул. Квіткова, ${i}, м. Київ`
        });
    }
    // Додаємо всіх користувачів
    const users = [...admins, ...clients];
    return new Promise((resolve, reject) => {
        // Починаємо транзакцію
        db.run('BEGIN TRANSACTION', (err) => {
            if (err) return reject(err);
            const promises = users.map(user => {
                return new Promise((resolve, reject) => {
                    db.run(
                        'INSERT INTO users (name, email, password, role, phone,
address) VALUES (?, ?, ?, ?, ?, ?)',
                        [user.name, user.email, user.password, user.role, user.phone
|| null, user.address || null],
                        function (err) {
                            if (err) reject(err);
                            else resolve(this.lastID);
                        }
                    );
                });
            });
            Promise.all(promises)
                .then(() => {
                    db.run('COMMIT', (err) => {
                        if (err) return reject(err);
                        console.log(`Додано ${users.length} користувачів`);
                        resolve();
                    });
                })
                .catch(err => {
                    db.run('ROLLBACK', () => {
                        reject(err);
                    });
                });
        });
    });
}
// Функція для заповнення категорій
function seedCategories() {
    console.log('Заповнення категорій...');
    const categories = [
        {
            name: 'Троянди',
            description: 'Вишукані та елегантні троянди різних кольорів та сортів',
            image: 'https://images.unsplash.com/photo-1559563362-
c667ba5f5480?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80'
        },
        {
            name: 'Букети',
            description: 'Готові композиції з різних квітів на будь-який смак',
            image: 'https://images.unsplash.com/photo-1589244159943-
460088ed5c92?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80'
```

```
        },
        {
            name: 'Тюльпани',
            description: 'Весняні тюльпани різноманітних кольорів та відтінків',
            image: 'https://images.unsplash.com/photo-1520763185298-
1b434c919102?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80'
        },
        {
            name: 'Кімнатні рослини',
            description: 'Різноманітні рослини для дому та офісу',
            image: 'https://images.unsplash.com/photo-1598880765941-
e5b7a4e969d3?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80'
        },
        {
            name: 'Весільні квіти',
            description: 'Особливі композиції для весільних церемоній',
            image: 'https://images.unsplash.com/photo-1525328304455-
a971a0162dbc?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80'
        }
    ];
    return new Promise((resolve, reject) => {
        db.run('BEGIN TRANSACTION', (err) => {
            if (err) return reject(err);
            const promises = categories.map(category => {
                return new Promise((resolve, reject) => {
                    db.run(
                        'INSERT INTO categories (name, description, image) VALUES (?,
?, ?)',
                        [category.name, category.description, category.image],
                        function (err) {
                            if (err) reject(err);
                            else resolve(this.lastID);
                        }
                    );
                });
            });
            Promise.all(promises)
                .then(() => {
                    db.run('COMMIT', (err) => {
                        if (err) return reject(err);
                        console.log(`Додано ${categories.length} категорій`);
                        resolve();
                    });
                })
                .catch(err => {
                    db.run('ROLLBACK', () => {
                        reject(err);
                    });
                });
        });
    });
}
// Функція для заповнення товарів
function seedProducts() {
    console.log('Заповнення товарів...');
    // Отримуємо категорії з БД, щоб використати їх ID
    return new Promise((resolve, reject) => {
        db.all('SELECT id FROM categories', (err, categories) => {
            if (err) return reject(err);
```

```
            if (categories.length === 0) return reject(new Error('Категорії не
знайдено'));
            const products = [
                // Категорія "Троянди"
                {
                    name: 'Червона троянда преміум',
                    description: 'Вишукана троянда насиченого червоного кольору.
Ідеальний подарунок для коханої людини.',
                    price: 120.00,
                    image: 'https://images.unsplash.com/photo-1518709779341-
56cf4535e94b?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Біла троянда',
                    description: 'Ніжна біла троянда символізує чистоту та невинність.
Ідеальна для весільних букетів.',
                    price: 110.00,
                    image: 'https://images.unsplash.com/photo-1548094990-
c16ca90f1f0d?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Рожева троянда',
                    description: 'Елегантна рожева троянда, що виражає вдячність і
захоплення.',
                    price: 115.00,
                    image: 'https://images.unsplash.com/photo-1550236520-
7050f3582da0?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Жовта троянда',
                    description: 'Сонячна жовта троянда символізує радість і дружбу.',
                    price: 105.00,
                    image: 'https://images.unsplash.com/photo-1590762877344-
1104f0e1e4d1?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Фіолетова троянда',
                    description: 'Рідкісна фіолетова троянда для особливих випадків.',
                    price: 140.00,
                    image: 'https://images.unsplash.com/photo-1571491242289-
d6f714d27727?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Червоні троянди (набір 7 шт)',
                    description: 'Набір з 7 червоних троянд преміум класу.',
```

```
                    price: 750.00,
                    image: 'https://images.unsplash.com/photo-1548094990-
c16ca90f1f0d?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                {
                    name: 'Мікс троянд (набір 5 шт)',
                    description: 'Набір з 5 троянд різних кольорів.',
                    price: 550.00,
                    image: 'https://images.unsplash.com/photo-1531594652722-
3229073863cf?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[0].id,
                    in_stock: true
                },
                // Категорія "Букети"
                {
                    name: 'Весняний букет',
                    description: 'Яскравий весняний букет з тюльпанів, нарцисів і
крокусів.',
                    price: 650.00,
                    image: 'https://images.unsplash.com/photo-1490349368332-
38177a9973c3?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Романтичний букет',
                    description: 'Ніжний букет з троянд, лілій та фрезій для
романтичного подарунка.',
                    price: 850.00,
                    image: 'https://images.unsplash.com/photo-1486914890041-
6b58f35fd8a5?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Святковий букет "Радість"',
                    description: 'Яскравий букет з різноманітних квітів для святкового
настрою.',
                    price: 750.00,
                    image: 'https://images.unsplash.com/photo-1537181534458-
45dcee76ae90?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Букет "Літній настрій"',
                    description: 'Яскравий букет з польових квітів, що нагадує про
літо.',
                    price: 550.00,
                    image: 'https://images.unsplash.com/photo-1594900689460-
fdbe0d6c455e?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
```

```
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Елегантний букет',
                    description: 'Вишуканий букет з орхідей та екзотичних квітів.',
                    price: 1100.00,
                    image: 'https://images.unsplash.com/photo-1561181286-
d3fee7d55364?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Букет "Сюрприз"',
                    description: 'Унікальний букет, складений флористом з сезонних
квітів.',
                    price: 600.00,
                    image: 'https://images.unsplash.com/photo-1563241527-
3004b7be0ffd?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                {
                    name: 'Букет "Вдячність"',
                    description: 'Ідеальний букет для висловлення подяки з троянд,
хризантем та гербер.',
                    price: 700.00,
                    image: 'https://images.unsplash.com/photo-1548385784-
7005d6e2953b?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[1].id,
                    in_stock: true
                },
                // Категорія "Тюльпани"
                {
                    name: 'Червоні тюльпани (10 шт)',
                    description: 'Букет з 10 червоних тюльпанів - класичний варіант
для подарунка.',
                    price: 350.00,
                    image: 'https://images.unsplash.com/photo-1617665146086-
c749cdde56ac?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Білі тюльпани (10 шт)',
                    description: 'Ніжні білі тюльпани, що символізують чистоту
намірів.',
                    price: 350.00,
                    image: 'https://images.unsplash.com/photo-1521533963861-
ea8b9b5834f2?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Рожеві тюльпани (10 шт)',
```

```
                    description: 'Чарівні рожеві тюльпани для весняного настрою.',
                    price: 350.00,
                    image: 'https://images.unsplash.com/photo-1582436520999-
71e7563103ea?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Жовті тюльпани (10 шт)',
                    description: 'Сонячні жовті тюльпани для яскравих емоцій.',
                    price: 350.00,
                    image: 'https://images.unsplash.com/photo-1599119567764-
b4338a1bfdee?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Фіолетові тюльпани (10 шт)',
                    description: 'Оригінальні фіолетові тюльпани для особливих
випадків.',
                    price: 400.00,
                    image: 'https://images.unsplash.com/photo-1615887627927-
26f7bffd2da4?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Мікс тюльпанів (15 шт)',
                    description: 'Яскравий мікс з 15 тюльпанів різних кольорів.',
                    price: 500.00,
                    image: 'https://images.unsplash.com/photo-1558704516-
f4b8602d8c90?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                {
                    name: 'Тюльпани пістрьові (7 шт)',
                    description: 'Унікальні тюльпани з пістрявими пелюстками для
цінителів.',
                    price: 420.00,
                    image: 'https://images.unsplash.com/photo-1615887627931-
1b4d12e7800e?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[2].id,
                    in_stock: true
                },
                // Категорія "Кімнатні рослини"
                {
                    name: 'Фікус бенджаміна',
                    description: 'Популярна кімнатна рослина з глянцевим листям, що
очищує повітря.',
                    price: 850.00,
                    image: 'https://images.unsplash.com/photo-1614594805320-
e6a5734a8bc3?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
```

```
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Орхідея фаленопсис',
                    description: 'Елегантна орхідея з великими квітами, що довго
цвіте.',
                    price: 750.00,
                    image: 'https://images.unsplash.com/photo-1590871282120-
85eb2da7363c?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Спатіфіллум (Жіноче щастя)',
                    description: 'Популярна кімнатна рослина з білими квітами, що
символізує жіноче щастя.',
                    price: 550.00,
                    image: 'https://images.unsplash.com/photo-1556608832-
579d070c3fb5?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Сукулент набір (3 шт)',
                    description: 'Набір з 3 різних сукулентів у декоративних
горщиках.',
                    price: 450.00,
                    image: 'https://images.unsplash.com/photo-1459156212016-
c812468e2115?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Монстера делікатесна',
                    description: 'Тропічна рослина з унікальним різьбленим листям.',
                    price: 950.00,
                    image: 'https://images.unsplash.com/photo-1614594895304-
fe7116ac3b58?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Сансевієрія (Тещин язик)',
                    description: 'Невибаглива рослина з вертикальним листям, ідеальна
для початківців.',
                    price: 500.00,
                    image: 'https://images.unsplash.com/photo-1620127807580-
990c3ecebd14?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                {
                    name: 'Антуріум',
                    description: 'Екзотична рослина з яскравими червоними квітами.',
```

```
                    price: 700.00,
                    image: 'https://images.unsplash.com/photo-1590622542930-
edae18648b43?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[3].id,
                    in_stock: true
                },
                // Категорія "Весільні квіти"
                {
                    name: 'Весільний букет "Ніжність"',
                    description: 'Розкішний букет для нареченої з білих троянд,
півоній та фрезій.',
                    price: 1500.00,
                    image: 'https://images.unsplash.com/photo-1529636798458-
92182e662485?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                                {
                    name: 'Весільний букет "Романтика"',
                    description: 'Елегантний букет для нареченої з пастельних троянд
та орхідей.',
                    price: 1700.00,
                    image: 'https://images.unsplash.com/photo-1556035511-
3168381ea4d4?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                {
                    name: 'Весільна бутоньєрка',
                    description: 'Елегантна бутоньєрка для нареченого, що гармонує з
букетом нареченої.',
                    price: 250.00,
                    image: 'https://images.unsplash.com/photo-1546011426-
49e135afd55d?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                {
                    name: 'Оформлення весільних арок',
                    description: 'Послуга з оформлення весільних арок живими квітами
та зеленню.',
                    price: 5000.00,
                    image: 'https://images.unsplash.com/photo-1506836467174-
27f1042ee3c7?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                {
                    name: 'Весільні композиції на столи',
                    description: 'Елегантні квіткові композиції для оформлення
святкових столів.',
                    price: 800.00,
                    image: 'https://images.unsplash.com/photo-1513076364437-
a1c5866e2b02?ixlib=rb-
```

```
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                {
                    name: 'Весільний букет "Класика"',
                    description: 'Класичний весільний букет з білих троянд.',
                    price: 1200.00,
                    image: 'https://images.unsplash.com/photo-1571986809942-
79718d9cf5cb?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                },
                {
                    name: 'Весільний букет "Екзотика"',
                    description: 'Оригінальний весільний букет з екзотичних квітів та
зелені.',
                    price: 2000.00,
                    image: 'https://images.unsplash.com/photo-1599051368235-
bcea578e3ad6?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1000&q=80',
                    category_id: categories[4].id,
                    in_stock: true
                }
            ];
            db.run('BEGIN TRANSACTION', (err) => {
                if (err) return reject(err);
                const promises = products.map(product => {
                    return new Promise((resolve, reject) => {
                        db.run(
                            'INSERT INTO products (name, description, price, image,
category_id, in_stock) VALUES (?, ?, ?, ?, ?, ?)',
                            [product.name, product.description, product.price,
product.image, product.category_id, product.in_stock ? 1 : 0],
                            function (err) {
                                if (err) reject(err);
                                else resolve(this.lastID);
                            }
                        );
                    });
                });
                Promise.all(promises)
                    .then(() => {
                        db.run('COMMIT', (err) => {
                            if (err) return reject(err);
                            console.log(`Додано ${products.length} товарів`);
                            resolve();
                        });
                    })
                    .catch(err => {
                        db.run('ROLLBACK', () => {
                            reject(err);
                        });
                    });
            });
        });
    });
}
// Функція для заповнення повідомлень чату
async function seedMessages() {
    console.log('Заповнення повідомлень чату...');
```

```
    return new Promise((resolve, reject) => {
        // Отримуємо користувачів з БД, щоб використати їх ID
        db.all('SELECT id, role FROM users', (err, users) => {
            if (err) return reject(err);
            if (users.length === 0) return reject(new Error('Користувачів не
знайдено'));
            // Відбираємо менеджерів та клієнтів
            const managers = users.filter(user => user.role === 'manager');
            const clients = users.filter(user => user.role === 'client');

            if (managers.length === 0 || clients.length === 0) {
                return reject(new Error('Недостатньо користувачів для створення
повідомлень'));
            }
            const messages = [];
            const now = new Date();
            // Для кожного клієнта генеруємо чат з менеджером
            clients.forEach(client => {
                // Вибираємо випадкового менеджера
                const manager = managers[Math.floor(Math.random() * managers.length)];
                // Час для перших повідомлень (декілька днів тому)
                let messageTime = new Date(now);
                messageTime.setDate(messageTime.getDate() - Math.floor(Math.random() *
10));
                // Початок розмови від клієнта
                messages.push({
                    sender_id: client.id,
                    receiver_id: manager.id,
                    content: 'Доброго дня! Хотів(-ла) би дізнатись про наявність
квітів для дня народження.',
                    created_at: messageTime.toISOString(),
                    read: 1
                });
                // Відповідь менеджера
                messageTime = new Date(messageTime);
                messageTime.setMinutes(messageTime.getMinutes() +
Math.floor(Math.random() * 30));
                messages.push({
                    sender_id: manager.id,
                    receiver_id: client.id,
                    content: 'Вітаю! Так, у нас є багато варіантів. Що саме Вас
цікавить: букет, композиція, кімнатна рослина?',
                    created_at: messageTime.toISOString(),
                    read: 1
                });
                // Відповідь клієнта
                messageTime = new Date(messageTime);
                messageTime.setMinutes(messageTime.getMinutes() +
Math.floor(Math.random() * 60));
                messages.push({
                    sender_id: client.id,
                    receiver_id: manager.id,
                    content: 'Думаю, букет буде найкращим варіантом. Яка ціна на
святкові букети?',
                    created_at: messageTime.toISOString(),
                    read: 1
                });
                // Відповідь менеджера
                messageTime = new Date(messageTime);
                messageTime.setMinutes(messageTime.getMinutes() +
Math.floor(Math.random() * 30));
                messages.push({
                    sender_id: manager.id,
                    receiver_id: client.id,
```

```
                    content: 'Ціни на святкові букети починаються від 500 грн. Ми
можемо запропонувати різні варіанти в залежності від бюджету. Який бюджет Ви
розглядаєте?',
                    created_at: messageTime.toISOString(),
                    read: 1
                });
                // Нещодавні повідомлення (вчора або сьогодні)
                messageTime = new Date(now);
                messageTime.setHours(messageTime.getHours() - Math.floor(Math.random()
* 24));
                messages.push({
                    sender_id: client.id,
                    receiver_id: manager.id,
                    content: 'Я хотів(-ла) би щось в межах 700-1000 грн. Можете
показати приклади таких букетів?',
                    created_at: messageTime.toISOString(),
                    read: 1
                });
                // Остання відповідь менеджера
                messageTime = new Date(messageTime);
                messageTime.setMinutes(messageTime.getMinutes() +
Math.floor(Math.random() * 30));
                messages.push({
                    sender_id: manager.id,
                    receiver_id: client.id,
                    content: 'Звичайно! В цій ціновій категорії у нас є чудові
варіанти. Можете переглянути їх у нашому каталозі в розділі "Букети". Також можемо
зробити індивідуальний букет відповідно до ваших побажань. Коли вам потрібні квіти?',
                    created_at: messageTime.toISOString(),
                    read: Math.random() > 0.5 ? 1 : 0 // Деякі повідомлення
непрочитані
                });
            });
            db.run('BEGIN TRANSACTION', (err) => {
                if (err) return reject(err);
                const promises = messages.map(message => {
                    return new Promise((resolve, reject) => {
                        db.run(
                            'INSERT INTO messages (sender_id, receiver_id, content,
read, created_at) VALUES (?, ?, ?, ?, ?)',
                            [message.sender_id, message.receiver_id, message.content,
message.read, message.created_at],
                            function (err) {
                                if (err) reject(err);
                                else resolve(this.lastID);
                            }
                        );
                    });
                });
                Promise.all(promises)
                    .then(() => {
                        db.run('COMMIT', (err) => {
                            if (err) return reject(err);
                            console.log(`Додано ${messages.length} повідомлень чату`);
                            resolve();
                        });
                    })
                    .catch(err => {
                        db.run('ROLLBACK', () => {
                            reject(err);
                        });
                    });
            });
        });
    });
```

```
}
// Запускаємо заповнення бази даних
seedDatabase();
```

index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Квіткова Лавка</title>
    <link rel="stylesheet" href="/styles.css">
    <!-- Socket.IO для чату -->
    <script src="https://cdn.socket.io/4.7.2/socket.io.min.js"></script>
</head>
<body>
    <div id="app">
        <!-- Шапка сайту -->
        <header class="header">
            <div class="container header-container">
                <div class="logo-container">
                    <div class="logo">
                        <a href="#" data-page="home">
                            <img id="logo-image" src="/logo.png" alt="Квіткова Лавка">
                        </a>
                    </div>
                </div>
                <button class="menu-toggle" id="menuToggle">
                    <span></span>
                    <span></span>
                    <span></span>
                </button>
                <nav class="nav" id="mainNav">
                    <ul class="nav-list">
                        <li class="nav-item">
                            <a href="#" data-page="home">Головна</a>
                        </li>
                        <li class="nav-item">
                            <a href="#" data-page="products">Каталог</a>
                        </li>
                        <!-- Динамічні елементи меню додаються через JavaScript -->
                        <li class="nav-item auth-item" id="loginMenuItem">
                            <a href="#" data-page="login">Вхід</a>
                        </li>
                        <li class="nav-item auth-item" id="registerMenuItem">
                            <a href="#" data-page="register">Реєстрація</a>
                        </li>
                    </ul>
                </nav>
            </div>
        </header>
        <!-- Головний контент -->
        <main class="main-content" id="mainContent">
            <!-- Контент буде додаватися динамічно через JavaScript -->
            <div class="loading">Завантаження...</div>
        </main>
        <!-- Підвал сайту -->
        <footer class="footer">
            <div class="container">
                <div class="footer-content">
                    <div class="footer-section">
                        <h3>Квіткова Лавка</h3>
                        <p>Ваш надійний партнер у світі квітів.</p>
```

```
                        <p>Ми допоможемо зробити будь-який день особливим!</p>
                    </div>
                    <div class="footer-section">
                        <h3>Контакти</h3>
                        <p>Телефон: +380 50 123 4567</p>
                        <p>Email: info@flower-shop.ua</p>
                        <p>Адреса: м. Київ, вул. Квіткова, 12</p>
                    </div>
                    <div class="footer-section">
                        <h3>Соціальні мережі</h3>
                        <div class="social-links">
                            <a href="#" class="social-link">Facebook</a>
                            <a href="#" class="social-link">Instagram</a>
                            <a href="#" class="social-link">Telegram</a>
                        </div>
                    </div>
                </div>
                <div class="footer-bottom">
                    <p>&copy; <span id="currentYear"></span> Квіткова Лавка. Усі права
захищені.</p>
                </div>
            </div>
        </footer>
    </div>
    <script>
        // Основні налаштування
        const API_URL = 'http://localhost:3001/api';
        let currentUser = null;
        let token = localStorage.getItem('token');
        let socket = null;
        let currentPage = 'home';
        let cartItems = JSON.parse(localStorage.getItem('cart') || '[]');
        // Елементи DOM
        const mainContent = document.getElementById('mainContent');
        const mainNav = document.getElementById('mainNav');
        const menuToggle = document.getElementById('menuToggle');
        const currentYearElement = document.getElementById('currentYear');
        const loginMenuItem = document.getElementById('loginMenuItem');
        const registerMenuItem = document.getElementById('registerMenuItem');
        // Встановлюємо поточний рік у футері
        currentYearElement.textContent = new Date().getFullYear();
        // Ініціалізація додатку
        document.addEventListener('DOMContentLoaded', () => {
            // Ініціалізуємо меню
            initMenu();
            // Перевіряємо автентифікацію
            checkAuth();
            // Додаємо обробники подій
            addEventListeners();
            // Завантажуємо початкову сторінку
            loadPage(getInitialPage());
        });
        // Функція для ініціалізації меню
        function initMenu() {
            menuToggle.addEventListener('click', () => {
                mainNav.classList.toggle('open');
            });
            // Закриваємо меню при кліку за його межами
            document.addEventListener('click', (e) => {
                if (!mainNav.contains(e.target) && !menuToggle.contains(e.target) &&
mainNav.classList.contains('open')) {
                    mainNav.classList.remove('open');
                }
            });
        }
```

```
// Функція для перевірки автентифікації
async function checkAuth() {
    if (token) {
        try {
            const response = await fetch(`${API_URL}/me`, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            if (response.ok) {
                currentUser = await response.json();
                initSocket();
                updateAuthUI();
            } else {
                token = null;
                localStorage.removeItem('token');
                updateAuthUI();
            }
        } catch (error) {
            console.error('Помилка при перевірці автентифікації:', error);
            token = null;
            localStorage.removeItem('token');
            updateAuthUI();
        }
    } else {
        updateAuthUI();
    }
}
// Функція для ініціалізації сокету
function initSocket() {
    if (token && !socket) {
        socket = io('http://localhost:3001');
        socket.emit('authenticate', token);
        // Обробка повідомлень
        socket.on('message', (message) => {
            // Додаємо нове повідомлення до чату, якщо сторінка чату відкрита
            if (currentPage === 'chat') {
                appendMessage(message);
            } else {
                // Можна додати сповіщення про нове повідомлення
                console.log('Нове повідомлення:', message);
            }
        });
        // Обробка непрочитаних повідомлень
        socket.on('unread_messages', (messages) => {
            console.log('Отримано непрочитані повідомлення:', messages);
            // Можна додати сповіщення або індикатор непрочитаних повідомлень
        });
        // Обробка помилок
        socket.on('error', (error) => {
            console.error('Помилка сокета:', error);
            alert(`Помилка підключення до чату: ${error.message}`);
        });
    }
}
// Функція для оновлення інтерфейсу відповідно до статусу автентифікації
function updateAuthUI() {
    const navList = document.querySelector('.nav-list');
    // Видаляємо всі елементи меню, які стосуються автентифікації
    document.querySelectorAll('.auth-item, .user-item').forEach(item => {
        item.remove();
    });
    if (currentUser) {
        // Додаємо пункти меню для автентифікованих користувачів
        let menuItems = '';
```

```
                if (currentUser.role === 'client') {
                    const cartCount = cartItems.reduce((total, item) => total +
item.quantity, 0);
                    menuItems += `
                                <li class="nav-item user-item">
                                    <a href="#" data-page="cart">Кошик ${cartCount > 0
? `<span class="cart-badge">${cartCount}</span>` : ''}</a>
                                </li>
                            `;
                }
                menuItems += `
                            <li class="nav-item user-item">
                                <a href="#" data-page="profile">Профіль</a>
                            </li>
                            <li class="nav-item user-item">
                                <a href="#" data-page="orders">Замовлення</a>
                            </li>
                            <li class="nav-item user-item">
                                <a href="#" data-page="chat">Чат</a>
                            </li>
                        `;
                if (currentUser.role === 'manager') {
                    menuItems += `
                                <li class="nav-item user-item dropdown">
                                    <span class="dropdown-toggle">Адмін</span>
                                    <ul class="dropdown-menu">
                                        <li><a href="#" data-page="admin-
products">Товари</a></li>
                                        <li><a href="#" data-page="admin-
categories">Категорії</a></li>
                                        <li><a href="#" data-page="admin-
users">Користувачі</a></li>
                                    </ul>
                                </li>
                            `;
                }
                menuItems += `
                            <li class="nav-item user-item">
                                <a href="#" id="logoutBtn">Вихід</a>
                            </li>
                        `;
                navList.insertAdjacentHTML('beforeend', menuItems);
                // Додаємо обробник для кнопки виходу
                document.getElementById('logoutBtn').addEventListener('click', (e) =>
{
                    e.preventDefault();
                    logout();
                });
            } else {
                // Додаємо пункти меню для неавтентифікованих користувачів
                const menuItems = `
                            <li class="nav-item auth-item">
                                <a href="#" data-page="login">Вхід</a>
                            </li>
                            <li class="nav-item auth-item">
                                <a href="#" data-page="register">Реєстрація</a>
                            </li>
                        `;
                navList.insertAdjacentHTML('beforeend', menuItems);
            }
            // Оновлюємо обробники подій для нових елементів меню
            addEventListeners();
        }
        // Функція додавання обробників подій
        function addEventListeners() {
```

```
    // Обробники для навігації
    document.querySelectorAll('a[data-page]').forEach(link => {
        link.addEventListener('click', (e) => {
            e.preventDefault();
            const page = e.currentTarget.getAttribute('data-page');
            loadPage(page);
        });
    });
}

// Функція виходу з аккаунту
function logout() {
    token = null;
    currentUser = null;
   localStorage.removeItem('token');
    if (socket) {
        socket.disconnect();
        socket = null;
    }
    updateAuthUI();
    loadPage('home');
}
// Функція визначення початкової сторінки
function getInitialPage() {
    // Тут можна додати логіку роботи з URL якщо потрібно
    return 'home';
}
// Функція завантаження сторінки
function loadPage(page) {
    currentPage = page;
    mainContent.innerHTML = '<div class="loading">Завантаження...</div>';
    switch (page) {
        case 'home':
            loadHomePage();
            break;
        case 'products':
            loadProductsPage();
            break;
        case 'login':
            loadLoginPage();
            break;
        case 'register':
            loadRegisterPage();
            break;
        case 'cart':
            if (currentUser && currentUser.role === 'client') {
                loadCartPage();
            } else {
                loadLoginPage();
            }
            break;
        case 'profile':
            if (currentUser) {
                loadProfilePage();
            } else {
                loadLoginPage();
            }
            break;
        case 'orders':
            if (currentUser) {
                loadOrdersPage();
            } else {
                loadLoginPage();
            }
            break;
        case 'chat':
```

```
                    if (currentUser) {
                        loadChatPage();
                    } else {
                        loadLoginPage();
                    }
                    break;
                case 'admin-products':
                    if (currentUser && currentUser.role === 'manager') {
                        loadAdminProductsPage();
                    } else {
                        loadHomePage();
                    }
                    break;
                case 'admin-categories':
                    if (currentUser && currentUser.role === 'manager') {
                        loadAdminCategoriesPage();
                    } else {
                        loadHomePage();
                    }
                    break;
                case 'admin-users':
                    if (currentUser && currentUser.role === 'manager') {
                        loadAdminUsersPage();
                    } else {
                        loadHomePage();
                    }
                    break;
                default:
                    if (page.startsWith('product-')) {
                        const productId = page.split('-')[1];
                        loadProductDetailPage(productId);
                    } else {
                        loadNotFoundPage();
                    }
            }
        }

        // Функція для завантаження головної сторінки
        async function loadHomePage() {
            try {
                // Завантажуємо категорії та популярні товари
                const [categoriesResponse, featuredProductsResponse] = await
Promise.all([
                    fetch(`${API_URL}/categories`),
                    fetch(`${API_URL}/products?limit=6`)
                ]);
                const categories = await categoriesResponse.json();
                const featuredProducts = await featuredProductsResponse.json();
                const html = `
                            <div class="home-page">
                                <section class="hero">
                                    <div class="container">
                                        <div class="hero-content">
                                            <h1>Ласкаво просимо до Квіткової
Лавки</h1>
                                            <p>Тут ви знайдете найкращі квіти для
будь-якого випадку</p>
                                            <a href="#" class="btn btn-primary" data-
page="products">Переглянути каталог</a>
                                        </div>
                                    </div>
                                </section>
                                <section class="categories section">
                                    <div class="container">
                                        <h2 class="section-title">Категорії</h2>
                                        <div class="categories-grid">
```

```
                                        ${categories.map(category => `
                                            <div class="category-card">
                                                <a href="#" data-
category="${category.id}" class="category-link">
                                                    <div class="category-image">
                                                        <img src="${category.image
|| 'https://via.placeholder.com/300'}" alt="${category.name}">
                                                    </div>
                                                    <h3 class="category-
name">${category.name}</h3>
                                                </a>
                                            </div>
                                        `).join('')}
                                    </div>
                                </div>
                            </section>
                            <section class="featured-products section">
                                <div class="container">
                                    <h2 class="section-title">Популярні
товари</h2>
                                    <div class="products-grid">
                                        ${featuredProducts.map(product => `
                                            <div class="product-card">
                                                <a href="#" data-
product="${product.id}" class="product-link">
                                                    <div class="product-image">
                                                        <img src="${product.image
|| 'https://via.placeholder.com/300'}" alt="${product.name}">
                                                    </div>
                                                    <div class="product-info">
                                                        <h3 class="product-
name">${product.name}</h3>
                                                        <p class="product-
price">${product.price.toFixed(2)} грн</p>
                                                    </div>
                                                </a>
                                            </div>
                                        `).join('')}
                                    </div>
                                    <div class="text-center mt-4">
                                        <a href="#" class="btn btn-secondary"
data-page="products">Переглянути всі товари</a>
                                    </div>
                                </div>
                            </section>
                        </div>
                    `;
                mainContent.innerHTML = html;
                // Додаємо обробники подій для нових елементів
                document.querySelectorAll('.category-link').forEach(link => {
                    link.addEventListener('click', (e) => {
                        e.preventDefault();
                        const categoryId = e.currentTarget.getAttribute('data-
category');
                        loadPage('products');
                        // Встановлюємо фільтр за категорією після завантаження
сторінки
                        setTimeout(() => {
                            const categorySelect =
document.getElementById('category');
                            if (categorySelect) {
                                categorySelect.value = categoryId;
                                categorySelect.dispatchEvent(new Event('change'));
                            }
                        }, 100);
```

```
                    });
                });
                document.querySelectorAll('.product-link').forEach(link => {
                    link.addEventListener('click', (e) => {
                        e.preventDefault();
                        const productId = e.currentTarget.getAttribute('data-
product');
                        loadPage(`product-${productId}`);
                    });
                });
                // Перепідключаємо обробники для навігації
                addEventListeners();
            } catch (error) {
                console.error('Помилка при завантаженні головної сторінки:', error);
                mainContent.innerHTML = '<div class="error">Помилка при завантаженні
даних. Спробуйте пізніше.</div>';
            }
        }
        // Функція для завантаження сторінки каталогу товарів
        async function loadProductsPage() {
            try {
                // Отримуємо категорії для фільтрування
                const categoriesResponse = await fetch(`${API_URL}/categories`);
                const categories = await categoriesResponse.json();
                // Відображаємо форму фільтрації та результати
                const html = `
                            <div class="catalog-page">
                                <div class="container">
                                    <h1 class="page-title">Каталог товарів</h1>
                                    <div class="filters-panel">
                                        <form class="search-form" id="searchForm">
                                            <input type="text" name="search"
id="searchInput" placeholder="Пошук товарів...">
                                            <button type="submit" class="btn btn-
primary">Пошук</button>
                                        </form>
                                        <div class="filter-controls">
                                            <div class="filter-item">
                                                <label
for="category">Категорія:</label>
                                                <select id="category" name="category">
                                                    <option value="">Усі
категорії</option>
                                                    ${categories.map(category => `
                                                        <option
value="${category.id}">${category.name}</option>
                                                    `).join('')}
                                                </select>
                                            </div>
                                            <div class="filter-item">
                                                <label for="sort">Сортування:</label>
                                                <select id="sort" name="sort">
                                                    <option
value="newest">Найновіші</option>
                                                    <option value="price_asc">Ціна:
від низької до високої</option>
                                                    <option value="price_desc">Ціна:
від високої до низької</option>
                                                    <option value="name_asc">Назва: А-
Я</option>
                                                    <option value="name_desc">Назва:
Я-А</option>
                                                </select>
                                            </div>
                                        </div>
```

```
                                    </div>
                                    <div id="productsContainer" class="products-grid">
                                        <div class="loading">Завантаження
товарів...</div>
                                    </div>
                                </div>
                            </div>
                        `;
                mainContent.innerHTML = html;
                // Отримуємо контейнер для товарів
                const productsContainer =
document.getElementById('productsContainer');
                // Функція для завантаження товарів з фільтрами
                async function fetchProducts(filters = {}) {
                    try {
                        productsContainer.innerHTML = '<div
class="loading">Завантаження товарів...</div>';
                        const queryParams = new URLSearchParams();
                        if (filters.category) queryParams.append('category',
filters.category);
                        if (filters.search) queryParams.append('search',
filters.search);
                        if (filters.sort) queryParams.append('sort', filters.sort);
                        const response = await
fetch(`${API_URL}/products?${queryParams.toString()}`);
                        const products = await response.json();
                        if (products.length === 0) {
                            productsContainer.innerHTML = '<div class="no-
results">Товари не знайдено</div>';
                            return;
                        }
                        const productsHtml = products.map(product => `
                                    <div class="product-card">
                                        <a href="#" data-product="${product.id}"
class="product-link">
                                            <div class="product-image">
                                                <img src="${product.image ||
'https://via.placeholder.com/300'}" alt="${product.name}">
                                            </div>
                                            <div class="product-info">
                                                <h3 class="product-
name">${product.name}</h3>
                                                <p class="product-
price">${product.price.toFixed(2)} грн</p>
                                                <p class="product-
category">${product.category_name || 'Без категорії'}</p>
                                            </div>
                                        </a>
                                    </div>
                                `).join('');
                        productsContainer.innerHTML = productsHtml;
                        // Додаємо обробники подій для нових елементів
                                    document.querySelectorAll('.product-
link').forEach(link => {
                        link.addEventListener('click', (e) => {
                            e.preventDefault();
                            const productId = e.currentTarget.getAttribute('data-
product');
                            loadPage(`product-${productId}`);
                        });
                    });
                } catch (error) {
                    console.error('Помилка при завантаженні товарів:', error);
                    productsContainer.innerHTML = '<div class="error">Помилка при
завантаженні товарів</div>';
```

```
            }
        }
        // Початкове завантаження товарів
        fetchProducts();
        // Додаємо обробники подій для фільтрів
        document.getElementById('searchForm').addEventListener('submit', (e) => {
            e.preventDefault();
            const search = document.getElementById('searchInput').value;
            const category = document.getElementById('category').value;
            const sort = document.getElementById('sort').value;
            fetchProducts({ search, category, sort });
        });
        document.getElementById('category').addEventListener('change', () => {
            const search = document.getElementById('searchInput').value;
            const category = document.getElementById('category').value;
            const sort = document.getElementById('sort').value;
            fetchProducts({ search, category, sort });
        });
        document.getElementById('sort').addEventListener('change', () => {
            const search = document.getElementById('searchInput').value;
            const category = document.getElementById('category').value;
            const sort = document.getElementById('sort').value;
            fetchProducts({ search, category, sort });
        });
    } catch (error) {
        console.error('Помилка при завантаженні сторінки каталогу:', error);
        mainContent.innerHTML = '<div class="error">Помилка при завантаженні
каталогу</div>';
    }
}
// Функція для додавання товару в кошик
function addToCart(product, quantity) {
    // Перевіряємо, чи користувач авторизований
    if (!currentUser) {
        loadPage('login');
        return;
    }
    // Перевіряємо, чи користувач є клієнтом
    if (currentUser.role !== 'client') {
        alert('Тільки клієнти можуть додавати товари в кошик');
        return;
    }
    // Отримуємо поточний стан кошика з localStorage
    const cart = JSON.parse(localStorage.getItem('cart') || '[]');
    // Перевіряємо, чи товар вже є в кошику
    const existingItemIndex = cart.findIndex(item => item.product_id === product.id);
    if (existingItemIndex !== -1) {
        // Оновлюємо кількість, якщо товар вже є в кошику
        cart[existingItemIndex].quantity += quantity;
    } else {
        // Додаємо новий товар
        cart.push({
            product_id: product.id,
            name: product.name,
            price: product.price,
            image: product.image || 'https://via.placeholder.com/100',
            quantity: quantity
        });
    }
    // Зберігаємо оновлений кошик
    localStorage.setItem('cart', JSON.stringify(cart));
    // Оновлюємо відображення кількості товарів у кошику в меню
    updateCartBadge();
    // Показуємо повідомлення користувачу
    alert(`Товар "${product.name}" додано до кошика в кількості ${quantity} шт.`);
```

```
}
// Функція для оновлення відображення кількості товарів у кошику
function updateCartBadge() {
    // Перевіряємо, чи користувач авторизований та є клієнтом
    if (currentUser && currentUser.role === 'client') {
        const cart = JSON.parse(localStorage.getItem('cart') || '[]');
        const totalItems = cart.reduce((sum, item) => sum + item.quantity, 0);
        // Знаходимо всі елементи з класом cart-badge
        const cartBadges = document.querySelectorAll('.cart-badge');
        // Оновлюємо їх вміст
        cartBadges.forEach(badge => {
            if (totalItems > 0) {
                badge.textContent = totalItems;
                badge.style.display = 'inline-block';
            } else {
                badge.style.display = 'none';
            }
        });
    }
}
// Функція для завантаження сторінки товару
        async function loadProductDetailPage(productId) {
    try {
        const mainContent = document.getElementById('mainContent');
        mainContent.innerHTML = '<div class="loading">Завантаження...</div>';
        // Спробуємо використати ендпоінт для отримання списку всіх товарів і знайти
потрібний
        const response = await fetch(`${API_URL}/products`);
        if (!response.ok) {
            throw new Error(`Помилка запиту: ${response.status}`);
        }
        const products = await response.json();
        const product = products.find(p => p.id == productId);
        if (!product) {
            throw new Error('Товар не знайдено');
        }
        // Відображення деталей товару
        const html = `
    <div class="product-detail-page">
        <div class="container">
            <h1 class="page-title">${product.name}</h1>
            <div class="product-detail">
                <div class="product-image">
                    <img src="${product.image || 'https://via.placeholder.com/500'}"
alt="${product.name}">
                </div>
                <div class="product-info">
                    <h2 class="product-name">${product.name}</h2>
                    <p class="product-price">${product.price.toFixed(2)} грн</p>
                    <p class="product-category">Категорія: ${product.category_name ||
'Без категорії'}</p>
                    <div class="product-description">
                        <h3>Опис</h3>
                        <p>${product.description || 'Опис відсутній'}</p>
                    </div>
                    ${product.in_stock ? `
                        <div class="product-actions">
                            <div class="quantity-control">
                                <label for="quantity">Кількість:</label>
                                <input type="number" id="quantity" min="1" value="1">
                            </div>
                            <button class="btn btn-primary btn-add-to-cart"
id="addToCartBtn">
                                Додати до кошика
                            </button>
```

```
                        </div>
                    ` : `
                        <div class="out-of-stock">
                            <p>Товар тимчасово відсутній</p>
                        </div>
                    `}
                </div>
            </div>
        </div>
    </div>
`;
        mainContent.innerHTML = html;
        // Додаємо обробник події для кнопки "Додати до кошика"
        if (product.in_stock) {
            document.getElementById('addToCartBtn').addEventListener('click', () => {
                if (!currentUser) {
                    loadPage('login');
                    return;
                }
                if (currentUser.role !== 'client') {
                    alert('Тільки клієнти можуть додавати товари в кошик');
                    return;
                }
                const quantity = parseInt(document.getElementById('quantity').value);
                if (isNaN(quantity) || quantity < 1) {
                    alert('Будь ласка, вкажіть дійсну кількість');
                    return;
                }
                // Додаємо товар до кошика
                addToCart(product, quantity);
            });
        }
        // Якщо користувач авторизований, окремо зберігаємо історію перегляду
        if (token && currentUser && currentUser.role === 'client') {
            fetch(`${API_URL}/products/${productId}/view`, {
                method: 'POST',
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            }).catch(e => console.error('Помилка збереження історії:', e));
        }
        // Ініціалізуємо анімації та паралакс ефекти для сторінки
        initPageAnimations();
        handleParallax();
    } catch (error) {
       console.error('Помилка при завантаженні сторінки товару:', error);
        // Відображаємо повідомлення про помилку
        mainContent.innerHTML = `
    <div class="container">
        <div class="error-container" style="text-align: center; padding: 50px 20px;">
            <h2>Товар не знайдено або виникла помилка</h2>
            <p>На жаль, не вдалося завантажити дані про товар. Будь ласка, спробуйте
пізніше.</p>
            <div style="margin-top: 30px;">
                <a href="#" class="btn btn-primary" onclick="loadPage('products');
return false;">Повернутися до каталогу</a>
            </div>
        </div>
    </div>
`;
    }
}// Функція для завантаження сторінки "Не знайдено"
function loadNotFoundPage() {
    const html = `
                <div class="not-found-page">
```

```
                    <div class="container">
                        <h1>404</h1>
                        <h2>Сторінку не знайдено</h2>
                        <p>Сторінка, яку ви шукаєте, не існує або була переміщена</p>
                        <a href="#" class="btn btn-primary" data-page="home">На
головну</a>
                    </div>
                </div>
            `;
    mainContent.innerHTML = html;
    addEventListeners();
}
// Функція для завантаження сторінки входу
function loadLoginPage() {
    const html = `
                <div class="auth-page">
                    <div class="container">
                        <div class="auth-form-container">
                            <h2>Вхід в аккаунт</h2>
                            <div id="loginError" class="alert alert-danger"
style="display: none;"></div>
                            <form class="auth-form" id="loginForm">
                                <div class="form-group">
                                    <label for="email">Email</label>
                                    <input type="email" id="email" name="email"
required>
                                </div>
                                <div class="form-group">
                                    <label for="password">Пароль</label>
                                    <input type="password" id="password"
name="password" required>
                                </div>
                                <button type="submit" class="btn btn-primary w-100"
id="loginBtn">Увійти</button>
                            </form>
                            <div class="auth-links">
                                <p>Немає аккаунту? <a href="#" data-
page="register">Зареєструватися</a></p>
                            </div>
                        </div>
                    </div>
                </div>
        `;
    mainContent.innerHTML = html;
    // Додаємо обробник для форми входу
    document.getElementById('loginForm').addEventListener('submit', async (e) => {
        e.preventDefault();
                const email = document.getElementById('email').value;
                const password = document.getElementById('password').value;
                const loginBtn = document.getElementById('loginBtn');
                const loginError = document.getElementById('loginError');
                loginBtn.disabled = true;
                loginBtn.textContent = 'Завантаження...';
                loginError.style.display = 'none';
                try {
                    const response = await fetch(`${API_URL}/login`, {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json'
                        },
                        body: JSON.stringify({ email, password })
                    });
                    const data = await response.json();
                    if (response.ok) {
                        token = data.token;
```

```
                        localStorage.setItem('token', token);
                        currentUser = data.user;
                        initSocket();
                        updateAuthUI();
                        loadPage('home');
                    } else {
                        loginError.textContent = data.message || 'Помилка при вході.
Перевірте введені дані.';
                        loginError.style.display = 'block';
                    }
                } catch (error) {
                    console.error('Помилка при вході:', error);
                    loginError.textContent = 'Помилка при вході. Спробуйте пізніше.';
                    loginError.style.display = 'block';
                } finally {
                    loginBtn.disabled = false;
                    loginBtn.textContent = 'Увійти';
                }
            });
            addEventListeners();
        }
        // Функція для завантаження сторінки реєстрації
        function loadRegisterPage() {
            const html = `
                    <div class="auth-page">
                        <div class="container">
                            <div class="auth-form-container">
                                <h2>Реєстрація</h2>
                                <div id="registerError" class="alert alert-danger"
style="display: none;"></div>
                                <form class="auth-form" id="registerForm">
                                    <div class="form-group">
                                        <label for="reg-name">Ім'я</label>
                                        <input type="text" id="reg-name" name="name"
required>
                                    </div>
                                    <div class="form-group">
                                        <label for="reg-email">Email</label>
                                        <input type="email" id="reg-email"
name="email" required>
                                    </div>
                                    <div class="form-group">
                                        <label for="reg-password">Пароль</label>
                                        <input type="password" id="reg-password"
name="password" required>
                                    </div>
                                    <div class="form-group">
                                        <label for="reg-confirm-password">Підтвердіть
пароль</label>
                                        <input type="password" id="reg-confirm-
password" name="confirmPassword" required>
                                    </div>
                                    <div class="form-group">
                                        <label for="reg-role">Роль</label>
                                        <select id="reg-role" name="role">
                                            <option value="client">Клієнт</option>
                                            <option value="manager">Менеджер</option>
                                        </select>
                                    </div>
                                    <button type="submit" class="btn btn-primary w-
100" id="registerBtn">Зареєструватися</button>
                                </form>
                                <div class="auth-links">
                                    <p>Вже є аккаунт? <a href="#" data-
page="login">Увійти</a></p>
```

```
                                </div>
                            </div>
                        </div>
                    </div>
                `;
            mainContent.innerHTML = html;

            // Додаємо обробник для форми реєстрації
            document.getElementById('registerForm').addEventListener('submit', async
(e) => {
                e.preventDefault();
                const name = document.getElementById('reg-name').value;
                const email = document.getElementById('reg-email').value;
                const password = document.getElementById('reg-password').value;
                const confirmPassword = document.getElementById('reg-confirm-
password').value;
                const role = document.getElementById('reg-role').value;
                const registerBtn = document.getElementById('registerBtn');
                const registerError = document.getElementById('registerError');
                // Перевірка валідності введених даних
                if (password !== confirmPassword) {
                    registerError.textContent = 'Паролі не співпадають';
                    registerError.style.display = 'block';
                    return;
                }
                registerBtn.disabled = true;
                registerBtn.textContent = 'Завантаження...';
                registerError.style.display = 'none';
                try {
                    const response = await fetch(`${API_URL}/register`, {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json'
                        },
                        body: JSON.stringify({ name, email, password, role })
                    });
                    const data = await response.json();
                    if (response.ok) {
                        token = data.token;
                        localStorage.setItem('token', token);
                        currentUser = data.user;
                        initSocket();
                        updateAuthUI();
                        loadPage('home');
                    } else {
                        registerError.textContent = data.message || 'Помилка при
реєстрації';
                        registerError.style.display = 'block';
                    }
                } catch (error) {
                    console.error('Помилка при реєстрації:', error);
                    registerError.textContent = 'Помилка при реєстрації. Спробуйте
пізніше.';
                    registerError.style.display = 'block';
                } finally {
                    registerBtn.disabled = false;
                    registerBtn.textContent = 'Зареєструватися';
                }
            });
            addEventListeners();
        }
        // Функція для завантаження сторінки кошика
        function loadCartPage() {
            if (!currentUser || currentUser.role !== 'client') {
                loadPage('login');
                return;
```

```
            }
            // Оновлюємо дані кошика з localStorage
            cartItems = JSON.parse(localStorage.getItem('cart') || '[]');
            // Розраховуємо загальну вартість
            const totalPrice = cartItems.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
            const html = `
                    <div class="cart-page">
                        <div class="container">
                            <h1 class="page-title">Кошик</h1>
                            <div id="cartError" class="alert alert-danger"
style="display: none;"></div>
                            ${cartItems.length === 0 ? `
                                <div class="empty-cart">
                                    <p>Ваш кошик порожній</p>
                                    <a href="#" class="btn btn-primary" data-
page="products">Перейти до каталогу</a>
                                </div>
                            ` : `
                                <div class="cart-items">
                                    ${cartItems.map((item, index) => `
                                        <div class="cart-item" data-index="${index}">
                                            <div class="item-image">
                                                <img src="${item.image ||
'https://via.placeholder.com/100'}" alt="${item.name}">
                                            </div>
                                            <div class="item-details">
                                                <h3 class="item-
name">${item.name}</h3>
                                                <p class="item-
price">${item.price.toFixed(2)} грн</p>
                                            </div>
                                            <div class="item-quantity">
                                                <button class="btn btn-sm btn-outline
decrease-quantity">-</button>
                                                <span>${item.quantity}</span>
                                                <button class="btn btn-sm btn-outline
increase-quantity">+</button>
                                            </div>
                                            <div class="item-total">
                                                <p>${(item.price *
item.quantity).toFixed(2)} грн</p>
                                            </div>
                                            <button class="btn btn-sm btn-danger
remove-item">Видалити</button>
                                        </div>
                                    `).join('')}
                                </div>
                                <div class="cart-summary">
                                    <div class="cart-total">
                                        <h3>Загальна сума:</h3>
                                        <p>${totalPrice.toFixed(2)} грн</p>
                                    </div>
                                    <div class="cart-actions">
                                        <button class="btn btn-outline"
id="clearCartBtn">Очистити кошик</button>
                                        <button class="btn btn-primary"
id="checkoutBtn">Оформити замовлення</button>
                                    </div>
                                </div>
                            `}
                        </div>
                    </div>
                `;
            mainContent.innerHTML = html;
```

```
            if (cartItems.length > 0) {
                // Обробники подій для кошика
                document.querySelectorAll('.increase-quantity').forEach(button => {
                    button.addEventListener('click', (e) => {
                        const index = parseInt(e.target.closest('.cart-
item').getAttribute('data-index'));
                        updateCartItemQuantity(index, cartItems[index].quantity + 1);
                    });
                });
                document.querySelectorAll('.decrease-quantity').forEach(button => {
                    button.addEventListener('click', (e) => {
                        const index = parseInt(e.target.closest('.cart-
item').getAttribute('data-index'));
                        if (cartItems[index].quantity > 1) {
                            updateCartItemQuantity(index, cartItems[index].quantity -
1);
                        }
                    });
                });
                document.querySelectorAll('.remove-item').forEach(button => {
                    button.addEventListener('click', (e) => {
                        const index = parseInt(e.target.closest('.cart-
item').getAttribute('data-index'));
                        removeCartItem(index);
                    });
                });
                document.getElementById('clearCartBtn').addEventListener('click', ()
=> {
                    clearCart();
                });
                document.getElementById('checkoutBtn').addEventListener('click', () =>
{
                    checkout();
                });
            }
            addEventListeners();
        }
        // Функція для оновлення кількості товару в кошику
        function updateCartItemQuantity(index, newQuantity) {
            if (newQuantity < 1) return;
            cartItems[index].quantity = newQuantity;
            localStorage.setItem('cart', JSON.stringify(cartItems));
            // Перезавантажуємо сторінку кошика з оновленими даними
            loadCartPage();
        }
        // Функція для видалення товару з кошика
        function removeCartItem(index) {
            cartItems.splice(index, 1);
            localStorage.setItem('cart', JSON.stringify(cartItems));
            // Перезавантажуємо сторінку кошика з оновленими даними
            loadCartPage();
            // Оновлюємо інтерфейс (відображення кількості товарів у меню)
            updateAuthUI();
        }
        // Функція для очищення кошика
        function clearCart() {
            cartItems = [];
            localStorage.setItem('cart', JSON.stringify(cartItems));
            // Перезавантажуємо сторінку кошика з оновленими даними
            loadCartPage();
            // Оновлюємо інтерфейс
            updateAuthUI();
        }
        // Функція для оформлення замовлення
        async function checkout() {
```

```
            if (cartItems.length === 0) {
                document.getElementById('cartError').textContent = 'Кошик порожній';
                document.getElementById('cartError').style.display = 'block';
                return;
            }

            const checkoutBtn = document.getElementById('checkoutBtn');
            const cartError = document.getElementById('cartError');
            checkoutBtn.disabled = true;
            checkoutBtn.textContent = 'Обробка...';
            cartError.style.display = 'none';
            try {
                // Формуємо дані для створення замовлення
                const totalPrice = cartItems.reduce((sum, item) => sum + (item.price *
item.quantity), 0);
                const orderData = {
                    products: cartItems.map(item => ({
                        product_id: item.product_id,
                        quantity: item.quantity,
                        price: item.price
                    })),
                    total_price: totalPrice
                };
                // Створюємо замовлення
                const orderResponse = await fetch(`${API_URL}/orders`, {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                        'Authorization': `Bearer ${token}`
                    },
                    body: JSON.stringify(orderData)
                });
                const orderResult = await orderResponse.json();
                if (orderResponse.ok) {
                    // Симуляція оплати
                    const paymentResponse = await
fetch(`${API_URL}/payments/simulate`, {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json',
                            'Authorization': `Bearer ${token}`
                        },
                        body: JSON.stringify({
                            payment_id: orderResult.payment_id,
                            order_id: orderResult.order.id
                        })
                    });
                    const paymentResult = await paymentResponse.json();
                    if (paymentResponse.ok) {
                        // Очищаємо кошик після успішного замовлення
                        clearCart();
                        // Показуємо повідомлення про успіх
                        alert('Замовлення успішно оформлено та оплачено!');
                        // Перенаправляємо на сторінку замовлень
                        loadPage('orders');
                    } else {
                        throw new Error(paymentResult.message || 'Помилка при оплаті
замовлення');
                    }
                } else {
                    throw new Error(orderResult.message || 'Помилка при створенні
замовлення');
                }
            } catch (error) {
                console.error('Помилка при оформленні замовлення:', error);
```

```
                cartError.textContent = error.message || 'Помилка при оформленні
замовлення';
                cartError.style.display = 'block';
            } finally {
                if (checkoutBtn) {
                    checkoutBtn.disabled = false;
                    checkoutBtn.textContent = 'Оформити замовлення';
                }
            }
        }
        // Функція для завантаження сторінки профілю
        function loadProfilePage() {
            if (!currentUser) {
                loadPage('login');
                return;
            }
            const html = `
                    <div class="profile-page">
                        <div class="container">
                            <h1 class="page-title">Мій профіль</h1>
                            <div class="profile-tabs">
                                <div class="profile-tab active" data-
tab="profile">Особисті дані</div>
                                ${currentUser.role === 'client' ? `<div
class="profile-tab" data-tab="history">Історія переглядів</div>` : ''}
                            </div>
                            <div class="profile-content">
                                <div class="profile-tab-content active" id="profile-
tab">
                                    <div class="profile-form-container">
                                        <div id="profileMessage" class="alert alert-
success" style="display: none;"></div>
                                        <div id="profileError" class="alert alert-
danger" style="display: none;"></div>
                                        <form class="profile-form" id="profileForm">
                                            <div class="form-group">
                                                <label for="profile-name">Ім'я</label>
                                                <input type="text" id="profile-name"
name="name" value="${currentUser.name || ''}">
                                            </div>
                                            <div class="form-group">
                                                <label for="profile-
email">Email</label>
                                                <input type="email" id="profile-email"
value="${currentUser.email || ''}" disabled>
                                                <small class="form-text text-
muted">Email не можна змінити</small>
                                            </div>
                                            <div class="form-group">
                                                <label for="profile-
phone">Телефон</label>
                                                <input type="tel" id="profile-phone"
name="phone" value="${currentUser.phone || ''}">
                                            </div>
                                            <div class="form-group">
                                                <label for="profile-
address">Адреса</label>
                                                <textarea id="profile-address"
name="address" rows="3">${currentUser.address || ''}</textarea>
                                            </div>
                                            <div class="form-group">
                                                <label for="profile-password">Новий
пароль</label>
                                                <input type="password" id="profile-
password" name="password">
```

```
                                                <small class="form-text text-
muted">Залиште пустим, якщо не хочете змінювати пароль</small>
                                            </div>
                                            <div class="form-group">
                                                <label for="profile-confirm-
password">Підтвердження нового пароля</label>
                                                <input type="password" id="profile-
confirm-password" name="confirmPassword">
                                            </div>
                                           <button type="submit" class="btn btn-
primary" id="saveProfileBtn">Зберегти зміни</button>
                                        </form>
                                    </div>
                                </div>
                                <div class="profile-tab-content" id="history-tab"
style="display: none;">
                                    <h2>Історія переглядів</h2>
                                    <div id="viewHistoryContainer">
                                        <div class="loading">Завантаження історії
переглядів...</div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                `;
            mainContent.innerHTML = html;

            // Перемикання між вкладками
            document.querySelectorAll('.profile-tab').forEach(tab => {
                tab.addEventListener('click', (e) => {
                    const tabName = e.currentTarget.getAttribute('data-tab');
                    // Змінюємо активну вкладку
                    document.querySelectorAll('.profile-tab').forEach(t =>
t.classList.remove('active'));
                    e.currentTarget.classList.add('active');
                    // Показуємо відповідний вміст
                    document.querySelectorAll('.profile-tab-content').forEach(content
=> {
                        content.style.display = 'none';
                    });
                    if (tabName === 'profile') {
                        document.getElementById('profile-tab').style.display =
'block';
                    } else if (tabName === 'history') {
                        document.getElementById('history-tab').style.display =
'block';
                        loadViewHistory();
                    }
                });
            });
            // Обробник форми профілю
            document.getElementById('profileForm').addEventListener('submit', async
(e) => {
                e.preventDefault();
                const name = document.getElementById('profile-name').value;
                const phone = document.getElementById('profile-phone').value;
                const address = document.getElementById('profile-address').value;
                const password = document.getElementById('profile-password').value;
                const confirmPassword = document.getElementById('profile-confirm-
password').value;
                const saveBtn = document.getElementById('saveProfileBtn');
                const profileMessage = document.getElementById('profileMessage');
                const profileError = document.getElementById('profileError');
                // Перевірка паролів
                if (password && password !== confirmPassword) {
```

```
                    profileError.textContent = 'Паролі не співпадають';
                    profileError.style.display = 'block';
                    profileMessage.style.display = 'none';
                    return;
                }
                saveBtn.disabled = true;
                saveBtn.textContent = 'Збереження...';
                profileError.style.display = 'none';
                profileMessage.style.display = 'none';
                try {
                    // Формуємо дані для оновлення
                    const updateData = { name, phone, address };
                    if (password) {
                        updateData.password = password;
                    }
                    const response = await fetch(`${API_URL}/users/profile`, {
                        method: 'PUT',
                        headers: {
                            'Content-Type': 'application/json',
                            'Authorization': `Bearer ${token}`
                        },
                        body: JSON.stringify(updateData)
                    });
                    const data = await response.json();
                    if (response.ok) {
                        // Оновлюємо дані користувача
                        Object.assign(currentUser, updateData);
                        // Очищаємо поля пароля
                        document.getElementById('profile-password').value = '';
                        document.getElementById('profile-confirm-password').value =
''';

                        // Показуємо повідомлення про успіх
                        profileMessage.textContent = 'Профіль успішно оновлено';
                        profileMessage.style.display = 'block';
                    } else {
                        throw new Error(data.message || 'Помилка при оновленні
профілю');
                    }
                } catch (error) {
                    console.error('Помилка при оновленні профілю:', error);
                    profileError.textContent = error.message || 'Помилка при оновленні
профілю';
                    profileError.style.display = 'block';
                } finally {
                    saveBtn.disabled = false;
                    saveBtn.textContent = 'Зберегти зміни';
                }
            });
            addEventListeners();
        }

        // Функція для завантаження історії переглядів
        async function loadViewHistory() {
            if (!currentUser || currentUser.role !== 'client') return;
            const viewHistoryContainer =
document.getElementById('viewHistoryContainer');
            viewHistoryContainer.innerHTML = '<div class="loading">Завантаження
історії переглядів...</div>';
            try {
                const response = await fetch(`${API_URL}/view-history`, {
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    throw new Error('Помилка при завантаженні історії переглядів');
```

```
                }
                const viewHistory = await response.json();
                if (viewHistory.length === 0) {
                    viewHistoryContainer.innerHTML = '<p class="no-history">Ви ще не
переглядали товари</p>';
                    return;
                }
                const historyHtml = `
                        <div class="products-grid">
                            ${viewHistory.map(item => `
                                <div class="product-card">
                                    <a href="#" data-product="${item.product_id}"
class="product-link">
                                        <div class="product-image">
                                            <img src="${item.image ||
'https://via.placeholder.com/300'}" alt="${item.name}">
                                        </div>
                                        <div class="product-info">
                                            <h3 class="product-name">${item.name}</h3>
                                            <p class="product-
price">${item.price.toFixed(2)} грн</p>
                                            <p class="product-
category">${item.category_name || 'Без категорії'}</p>
                                            <p class="view-date">
                                                Переглянуто: ${new
Date(item.viewed_at).toLocaleString('uk-UA')}
                                            </p>
                                        </div>
                                    </a>
                                </div>
                            `).join('')}
                        </div>
                    `;
                viewHistoryContainer.innerHTML = historyHtml;
                // Додаємо обробники подій для посилань на товари
                document.querySelectorAll('.product-link').forEach(link => {
                    link.addEventListener('click', (e) => {
                        e.preventDefault();
                        const productId = e.currentTarget.getAttribute('data-
product');
                        loadPage(`product-${productId}`);
                    });
                });
            } catch (error) {
                console.error('Помилка при завантаженні історії переглядів:', error);
                viewHistoryContainer.innerHTML = '<div class="error">Помилка при
завантаженні історії переглядів</div>';
            }
        }
        // Функція для завантаження сторінки замовлень
        async function loadOrdersPage() {
            if (!currentUser) {
                loadPage('login');
                return;
            }
            const html = `
        <div class="orders-page">
            <div class="container">
                <h1 class="page-title">
                    ${currentUser.role === 'manager' ? 'Всі замовлення' : 'Мої
замовлення'}
                </h1>
                <div id="ordersContainer">
                    <div class="loading">Завантаження замовлень...</div>
                </div>
```

```
            </div>
        </div>
    `;
            mainContent.innerHTML = html;
            try {
                const response = await fetch(`${API_URL}/orders`, {
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    throw new Error('Помилка при завантаженні замовлень');
                }
                const orders = await response.json();
                const ordersContainer = document.getElementById('ordersContainer');
                if (orders.length === 0) {
                    ordersContainer.innerHTML = '<div class="no-orders">Замовлень не
знайдено</div>';
                    return;
                }
                const ordersHtml = `
            <div class="orders-list">
                ${orders.map(order => `
                    <div class="order-card">
                        <div class="order-header">
                            <h3 class="order-id">
                                Замовлення #${order.id}
                            </h3>
                            <span class="order-status
${getStatusClass(order.status)}">
                                ${getStatusText(order.status)}
                            </span>
                        </div>

                        <div class="order-info">
                            <p class="order-date">
                                Дата: ${new Date(order.created_at).toLocaleString('uk-
UA')}
                            </p>
                            <p class="order-total">
                                Сума: ${order.total_price.toFixed(2)} грн
                            </p>

                            ${currentUser.role === 'manager' ? `
                                <p class="order-client">
                                    Клієнт: ${order.user_name} (${order.user_email})
                                </p>

                                <div class="order-status-control">
                                    <label>Змінити статус:</label>
                                    <select class="status-select" data-order-
id="${order.id}">
                                        <option value="pending" ${order.status ===
'pending' ? 'selected' : ''}>В обробці</option>
                                        <option value="paid" ${order.status === 'paid'
? 'selected' : ''}>Оплачено</option>
                                        <option value="shipped" ${order.status ===
'shipped' ? 'selected' : ''}>Відправлено</option>
                                        <option value="delivered" ${order.status ===
'delivered' ? 'selected' : ''}>Доставлено</option>
                                        <option value="cancelled" ${order.status ===
'cancelled' ? 'selected' : ''}>Скасовано</option>
                                    </select>
                                </div>
                            ` : ''}
```

```
                        </div>

                        <div class="order-items">
                            <h4>Товари в замовленні:</h4>
                            ${order.items && order.items.map(item => `
                                <div class="order-item">
                                    <div class="item-image">
                                        <img src="${item.image ||
'https://via.placeholder.com/50'}" alt="${item.name}">
                                    </div>
                                    <div class="item-info">
                                        <h5 class="item-name">${item.name}</h5>
                                        <p class="item-price">${item.price.toFixed(2)}
грн x ${item.quantity}</p>
                                    </div>
                                    <div class="item-total">
                                        ${(item.price * item.quantity).toFixed(2)} грн
                                    </div>
                                </div>
                            `).join('')}
                        </div>
                    </div>
                `).join('')}
            </div>
        `;

                ordersContainer.innerHTML = ordersHtml;
                // Додаємо обробники подій для зміни статусу замовлення (для
менеджерів)
                if (currentUser.role === 'manager') {
                    document.querySelectorAll('.status-select').forEach(select => {
                        select.addEventListener('change', async (e) => {
                            const orderId = e.target.getAttribute('data-order-id');
                            const newStatus = e.target.value;
                            try {
                                const response = await
fetch(`${API_URL}/orders/${orderId}/status`, {
                                    method: 'PUT',
                                    headers: {
                                        'Content-Type': 'application/json',
                                        'Authorization': `Bearer ${token}`
                                    },
                                    body: JSON.stringify({ status: newStatus })
                                });
                                if (response.ok) {
                                    // Оновлюємо сторінку замовлень
                                    loadOrdersPage();
                                } else {
                                    const data = await response.json();
                                    throw new Error(data.message || 'Помилка при
оновленні статусу замовлення');
                                }
                            } catch (error) {
                                alert(error.message);
                            }
                        });
                    });
                }
            } catch (error) {
                console.error('Помилка при завантаженні замовлень:', error);
                document.getElementById('ordersContainer').innerHTML = `
            <div class="error">Помилка при завантаженні замовлень:
${error.message}</div>
        `;
            }
        }
```

```
        // Функції для відображення статусу замовлення
        function getStatusText(status) {
            switch (status) {
                case 'pending': return 'В обробці';
                case 'paid': return 'Оплачено';
                case 'shipped': return 'Відправлено';
                case 'delivered': return 'Доставлено';
                case 'cancelled': return 'Скасовано';
                default: return 'Невідомий статус';
            }
        }
        function getStatusClass(status) {
            switch (status) {
                case 'pending': return 'status-pending';
                case 'paid': return 'status-paid';
                case 'shipped': return 'status-shipped';
                case 'delivered': return 'status-delivered';
                case 'cancelled': return 'status-cancelled';
                default: return '';
            }
        }
        // Функція для завантаження сторінки чату
        async function loadChatPage() {
            if (!currentUser) {
                loadPage('login');
                return;
            }
            if (!socket && token) {
                initSocket();
            }
            const html = `
    <div class="chat-page">
        <div class="container">
            <h1 class="page-title">Чат</h1>

            <div class="chat-container">
                <div class="chat-sidebar">
                    <h3>Контакти</h3>
                    <div id="contactsList">
                        <div class="loading">Завантаження контактів...</div>
                    </div>
                </div>

                <div class="chat-main">
                    <div class="chat-messages" id="chatMessages">
                        <div class="no-contact-selected">
                            <p>Оберіть контакт зі списку зліва, щоб почати
спілкування</p>
                        </div>
                    </div>
                    <div class="chat-input">
                        <input type="text" id="messageInput" placeholder="Напишіть
повідомлення..." disabled>
                        <button id="sendMessageBtn" disabled>Надіслати</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
    `;
            mainContent.innerHTML = html;
            try {
                // Використовуємо новий ендпоінт спеціально для чату, без тире
                console.log('Намагаємося отримати список користувачів для чату...');
                const response = await fetch(`${API_URL}/chatusers`, {
```

```
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                console.log('Статус відповіді:', response.status);
                if (!response.ok) {
                    throw new Error(`Помилка запиту: ${response.status}`);
                }
                const contacts = await response.json();
                console.log('Отримано контакти для чату:', contacts);
                const contactsList = document.getElementById('contactsList');
                if (!contacts || contacts.length === 0) {
                    contactsList.innerHTML = '<p class="no-contacts">Контактів не
знайдено</p>';
                    return;
                }

                const contactsHtml = `
            <ul class="contacts-list">
                ${contacts.map(contact => `
                    <li class="contact-item" data-user-id="${contact.id}" data-user-
name="${contact.name}">
                        <div class="contact-info">
                            <h4 class="contact-name">${contact.name}</h4>
                            <p class="contact-email">${contact.email}</p>
                        </div>
                    </li>
                `).join('')}
            </ul>
        `;
                contactsList.innerHTML = contactsHtml;
                // Додаємо обробники для вибору контакту
                document.querySelectorAll('.contact-item').forEach(item => {
                    item.addEventListener('click', (e) => {
                        document.querySelectorAll('.contact-item').forEach(i =>
i.classList.remove('active'));
                        e.currentTarget.classList.add('active');
                        const userId = e.currentTarget.getAttribute('data-user-id');
                        const userName = e.currentTarget.getAttribute('data-user-
name');
                        // Зберігаємо поточного співрозмовника
                        currentChatPartner = {
                            id: userId,
                            name: userName
                        };
                        // Завантажуємо історію чату
                        loadChatHistory(userId);
                        // Активуємо поле введення
                        document.getElementById('messageInput').disabled = false;
                        document.getElementById('sendMessageBtn').disabled = false;
                    });
                });
                // Налаштовуємо обробники для надсилання повідомлень
                document.getElementById('sendMessageBtn').addEventListener('click',
sendMessage);
                document.getElementById('messageInput').addEventListener('keypress',
(e) => {
                    if (e.key === 'Enter') sendMessage();
                });
            } catch (error) {
                console.error('Помилка при завантаженні чату:', error);
                document.getElementById('contactsList').innerHTML = `
            <div class="error">
                <p>Помилка при завантаженні контактів: ${error.message}</p>
                <button class="btn btn-primary" onclick="loadChatPage()">Спробувати
знову</button>
```

```
            </div>
        `;
            }
        }
        // Функція для завантаження історії чату з конкретним користувачем
        function loadChatHistory(userId) {
            if (!socket) {
                console.error('Сокет не ініціалізовано');
                return;
            }
            const chatMessages = document.getElementById('chatMessages');
            chatMessages.innerHTML = '<div class="loading">Завантаження
повідомлень...</div>';
            // Запитуємо історію чату через сокет
            socket.emit('get_chat_history', { receiver_id: userId });
            // Обробник отримання історії чату
            socket.once('chat_history', (messages) => {
                displayChatMessages(messages);
            });
            // Обробник помилок
            socket.once('error', (error) => {
                chatMessages.innerHTML = `<div class="error">Помилка:
${error.message}</div>`;
            });
        }
        // Функція для надсилання повідомлення
        function sendMessage() {
            if (!socket || !currentChatPartner) return;
            const input = document.getElementById('messageInput');
           const content = input.value.trim();
            if (content) {
                // Надсилаємо повідомлення через сокет
                socket.emit('send_message', {
                    receiver_id: currentChatPartner.id,
                    content: content
                });
                // Очищаємо поле введення
                input.value = '';
            }
        }
        // Функція для відображення повідомлень чату
        function displayChatMessages(messages) {
            const chatMessages = document.getElementById('chatMessages');
            if (!messages || messages.length === 0) {
                chatMessages.innerHTML = '<div class="no-messages"><p>Немає
повідомлень. Почніть розмову!</p></div>';
                return;
            }
            const messagesHtml = messages.map(message => `
        <div class="message ${message.sender_id == currentUser.id ? 'own-message' :
'other-message'}">
            <div class="message-content">
                ${message.content}
            </div>
            <div class="message-info">
                <span class="message-time">
                    ${new Date(message.created_at).toLocaleTimeString('uk-UA')}
                </span>
                ${message.sender_id == currentUser.id ? `
                    <span class="message-status">
                        ${message.read ? '✓✓' : '✓'}
                    </span>
                ` : ''}
            </div>
        </div>
```

```
    `).join('');
            chatMessages.innerHTML = messagesHtml;
            // Прокручуємо до останнього повідомлення
            chatMessages.scrollTop = chatMessages.scrollHeight;
        }
        // Функція для додавання нового повідомлення в чат
        function appendMessage(message) {
            // Перевіряємо, чи відноситься повідомлення до поточного чату
            if (!currentChatPartner || (message.sender_id != currentUser.id &&
message.sender_id != currentChatPartner.id) &&
                (message.receiver_id != currentUser.id && message.receiver_id !=
currentChatPartner.id)) {
                return;
            }
            const chatMessages = document.getElementById('chatMessages');
            if (!chatMessages) return;
            // Якщо немає повідомлень, очищаємо контейнер
            if (chatMessages.querySelector('.no-messages')) {
                chatMessages.innerHTML = '';
            }
            const messageHtml = `
        <div class="message ${message.sender_id == currentUser.id ? 'own-message' :
'other-message'}">
            <div class="message-content">
                ${message.content}
            </div>
            <div class="message-info">
                <span class="message-time">
                    ${new Date(message.created_at).toLocaleTimeString('uk-UA')}
                </span>
                ${message.sender_id == currentUser.id ? `
                    <span class="message-status">
                        ${message.read ? '✓✓' : '✓'}
                    </span>
                ` : ''}
            </div>
        </div>
    `;
            chatMessages.insertAdjacentHTML('beforeend', messageHtml);
            // Прокручуємо до останнього повідомлення
            chatMessages.scrollTop = chatMessages.scrollHeight;
            // Якщо повідомлення від іншого користувача, позначаємо його як прочитане
            if (message.sender_id != currentUser.id && socket) {
                socket.emit('mark_read', { message_ids: [message.id] });
            }
        }

        // Функція для завантаження сторінки адміністрування товарів
        async function loadAdminProductsPage() {
            if (!currentUser || currentUser.role !== 'manager') {
                loadPage('home');
                return;
            }
            const html = `
        <div class="admin-products-page">
            <div class="container">
                <div class="admin-header">
                    <h1 class="page-title">Управління товарами</h1>
                    <button class="btn btn-primary" id="addProductBtn">Додати
товар</button>
                </div>

                <div id="productsTableContainer">
                    <div class="loading">Завантаження товарів...</div>
                </div>
```

```
                <div id="productModal" class="modal" style="display: none;">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h2 id="productModalTitle">Додати товар</h2>
                            <button class="close-btn"
id="closeProductModal">&times;</button>
                        </div>

                        <form class="product-form" id="productForm">
                            <input type="hidden" id="product-id">

                            <div class="form-group">
                                <label for="product-name">Назва</label>
                                <input type="text" id="product-name" name="name"
required>
                            </div>

                            <div class="form-group">
                                <label for="product-description">Опис</label>
                                <textarea id="product-description" name="description"
rows="4"></textarea>
                            </div>

                            <div class="form-group">
                                <label for="product-price">Ціна</label>
                                <input type="number" id="product-price" name="price"
step="0.01" min="0" required>
                            </div>

                            <div class="form-group">
                                <label for="product-image">URL зображення</label>
                                <input type="text" id="product-image" name="image">
                            </div>

                            <div class="form-group">
                                <label for="product-category">Категорія</label>
                                <select id="product-category" name="category_id">
                                    <option value="">Виберіть категорію</option>
                                </select>
                            </div>

                            <div class="form-check">
                                <input type="checkbox" id="product-in-stock"
name="in_stock" checked>
                                <label for="product-in-stock">В наявності</label>
                            </div>

                            <div class="form-actions">
                                <button type="button" class="btn btn-secondary"
id="cancelProductBtn">Скасувати</button>
                                <button type="submit" class="btn btn-primary"
id="saveProductBtn">Створити</button>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    `;
        mainContent.innerHTML = html;
         // Завантажуємо категорії для форми
         try {
             const categoriesResponse = await fetch(`${API_URL}/categories`);
             const categories = await categoriesResponse.json();
             const categorySelect = document.getElementById('product-category');
```

```
                categories.forEach(category => {
                    const option = document.createElement('option');
                    option.value = category.id;
                    option.textContent = category.name;
                    categorySelect.appendChild(option);
                });
            } catch (error) {
                console.error('Помилка при завантаженні категорій:', error);
            }
            // Завантажуємо список товарів
            loadProductsList();
            // Додаємо обробники подій для модального вікна
            const modal = document.getElementById('productModal');
            const addProductBtn = document.getElementById('addProductBtn');
            const closeProductModal = document.getElementById('closeProductModal');
            const cancelProductBtn = document.getElementById('cancelProductBtn');
            const productForm = document.getElementById('productForm');
            addProductBtn.addEventListener('click', () => {
                resetProductForm();
                document.getElementById('productModalTitle').textContent = 'Додати
товар';
                document.getElementById('saveProductBtn').textContent = 'Створити';
                modal.style.display = 'flex';
            });
            closeProductModal.addEventListener('click', () => {
                modal.style.display = 'none';
            });
            cancelProductBtn.addEventListener('click', () => {
                modal.style.display = 'none';
            });
            // Закриття модального вікна при кліку поза його межами
            window.addEventListener('click', (e) => {
                if (e.target === modal) {
                    modal.style.display = 'none';
                }
            });
            // Обробник відправки форми товару
            productForm.addEventListener('submit', async (e) => {
                e.preventDefault();
                const productId = document.getElementById('product-id').value;
                const name = document.getElementById('product-name').value;
                const description = document.getElementById('product-
description').value;
                const price = parseFloat(document.getElementById('product-
price').value);
                const image = document.getElementById('product-image').value;
                const category_id = document.getElementById('product-category').value;
                const in_stock = document.getElementById('product-in-stock').checked;
                const saveBtn = document.getElementById('saveProductBtn');
                saveBtn.disabled = true;
                saveBtn.textContent = 'Збереження...';
                try {
                    const method = productId ? 'PUT' : 'POST';
                    const url = productId ? `${API_URL}/products/${productId}` :
`${API_URL}/products`;
                    const response = await fetch(url, {
                        method,
                        headers: {
                            'Content-Type': 'application/json',
                            'Authorization': `Bearer ${token}`
                        },
                        body: JSON.stringify({
                            name,
                            description,
                            price,
```

```
                        image,
                        category_id,
                        in_stock
                    })
                });
                if (!response.ok) {
                    const data = await response.json();
                    throw new Error(data.message || 'Помилка при збереженні
товару');
                }
                // Закриваємо модальне вікно і оновлюємо список товарів
                modal.style.display = 'none';
                loadProductsList();
            } catch (error) {
                console.error('Помилка при збереженні товару:', error);
                alert(error.message);
            } finally {
                saveBtn.disabled = false;
                saveBtn.textContent = productId ? 'Оновити' : 'Створити';
            }
        });
    }
    // Функція для завантаження списку товарів (для адмін-панелі)
    async function loadProductsList() {
        const productsTableContainer =
document.getElementById('productsTableContainer');
        try {
            const response = await fetch(`${API_URL}/products`, {
                headers: {
                    'Authorization': `Bearer ${token}`
                }
            });
            if (!response.ok) {
                throw new Error('Помилка при завантаженні товарів');
            }
            const products = await response.json();
            if (products.length === 0) {
                productsTableContainer.innerHTML = '<div class="no-items">Товари
не знайдено</div>';
                return;
            }

            const tableHtml = `
        <div class="admin-table-container">
            <table class="admin-table">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Зображення</th>
                        <th>Назва</th>
                        <th>Категорія</th>
                        <th>Ціна</th>
                        <th>Наявність</th>
                        <th>Дії</th>
                    </tr>
                </thead>
                <tbody>
                    ${products.map(product => `
                        <tr>
                            <td>${product.id}</td>
                            <td>
                                <img src="${product.image ||
'https://via.placeholder.com/50'}" alt="${product.name}" class="table-image">
                            </td>
                            <td>${product.name}</td>
                            <td>${product.category_name || 'Без категорії'}</td>
```

```
                                <td>${product.price.toFixed(2)} грн</td>
                                <td>
                                    <span class="status ${product.in_stock ? 'status-
active' : 'status-inactive'}">
                                        ${product.in_stock ? 'В наявності' : 'Немає в
наявності'}
                                    </span>
                                </td>
                                <td>
                                    <div class="table-actions">
                                        <button class="btn btn-sm btn-edit" data-
product-id="${product.id}">Редагувати</button>
                                        <button class="btn btn-sm btn-delete" data-
product-id="${product.id}">Видалити</button>
                                    </div>
                                </td>
                            </tr>
                        `).join('')}
                    </tbody>
                </table>
            </div>
        `;
                productsTableContainer.innerHTML = tableHtml;
                // Додаємо обробники подій для кнопок редагування і видалення
                document.querySelectorAll('.btn-edit').forEach(button => {
                    button.addEventListener('click', async (e) => {
                        const productId = e.target.getAttribute('data-product-id');
                        await loadProductForEdit(productId);
                    });
                });
                document.querySelectorAll('.btn-delete').forEach(button => {
                    button.addEventListener('click', async (e) => {
                        const productId = e.target.getAttribute('data-product-id');
                        await deleteProduct(productId);
                    });
                });
            } catch (error) {
                console.error('Помилка при завантаженні товарів:', error);
                productsTableContainer.innerHTML = `
            <div class="error">Помилка при завантаженні товарів:
${error.message}</div>
        `;
            }
        }
        // Функція для завантаження товару для редагування
        async function loadProductForEdit(productId) {
            try {
                const response = await fetch(`${API_URL}/products/${productId}`, {
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    throw new Error('Помилка при завантаженні товару');
                }
                const product = await response.json();
                document.getElementById('product-id').value = product.id;
                document.getElementById('product-name').value = product.name;
                document.getElementById('product-description').value =
product.description || '';
                document.getElementById('product-price').value = product.price;
                document.getElementById('product-image').value = product.image || '';
                document.getElementById('product-category').value =
product.category_id || '';
```

```
                document.getElementById('product-in-stock').checked =
product.in_stock;
                document.getElementById('productModalTitle').textContent = 'Редагувати
товар';
                document.getElementById('saveProductBtn').textContent = 'Оновити';
                document.getElementById('productModal').style.display = 'flex';
            } catch (error) {
                console.error('Помилка при завантаженні товару для редагування:',
error);
                alert(error.message);
            }
        }
        // Функція для видалення товару
        async function deleteProduct(productId) {
            if (!confirm('Ви впевнені, що хочете видалити цей товар?')) {
                return;
            }
            try {
                const response = await fetch(`${API_URL}/products/${productId}`, {
                    method: 'DELETE',
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    const data = await response.json();
                    throw new Error(data.message || 'Помилка при видаленні товару');
                }
                // Оновлюємо список товарів
                loadProductsList();
            } catch (error) {
                console.error('Помилка при видаленні товару:', error);
                alert(error.message);
            }
        }

        // Функція для скидання форми товару
        function resetProductForm() {
            document.getElementById('product-id').value = '';
            document.getElementById('product-name').value = '';
            document.getElementById('product-description').value = '';
            document.getElementById('product-price').value = '';
            document.getElementById('product-image').value = '';
            document.getElementById('product-category').value = '';
            document.getElementById('product-in-stock').checked = true;
        }
        // Функція для завантаження сторінки адміністрування категорій
        async function loadAdminCategoriesPage() {
            if (!currentUser || currentUser.role !== 'manager') {
                loadPage('home');
                return;
            }
            const html = `
        <div class="admin-categories-page">
            <div class="container">
                <div class="admin-header">
                    <h1 class="page-title">Управління категоріями</h1>
                    <button class="btn btn-primary" id="addCategoryBtn">Додати
категорію</button>
                </div>

                <div id="categoriesTableContainer">
                    <div class="loading">Завантаження категорій...</div>
                </div>

                <div id="categoryModal" class="modal" style="display: none;">
```

```
                    <div class="modal-content">
                        <div class="modal-header">
                            <h2 id="categoryModalTitle">Додати категорію</h2>
                            <button class="close-btn"
id="closeCategoryModal">&times;</button>
                        </div>

                        <form class="category-form" id="categoryForm">
                            <input type="hidden" id="category-id">

                            <div class="form-group">
                                <label for="category-name">Назва</label>
                                <input type="text" id="category-name" name="name"
required>
                            </div>

                            <div class="form-group">
                                <label for="category-description">Опис</label>
                                <textarea id="category-description" name="description"
rows="4"></textarea>
                            </div>

                            <div class="form-group">
                                <label for="category-image">URL зображення</label>
                                <input type="text" id="category-image" name="image">
                            </div>

                            <div class="form-actions">
                                <button type="button" class="btn btn-secondary"
id="cancelCategoryBtn">Скасувати</button>
                                <button type="submit" class="btn btn-primary"
id="saveCategoryBtn">Створити</button>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    `;
            mainContent.innerHTML = html;
            // Завантажуємо список категорій
            loadCategoriesList();
            // Додаємо обробники подій для модального вікна категорій
            const categoryModal = document.getElementById('categoryModal');
            const addCategoryBtn = document.getElementById('addCategoryBtn');
            const closeCategoryModal = document.getElementById('closeCategoryModal');
            const cancelCategoryBtn = document.getElementById('cancelCategoryBtn');
            const categoryForm = document.getElementById('categoryForm');
            addCategoryBtn.addEventListener('click', () => {
                resetCategoryForm();
                document.getElementById('categoryModalTitle').textContent = 'Додати
категорію';
                document.getElementById('saveCategoryBtn').textContent = 'Створити';
                categoryModal.style.display = 'flex';
            });
            closeCategoryModal.addEventListener('click', () => {
                categoryModal.style.display = 'none';
            });
            cancelCategoryBtn.addEventListener('click', () => {
                categoryModal.style.display = 'none';
            });
            // Закриття модального вікна при кліку поза його межами
            window.addEventListener('click', (e) => {
                if (e.target === categoryModal) {
                    categoryModal.style.display = 'none';
```

```
            }
        });
        // Обробник відправки форми категорії
        categoryForm.addEventListener('submit', async (e) => {
            e.preventDefault();
            const categoryId = document.getElementById('category-id').value;
            const name = document.getElementById('category-name').value;
            const description = document.getElementById('category-
description').value;
            const image = document.getElementById('category-image').value;
            const saveBtn = document.getElementById('saveCategoryBtn');
            saveBtn.disabled = true;
            saveBtn.textContent = 'Збереження...';
            try {
                const method = categoryId ? 'PUT' : 'POST';
                const url = categoryId ? `${API_URL}/categories/${categoryId}` :
`${API_URL}/categories`;
                const response = await fetch(url, {
                    method,
                    headers: {
                        'Content-Type': 'application/json',
                        'Authorization': `Bearer ${token}`
                    },
                    body: JSON.stringify({
                        name,
                        description,
                        image
                    })
                });
                if (!response.ok) {
                    const data = await response.json();
                    throw new Error(data.message || 'Помилка при збереженні
категорії');
                }
                // Закриваємо модальне вікно і оновлюємо список категорій
                categoryModal.style.display = 'none';
                loadCategoriesList();
            } catch (error) {
                console.error('Помилка при збереженні категорії:', error);
                alert(error.message);
            } finally {
                saveBtn.disabled = false;
                saveBtn.textContent = categoryId ? 'Оновити' : 'Створити';
            }
        });
    }
    // Функція для завантаження списку категорій (для адмін-панелі)
    async function loadCategoriesList() {
        const categoriesTableContainer =
document.getElementById('categoriesTableContainer');
        try {
            const response = await fetch(`${API_URL}/categories`);
            if (!response.ok) {
                throw new Error('Помилка при завантаженні категорій');
            }
            const categories = await response.json();
            if (categories.length === 0) {
                categoriesTableContainer.innerHTML = '<div class="no-
items">Категорії не знайдено</div>';
                return;
            }
            const tableHtml = `

        <div class="admin-table-container">
            <table class="admin-table">
                <thead>
```

```
                        <tr>
                            <th>ID</th>
                            <th>Зображення</th>
                            <th>Назва</th>
                            <th>Опис</th>
                            <th>Дії</th>
                        </tr>
                    </thead>
                    <tbody>
                        ${categories.map(category => `
                            <tr>
                                <td>${category.id}</td>
                                <td>
                                    <img src="${category.image || 
'https://via.placeholder.com/50'}" alt="${category.name}" class="table-image">
                                </td>
                                <td>${category.name}</td>
                                <td>${category.description || 'Без опису'}</td>
                                <td>
                                    <div class="table-actions">
                                        <button class="btn btn-sm btn-edit" data-
category-id="${category.id}">Редагувати</button>
                                        <button class="btn btn-sm btn-delete" data-
category-id="${category.id}">Видалити</button>
                                    </div>
                                </td>
                            </tr>
                        `).join('')}
                    </tbody>
                </table>
            </div>
        `;
                categoriesTableContainer.innerHTML = tableHtml;
                // Додаємо обробники подій для кнопок редагування і видалення
                document.querySelectorAll('.btn-edit').forEach(button => {
                    button.addEventListener('click', async (e) => {
                        const categoryId = e.target.getAttribute('data-category-id');
                        await loadCategoryForEdit(categoryId);
                    });
                });
                document.querySelectorAll('.btn-delete').forEach(button => {
                    button.addEventListener('click', async (e) => {
                        const categoryId = e.target.getAttribute('data-category-id');
                        await deleteCategory(categoryId);
                    });
                });
            } catch (error) {
                console.error('Помилка при завантаженні категорій:', error);
                categoriesTableContainer.innerHTML = `
            <div class="error">Помилка при завантаженні категорій: 
${error.message}</div>
        `;
            }
        }
        // Функція для завантаження категорії для редагування
        async function loadCategoryForEdit(categoryId) {
            try {
                console.log('Завантаження категорії для редагування, ID:', 
categoryId);
                // Отримуємо всі категорії та знаходимо потрібну за ID
                const response = await fetch(`${API_URL}/categories`);
                if (!response.ok) {
                    throw new Error('Помилка при завантаженні категорій');
                }
                const categories = await response.json();
```

```
                const category = categories.find(cat => cat.id == categoryId);
                if (!category) {
                    throw new Error('Категорію не знайдено');
                }
                console.log('Знайдена категорія для редагування:', category);
                // Заповнюємо форму даними категорії
                document.getElementById('category-id').value = category.id;
                document.getElementById('category-name').value = category.name;
                document.getElementById('category-description').value =
category.description || '';
                document.getElementById('category-image').value = category.image ||
'';
                document.getElementById('categoryModalTitle').textContent =
'Редагувати категорію';
                document.getElementById('saveCategoryBtn').textContent = 'Оновити';
                document.getElementById('categoryModal').style.display = 'flex';
            } catch (error) {
                console.error('Помилка при завантаженні категорії для редагування:',
error);
                alert(error.message);
            }
        }
        // Функція для видалення категорії
        async function deleteCategory(categoryId) {
            if (!confirm('Ви впевнені, що хочете видалити цю категорію? Усі пов\'язані
товари втратять зв\'язок з категорією.')) {
                return;
            }
            try {
                const response = await fetch(`${API_URL}/categories/${categoryId}`, {
                    method: 'DELETE',
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    const data = await response.json();
                    throw new Error(data.message || 'Помилка при видаленні
категорії');
                }
                // Оновлюємо список категорій
                loadCategoriesList();
            } catch (error) {
                console.error('Помилка при видаленні категорії:', error);
                alert(error.message);
            }
        }
        // Функція для скидання форми категорії
        function resetCategoryForm() {
            document.getElementById('category-id').value = '';
            document.getElementById('category-name').value = '';
            document.getElementById('category-description').value = '';
            document.getElementById('category-image').value = '';
        }
        // Функція для завантаження сторінки адміністрування користувачів
        async function loadAdminUsersPage() {
            if (!currentUser || currentUser.role !== 'manager') {
                loadPage('home');
                return;
            }
            const html = `
        <div class="admin-users-page">
            <div class="container">
                <h1 class="page-title">Управління користувачами</h1>
```

```
                <div id="usersTableContainer">
                    <div class="loading">Завантаження користувачів...</div>
                </div>
            </div>
        </div>
    `;
            mainContent.innerHTML = html;
            try {
                const response = await fetch(`${API_URL}/users`, {
                    headers: {
                        'Authorization': `Bearer ${token}`
                    }
                });
                if (!response.ok) {
                    throw new Error('Помилка при завантаженні користувачів');
                }
                const users = await response.json();
                const usersTableContainer =
document.getElementById('usersTableContainer');
                if (users.length === 0) {
                    usersTableContainer.innerHTML = '<div class="no-
items">Користувачів не знайдено</div>';
                    return;
                }
                const tableHtml = `
            <div class="admin-table-container">
                <table class="admin-table">
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Ім'я</th>
                            <th>Email</th>
                            <th>Телефон</th>
                            <th>Адреса</th>
                            <th>Дата реєстрації</th>
                            <th>Дії</th>
                        </tr>
                    </thead>
                    <tbody>
                        ${users.map(user => `
                            <tr>
                                <td>${user.id}</td>
                                <td>${user.name}</td>
                                <td>${user.email}</td>
                                <td>${user.phone || 'Не вказано'}</td>
                                <td>${user.address || 'Не вказано'}</td>
                                <td>${new Date(user.created_at).toLocaleString('uk-
UA')}</td>
                                <td>
                                    <div class="table-actions">
                                        <button class="btn btn-sm btn-delete" data-
user-id="${user.id}">Видалити</button>
                                    </div>
                                </td>
                            </tr>
                        `).join('')}
                    </tbody>
                </table>
            </div>
        `;
                usersTableContainer.innerHTML = tableHtml;
                // Додаємо обробники подій для кнопок видалення
                document.querySelectorAll('.btn-delete').forEach(button => {
                    button.addEventListener('click', async (e) => {
                        const userId = e.target.getAttribute('data-user-id');
```

```
                        if (confirm('Ви впевнені, що хочете видалити цього
користувача? Усі пов\'язані замовлення будуть збережені, але стануть анонімними.')) {
                            try {
                                const response = await
fetch(`${API_URL}/users/${userId}`, {
                                    method: 'DELETE',
                                    headers: {
                                        'Authorization': `Bearer ${token}`
                                    }
                                });
                                if (!response.ok) {
                                    const data = await response.json();
                                    throw new Error(data.message || 'Помилка при
видаленні користувача');
                                }
                                // Оновлюємо список користувачів
                                loadAdminUsersPage();
                            } catch (error) {
                                console.error('Помилка при видаленні користувача:',
error);
                                alert(error.message);
                            }
                        }
                    });
                });
            } catch (error) {
                console.error('Помилка при завантаженні користувачів:', error);
                document.getElementById('usersTableContainer').innerHTML = `
            <div class="error">Помилка при завантаженні користувачів:
${error.message}</div>
        `;
            }
        }
        function handleParallax() {
            // Додаємо обробник прокрутки для всіх сторінок
            window.addEventListener('scroll', function () {
                const scrollPosition = window.pageYOffset;
                // Паралакс для фону заголовка
                const pageTitles = document.querySelectorAll('.page-title');
                pageTitles.forEach(title => {
                    title.style.backgroundPosition = `center ${50 + scrollPosition *
0.05}%`;
                });
                // Паралакс для героя на головній сторінці
                const hero = document.querySelector('.hero');
                if (hero) {
                    hero.style.backgroundPosition = `center ${50 + scrollPosition *
0.1}%`;
                }
                // Ефект для карток категорій та товарів
                const cards = document.querySelectorAll('.category-card, .product-
card');
                cards.forEach((card, index) => {
                    const offset = (scrollPosition - card.offsetTop +
window.innerHeight) * 0.02;
                    const direction = index % 2 === 0 ? 1 : -1;
                    // Застосовуємо легкий зсув для створення "живого" ефекту
                    if (offset > 0 && offset < 15) {
                        card.style.transform = `translateY(${-offset * 0.5}px)
translateX(${offset * 0.2 * direction}px)`;
                    }
                });
                // Ефект паралаксу для контейнерів
                const containers = document.querySelectorAll('.container');
                containers.forEach((container, index) => {
```

```
                    if (container.closest('.hero')) return; // Пропускаємо контейнери
в героїї
                    const containerTop = container.getBoundingClientRect().top;
                    if (containerTop < window.innerHeight && containerTop > -
container.offsetHeight) {
                        const depth = index % 3 === 0 ? 0.05 : (index % 3 === 1 ? 0.03
: 0.01);
                        container.style.transform = `translateY(${scrollPosition *
depth}px)`;
                    }
                });
            });
            // Ініціалізуємо паралакс для поточного положення прокрутки
            window.dispatchEvent(new Event('scroll'));
        }
        // Функція для запуску анімацій при завантаженні сторінки
        function initPageAnimations() {
            // Анімація з'явлення контенту
            const mainContent = document.getElementById('mainContent');
            if (mainContent) {
                mainContent.style.opacity = '0';
                mainContent.style.transform = 'translateY(20px)';
                mainContent.style.transition = 'opacity 0.5s ease, transform 0.5s
ease';
                setTimeout(() => {
                    mainContent.style.opacity = '1';
                    mainContent.style.transform = 'translateY(0)';
                }, 100);
            }
            // Анімація для заголовків
            const pageTitles = document.querySelectorAll('.page-title');
            pageTitles.forEach(title => {
                title.style.opacity = '0';
                title.style.transform = 'translateY(-20px)';
                title.style.transition = 'opacity 0.7s ease, transform 0.7s ease';
                setTimeout(() => {
                    title.style.opacity = '1';
                    title.style.transform = 'translateY(0)';
                }, 200);
            });
            // Анімація для карток
                    const cards = document.querySelectorAll('.category-card,
.product-card');
            cards.forEach((card, index) => {
                card.style.opacity = '0';
                card.style.transform = 'translateY(30px)';
                card.style.transition = 'opacity 0.5s ease, transform 0.5s ease';
                setTimeout(() => {
                    card.style.opacity = '1';
                    card.style.transform = 'translateY(0)';
                }, 300 + index * 50); // Поступова поява карток
            });
        }
        // Модифікуємо функцію loadPage, щоб додати ефекти
        const originalLoadPage = loadPage;
        loadPage = function (page) {
            // Ефект зникнення перед зміною контенту
            const mainContent = document.getElementById('mainContent');
            if (mainContent) {
                mainContent.style.opacity = '0';
                mainContent.style.transform = 'translateY(20px)';
                setTimeout(() => {
                    // Викликаємо оригінальну функцію після анімації зникнення
                    originalLoadPage(page);
                    // Запускаємо ефекти появи для нового контенту
```

```
                    setTimeout(() => {
                        initPageAnimations();
                        handleParallax();
                    }, 100);
                }, 300);
            } else {
                originalLoadPage(page);
                setTimeout(() => {
                    initPageAnimations();
                    handleParallax();
                }, 100);
            }
        };
        // Ініціалізуємо паралакс та анімації при завантаженні сторінки
        document.addEventListener('DOMContentLoaded', () => {
            initPageAnimations();
            handleParallax();
            // Додаємо обробник для адаптивності при зміні розміру вікна
            window.addEventListener('resize', handleParallax);
        });
    </script>
</body>
</html>
```

styles.css

```
/* Основні змінні кольорів з палітри */
:root {
    --color-light: #ffffff;
    --color-cream: #e6d9c0;
    --color-gumidnt: #9a6d38;
    --color-siller: #c1c1c1;
    --color-coconite: #a5a5a5;
    --color-dark: #303030;
    --color-builck: #101010;
    --color-gold: #d4af37;
    --color-silver: #c0c0c0;
    --color-gray: #808080;
    --font-main: 'Roboto', sans-serif;
    --font-heading: 'Playfair Display', serif;
    --transition: all 0.3s ease;
}
/* Базові стилі */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
body {
    font-family: var(--font-main);
    font-size: 16px;
    line-height: 1.6;
    color: var(--color-dark);
    background-color: var(--color-light);
}
a {
    text-decoration: none;
    color: var(--color-gumidnt);
    transition: var(--transition);
}
    a:hover {
```

```
        color: var(--color-gold);
    }
ul {
    list-style: none;
}
img {
    max-width: 100%;
    height: auto;
}
.container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 15px;
}
.btn {
    display: inline-block;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    font-size: 16px;
    cursor: pointer;
    transition: var(--transition);
    text-align: center;
}
.btn-primary {
    background-color: var(--color-gumidnt);
    color: var(--color-light);
}
    .btn-primary:hover {
        background-color: var(--color-gold);
        color: var(--color-dark);
    }
.btn-secondary {
    background-color: var(--color-cream);
    color: var(--color-dark);
}
    .btn-secondary:hover {
        background-color: var(--color-siller);
    }
.btn-outline {
    background-color: transparent;
    border: 1px solid var(--color-gumidnt);
    color: var(--color-gumidnt);
}
    .btn-outline:hover {
        background-color: var(--color-gumidnt);
        color: var(--color-light);
    }
.btn-danger {
    background-color: #e74c3c;
    color: var(--color-light);
}
    .btn-danger:hover {
        background-color: #c0392b;
    }
.btn-sm {
    padding: 5px 10px;
    font-size: 14px;
}
.btn-link {
    background: none;
    border: none;
    padding: 0;
    font: inherit;
```

```
    color: var(--color-gumidnt);
    cursor: pointer;
    text-decoration: none;
}
    .btn-link:hover {
        color: var(--color-gold);
        text-decoration: underline;
    }
.btn-edit {
    background-color: var(--color-siller);
    color: var(--color-dark);
}
    .btn-edit:hover {
        background-color: var(--color-coconite);
    }
.btn-delete {
    background-color: #e74c3c;
    color: var(--color-light);
}
    .btn-delete:hover {
        background-color: #c0392b;
    }
.w-100 {
    width: 100%;
}
.text-center {
    text-align: center;
}
.mt-4 {
    margin-top: 1.5rem;
}
.loading {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 200px;
    font-size: 18px;
    color: var(--color-gumidnt);
}
.error {
    padding: 15px;
    margin-bottom: 20px;
    background-color: #fadbd8;
    color: #e74c3c;
    border-radius: 4px;
}
.alert {
    padding: 15px;
    margin-bottom: 20px;
    border-radius: 4px;
}
.alert-success {
    background-color: #d4edda;
    color: #155724;
}
.alert-danger {
    background-color: #f8d7da;
    color: #721c24;
}
/* Шапка сайту */
.header {
    background-color: var(--color-light);
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    position: sticky;
    top: 0;
```

```
    z-index: 1000;
}
.header-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 15px;
}
.logo a {
    font-family: var(--font-heading);
    font-size: 24px;
    font-weight: 700;
    color: var(--color-gumidnt);
}
.nav-list {
    display: flex;
    align-items: center;
}
.nav-item {
    margin-left: 20px;
    position: relative;
}
.dropdown-toggle {
    cursor: pointer;
    display: flex;
    align-items: center;
}
    .dropdown-toggle::after {
        content: '▼';
        font-size: 10px;
        margin-left: 5px;
    }
.dropdown-menu {
    position: absolute;
    top: 100%;
    left: 0;
    background-color: var(--color-light);
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    border-radius: 4px;
    padding: 10px 0;
    width: 150px;
    display: none;
    z-index: 1000;
}
.nav-item:hover .dropdown-menu {
    display: block;
}
.dropdown-menu li {
    padding: 5px 15px;
}
.dropdown-menu a {
    display: block;
    padding: 5px 0;
}
.cart-badge {
    display: inline-block;
    background-color: var(--color-gold);
    color: var(--color-dark);
    font-size: 12px;
    font-weight: bold;
    border-radius: 50%;
    padding: 3px 6px;
    margin-left: 5px;
}
.menu-toggle {
```

```
    display: none;
    background: none;
    border: none;
    cursor: pointer;
    padding: 0;
    width: 30px;
    height: 20px;
    position: relative;
}
    .menu-toggle span {
        display: block;
        width: 100%;
        height: 2px;
        background-color: var(--color-dark);
        position: absolute;
        left: 0;
        transition: var(--transition);
    }
        .menu-toggle span:nth-child(1) {
            top: 0;
        }
        .menu-toggle span:nth-child(2) {
            top: 50%;
            transform: translateY(-50%);
        }
        .menu-toggle span:nth-child(3) {
            bottom: 0;
        }
/* Підвал сайту */
.footer {
    background-color: var(--color-builck); /* Темніший, майже чорний фон */
    color: var(--color-light);
    padding: 50px 0 20px;
    margin-top: 60px;
    position: relative;
    border-top: 5px solid var(--color-gumidnt); /* Декоративна верхня межа */
}
    .footer::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background-image: url('https://images.unsplash.com/photo-1549645402-
a9d655c5c344?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1920&q=70');
        background-size: cover;
        background-position: center;
        opacity: 0.1; /* Ледь видимий фоновий малюнок для текстури */
        z-index: 0;
    }
    .footer .container {
        background-color: transparent; /* Прибираємо фон контейнера для футера */
        box-shadow: none;
        position: relative;
        z-index: 1;
    }
.footer-content {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-bottom: 30px;
    gap: 20px;
```

```
}
.footer-section {
    flex: 1;
    min-width: 250px;
    margin-bottom: 30px;
    padding: 0 15px;
}
    .footer-section h3 {
        color: var(--color-gold); /* Золотистий колір для заголовків */
        margin-bottom: 20px;
        font-family: var(--font-heading);
        font-size: 22px;
        position: relative;
        padding-bottom: 10px;
    }
        .footer-section h3::after {
            content: "";
            position: absolute;
            left: 0;
            bottom: 0;
            width: 50px;
            height: 3px;
            background-color: var(--color-gumidnt); /* Декоративна лінія під
заголовком */
        }
    .footer-section p {
        margin-bottom: 15px;
        line-height: 1.8;
        color: var(--color-siller); /* Світло-сірий для тексту */
    }
.social-links {
    display: flex;
    flex-wrap: wrap;
    gap: 15px;
}
.social-link {
    color: var(--color-siller);
    text-decoration: none;
    transition: color 0.3s ease, transform 0.3s ease;
    padding: 5px 10px;
    border-radius: 5px;
    background-color: rgba(255, 255, 255, 0.05);
}
    .social-link:hover {
        color: var(--color-gold);
        transform: translateY(-3px);
        background-color: rgba(255, 255, 255, 0.1);
    }
.footer-bottom {
    text-align: center;
    padding-top: 20px;
    border-top: 1px solid rgba(255, 255, 255, 0.1);
    color: var(--color-coconite); /* Сірий колір для копірайту */
    font-size: 14px;
}
/* Адаптивність для футера */
@media (max-width: 768px) {
    .footer-content {
        flex-direction: column;
    }
    .footer-section {
        flex: 0 0 100%;
        text-align: center;
    }
        .footer-section h3::after {
```

```
            left: 50%;
            transform: translateX(-50%);
        }
    .social-links {
        justify-content: center;
    }
}
/* Секції на головній сторінці */
.section {
    padding: 60px 0;
}
.section-title {
    font-family: var(--font-heading);
    text-align: center;
    margin-bottom: 30px;
    color: var(--color-gumidnt);
    font-size: 32px;
}
.hero {
    background-color: var(--color-cream);
    padding: 80px 0;
    background-image: url('https://dengodel.com/wp-
content/uploads/2021/02/assortiment-cvetochnogo-magazina.jpg');
    background-size: cover;
    background-position: center;
    position: relative;
}
    .hero::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background-color: rgba(0, 0, 0, 0.4);
    }
.hero-content {
    position: relative;
    max-width: 600px;
    margin: 0 auto;
    text-align: center;
    color: var(--color-light);
}
.hero h1 {
    font-family: var(--font-heading);
    font-size: 42px;
    margin-bottom: 15px;
}
.hero p {
    font-size: 18px;
    margin-bottom: 30px;
}
.categories-grid, .products-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
    gap: 30px;
}
.category-card, .product-card {
    background-color: var(--color-light);
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1);
    transition: var(--transition);
    height: 100%;
}
```

```
    .category-card:hover, .product-card:hover {
        transform: translateY(-5px);
        box-shadow: 0 5px 20px rgba(0, 0, 0, 0.15);
    }
.category-image, .product-image {
    height: 200px;
    overflow: hidden;
}
    .category-image img, .product-image img {
        width: 100%;
        height: 100%;
        object-fit: cover;
        transition: var(--transition);
    }
.category-card:hover .category-image img,
.product-card:hover .product-image img {
    transform: scale(1.05);
}
.category-name {
    padding: 15px;
    text-align: center;
    font-family: var(--font-heading);
    color: var(--color-dark);
}
.product-info {
    padding: 15px;
}
.product-name {
    font-family: var(--font-heading);
    color: var(--color-dark);
    margin-bottom: 5px;
}
.product-price {
    color: var(--color-gumidnt);
    font-weight: bold;
    font-size: 18px;
}
.product-category {
    font-size: 14px;
    color: var(--color-gray);
    margin-top: 5px;
}
/* Сторінка авторизації */
.auth-page {
    padding: 60px 0;
}
.auth-form-container {
    max-width: 450px;
    margin: 0 auto;
    background-color: var(--color-light);
    border-radius: 8px;
    padding: 30px;
    box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1);
}
    .auth-form-container h2 {
        font-family: var(--font-heading);
        text-align: center;
        margin-bottom: 30px;
        color: var(--color-gumidnt);
    }
.auth-form .form-group {
    margin-bottom: 20px;
}
.auth-form label {
    display: block;
```

```
    margin-bottom: 5px;
    font-weight: 500;
}
.auth-form input,
.auth-form select,
.auth-form textarea {
    width: 100%;
    padding: 10px 15px;
    border: 1px solid #ddd;
    border-radius: 4px;
    transition: var(--transition);
}
    .auth-form input:focus,
    .auth-form select:focus,
    .auth-form textarea:focus {
        border-color: var(--color-gumidnt);
        outline: none;
    }
.auth-links {
    margin-top: 20px;
    text-align: center;
}
/* Сторінка каталогу */
.catalog-page {
    padding: 40px 0;
}
.page-title {
    font-family: var(--font-heading);
    text-align: center;
    margin-bottom: 30px;
    color: var(--color-gumidnt);
}
.filters-panel {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-bottom: 30px;
    padding: 15px;
    background-color: var(--color-cream);
    border-radius: 8px;
}
.search-form {
    display: flex;
    flex: 1;
    max-width: 500px;
    margin-right: 15px;
}
    .search-form input {
        flex: 1;
        padding: 10px 15px;
        border: 1px solid #ddd;
        border-radius: 4px 0 0 4px;
        border-right: none;
    }
    .search-form button {
        border-radius: 0 4px 4px 0;
    }
.filter-controls {
    display: flex;
    flex-wrap: wrap;
}
.filter-item {
    margin-left: 15px;
    min-width: 200px;
}
```

```
    .filter-item label {
        display: block;
        margin-bottom: 5px;
        font-weight: 500;
    }
    .filter-item select {
        width: 100%;
        padding: 10px 15px;
        border: 1px solid #ddd;
        border-radius: 4px;
        background-color: var(--color-light);
    }
.no-results {
    text-align: center;
    padding: 40px 0;
    font-size: 18px;
    color: var(--color-gray);
}
/* Сторінка товару */
.product-detail-page {
    padding: 40px 0;
}
.product-detail {
    display: flex;
    flex-wrap: wrap;
    gap: 30px;
}
    .product-detail .product-image {
        flex: 1;
        min-width: 300px;
        height: 400px;
        border-radius: 8px;
        overflow: hidden;
    }
    .product-detail .product-info {
        flex: 1;
        min-width: 300px;
    }
    .product-detail .product-name {
        font-family: var(--font-heading);
        font-size: 28px;
        margin-bottom: 10px;
    }
    .product-detail .product-price {
        font-size: 24px;
        color: var(--color-gumidnt);
        margin-bottom: 20px;
    }
.product-description {
    margin: 20px 0;
}
        .product-description h3 {
        font-family: var(--font-heading);
        margin-bottom: 10px;
    }
.product-actions {
    margin-top: 30px;
}
.quantity-control {
    display: flex;
    align-items: center;
    margin-bottom: 20px;
}
    .quantity-control label {
        margin-right: 10px;
```

```
    }
    .quantity-control input {
        width: 60px;
        padding: 8px;
        border: 1px solid #ddd;
        border-radius: 4px;
        text-align: center;
    }
.btn-add-to-cart {
    margin-top: 10px;
}
.out-of-stock {
    padding: 15px;
    background-color: #f8d7da;
    color: #721c24;
    border-radius: 4px;
    margin-top: 20px;
}
/* Кошик */
.cart-page {
    padding: 40px 0;
}
.empty-cart {
    text-align: center;
    padding: 40px 0;
}
    .empty-cart p {
        font-size: 18px;
        margin-bottom: 20px;
        color: var(--color-gray);
    }
.cart-items {
    margin-bottom: 30px;
}
.cart-item {
    display: flex;
    align-items: center;
    padding: 15px;
    border-bottom: 1px solid #ddd;
    margin-bottom: 15px;
}
    .cart-item .item-image {
        width: 80px;
        height: 80px;
        margin-right: 15px;
        border-radius: 4px;
        overflow: hidden;
    }
        .cart-item .item-image img {
            width: 100%;
            height: 100%;
            object-fit: cover;
        }
    .cart-item .item-details {
        flex: 1;
    }
    .cart-item .item-name {
        font-family: var(--font-heading);
        margin-bottom: 5px;
    }
    .cart-item .item-price {
        color: var(--color-gumidnt);
    }
    .cart-item .item-quantity {
        display: flex;
```

```
        align-items: center;
        margin: 0 20px;
    }
        .cart-item .item-quantity span {
            margin: 0 10px;
            min-width: 30px;
            text-align: center;
        }
    .cart-item .item-total {
        font-weight: bold;
        margin-right: 20px;
    }
.cart-summary {
    background-color: var(--color-cream);
    padding: 20px;
    border-radius: 8px;
}
.cart-total {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
    font-size: 18px;
}
    .cart-total p {
        font-weight: bold;
        color: var(--color-gumidnt);
        font-size: 24px;
    }
.cart-actions {
    display: flex;
    justify-content: space-between;
}
/* Профіль */
.profile-page {
    padding: 40px 0;
}
.profile-tabs {
    display: flex;
    margin-bottom: 30px;
    border-bottom: 1px solid #ddd;
}
.profile-tab {
    padding: 15px 30px;
    cursor: pointer;
    transition: var(--transition);
    border-bottom: 2px solid transparent;
}
    .profile-tab.active {
        color: var(--color-gumidnt);
        border-bottom-color: var(--color-gumidnt);
    }
    .profile-tab:hover {
        color: var(--color-gold);
    }
.profile-form-container {
    max-width: 600px;
    margin: 0 auto;
}
.profile-form .form-group {
    margin-bottom: 20px;
}
.profile-form label {
    display: block;
    margin-bottom: 5px;
```

```
    font-weight: 500;
}
.profile-form input,
.profile-form select,
.profile-form textarea {
    width: 100%;
    padding: 10px 15px;
    border: 1px solid #ddd;
    border-radius: 4px;
    transition: var(--transition);
}
    .profile-form input:focus,
    .profile-form select:focus,
    .profile-form textarea:focus {
        border-color: var(--color-gumidnt);
        outline: none;
    }
    .profile-form input:disabled {
        background-color: #f9f9f9;
        cursor: not-allowed;
    }
.form-text {
    font-size: 12px;
    margin-top: 5px;
}
.form-check {
    display: flex;
    align-items: center;
    margin-bottom: 20px;
}
    .form-check input {
        width: auto;
        margin-right: 10px;
    }
.no-history {
    text-align: center;
    padding: 40px 0;
    color: var(--color-gray);
}
.view-date {
    font-size: 12px;
    color: var(--color-gray);
    margin-top: 5px;
}
/* Замовлення */
.orders-page {
    padding: 40px 0;
}
.no-orders {
    text-align: center;
    padding: 40px 0;
    color: var(--color-gray);
}
.orders-list {
    display: flex;
    flex-direction: column;
    gap: 20px;
}
.order-card {
    background-color: var(--color-light);
    border-radius: 8px;
    overflow: hidden;
    box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1);
    padding: 20px;
}
```

```
.order-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 15px;
    padding-bottom: 15px;
    border-bottom: 1px solid #ddd;
}
.order-status {
    padding: 5px 10px;
    border-radius: 4px;
    font-size: 14px;
    font-weight: 500;
}
.status-pending {
    background-color: #fef9e7;
    color: #d35400;
}
.status-paid {
    background-color: #d1f2eb;
    color: #16a085;
}
.status-shipped {
    background-color: #ebf5fb;
    color: #2980b9;
}
.status-delivered {
    background-color: #e9f7ef;
    color: #27ae60;
}
.status-cancelled {
    background-color: #f8d7da;
    color: #721c24;
}
.order-info {
    margin-bottom: 20px;
}
    .order-info p {
        margin-bottom: 10px;
    }
.order-status-control {
    margin-top: 15px;
    display: flex;
    align-items: center;
}
    .order-status-control label {
        margin-right: 10px;
    }
    .order-status-control select {
        padding: 8px;
        border: 1px solid #ddd;
        border-radius: 4px;
    }
.order-items h4 {
    margin-bottom: 15px;
    font-family: var(--font-heading);
}
.order-item {
    display: flex;
    align-items: center;
    padding: 10px;
    border-bottom: 1px solid #eee;
}
    .order-item:last-child {
        border-bottom: none;
```

```
    }
    .order-item .item-image {
        width: 50px;
        height: 50px;
        margin-right: 15px;
        border-radius: 4px;
        overflow: hidden;
    }
    .order-item .item-info {
        flex: 1;
    }
    .order-item .item-name {
        font-family: var(--font-heading);
        margin-bottom: 5px;
    }
    .order-item .item-total {
        font-weight: bold;
    }
/* Чат */
.chat-page {
    padding: 40px 0;
}
.chat-container {
    display: flex;
    height: 600px;
    border: 1px solid #ddd;
    border-radius: 8px;
    overflow: hidden;
}
.chat-sidebar {
    width: 300px;
    background-color: var(--color-cream);
    padding: 20px;
    overflow-y: auto;
    border-right: 1px solid #ddd;
}
    .chat-sidebar h3 {
        margin-bottom: 15px;
        padding-bottom: 10px;
        border-bottom: 1px solid #ddd;
        font-family: var(--font-heading);
    }
.no-contacts {
    text-align: center;
    padding: 20px 0;
    color: var(--color-gray);
}
.contacts-list {
    display: flex;
    flex-direction: column;
    gap: 10px;
}
.contact-item {
    padding: 15px;
    border-radius: 8px;
    cursor: pointer;
    transition: var(--transition);
}
    .contact-item:hover {
        background-color: rgba(0, 0, 0, 0.05);
    }
    .contact-item.active {
        background-color: rgba(0, 0, 0, 0.1);
    }
.contact-name {
```

```
    font-family: var(--font-heading);
    margin-bottom: 5px;
}
.contact-email {
    font-size: 14px;
    color: var(--color-gray);
}
.chat-main {
    flex: 1;
    display: flex;
    flex-direction: column;
}
.chat-header {
    padding: 15px;
    background-color: var(--color-cream);
    border-bottom: 1px solid #ddd;
}
    .chat-header h3 {
        font-family: var(--font-heading);
    }
.no-contact-selected,
.no-messages {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100%;
    color: var(--color-gray);
}
.chat-messages {
    flex: 1;
    padding: 20px;
    overflow-y: auto;
    display: flex;
    flex-direction: column;
    gap: 15px;
}
.message {
    max-width: 70%;
    padding: 10px 15px;
    border-radius: 10px;
    position: relative;
}
.own-message {
    align-self: flex-end;
    background-color: var(--color-gumidnt);
    color: var(--color-light);
    border-bottom-right-radius: 0;
}
.other-message {
    align-self: flex-start;
    background-color: var(--color-siller);
    color: var(--color-dark);
    border-bottom-left-radius: 0;
}
.message-info {
    display: flex;
    justify-content: flex-end;
    font-size: 12px;
    margin-top: 5px;
}
.message-time {
    opacity: 0.7;
}
.message-status {
    margin-left: 5px;
```

```
}
.chat-form {
    display: flex;
    padding: 15px;
    border-top: 1px solid #ddd;
}
    .chat-form input {
        flex: 1;
        padding: 10px 15px;
        border: 1px solid #ddd;
        border-radius: 4px;
        margin-right: 10px;
    }
/* Адмін панель */
.admin-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 30px;
}
.admin-table-container {
    overflow-x: auto;
}
.admin-table {
    width: 100%;
    border-collapse: collapse;
}
    .admin-table th,
    .admin-table td {
        padding: 15px;
        text-align: left;
        border-bottom: 1px solid #ddd;
    }
    .admin-table th {
        background-color: var(--color-cream);
        font-weight: 500;
    }
.table-image {
    width: 50px;
    height: 50px;
    object-fit: cover;
    border-radius: 4px;
}
.table-actions {
    display: flex;
    gap: 10px;
}
.status {
    display: inline-block;
    padding: 3px 8px;
    border-radius: 4px;
    font-size: 14px;
}
.status-active {
    background-color: #e9f7ef;
    color: #27ae60;
}
.status-inactive {
    background-color: #f8d7da;
    color: #721c24;
}
.modal {
    position: fixed;
    top: 0;
    left: 0;
```

```
    right: 0;
    bottom: 0;
    background-color: rgba(0, 0, 0, 0.5);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
}
.modal-content {
    background-color: var(--color-light);
    border-radius: 8px;
    padding: 20px;
    width: 100%;
    max-width: 600px;
    max-height: 90vh;
    overflow-y: auto;
}
.modal-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
    padding-bottom: 15px;
    border-bottom: 1px solid #ddd;
}
    .modal-header h2 {
        font-family: var(--font-heading);
    }
.close-btn {
    background: none;
    border: none;
    font-size: 24px;
    cursor: pointer;
    color: var(--color-gray);
}
.product-form, .category-form {
    display: flex;
    flex-direction: column;
    gap: 15px;
}
.form-actions {
    display: flex;
    justify-content: flex-end;
    gap: 10px;
    margin-top: 20px;
}
/* Not Found Page */
.not-found-page {
    text-align: center;
    padding: 100px 0;
}
    .not-found-page h1 {
        font-size: 120px;
        margin-bottom: 0;
        color: var(--color-gumidnt);
    }
    .not-found-page h2 {
        font-family: var(--font-heading);
        margin-bottom: 20px;
    }
    .not-found-page p {
        font-size: 18px;
        color: var(--color-gray);
        margin-bottom: 30px;
    }
```

```
/* Адаптивність */
@media (max-width: 992px) {
    .hero h1 {
        font-size: 32px;
    }
    .section-title {
        font-size: 28px;
    }
    .product-detail {
        flex-direction: column;
    }
    .chat-container {
        flex-direction: column;
        height: auto;
    }
    .chat-sidebar {
        width: 100%;
        border-right: none;
        border-bottom: 1px solid #ddd;
    }
    .chat-messages {
        height: 400px;
    }
}
@media (max-width: 768px) {
    .header-container {
        flex-wrap: wrap;
    }
    .menu-toggle {
        display: block;
    }
    .nav {
        position: fixed;
        top: 70px;
        left: 0;
        right: 0;
        background-color: var(--color-light);
        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        padding: 20px;
        transform: translateY(-100%);
        transition: var(--transition);
        opacity: 0;
        visibility: hidden;
        z-index: 999;
    }
        .nav.open {
            transform: translateY(0);
            opacity: 1;
            visibility: visible;
        }
    .nav-list {
        flex-direction: column;
        align-items: flex-start;
    }
    .nav-item {
        margin: 0;
        width: 100%;
        padding: 10px 0;
        border-bottom: 1px solid #ddd;
    }
        .nav-item:last-child {
            border-bottom: none;
        }
    .dropdown-menu {
        position: static;
```

```
            box-shadow: none;
            display: block;
            width: 100%;
            padding: 0;
            margin-top: 10px;
        }
            .dropdown-menu li {
                padding: 10px 0;
                border-bottom: 1px solid #ddd;
            }
                .dropdown-menu li:last-child {
                    border-bottom: none;
                }
        .hero-content {
            padding: 0 15px;
        }
        .filters-panel {
            flex-direction: column;
        }
        .search-form {
            margin-right: 0;
            margin-bottom: 15px;
            max-width: 100%;
        }
        .filter-item {
            margin-left: 0;
            margin-top: 15px;
            width: 100%;
        }
        .cart-item {
            flex-wrap: wrap;
        }
            .cart-item .item-details {
                width: 100%;
                margin-bottom: 10px;
            }
            .cart-item .item-quantity {
                margin: 10px 0;
            }
        .cart-actions {
            flex-direction: column;
            gap: 10px;
        }
        .footer-section {
            flex: 0 0 100%;
        }
    }
    @media (max-width: 576px) {
        .categories-grid, .products-grid {
            grid-template-columns: 1fr;
        }
        .product-detail .product-image {
            height: 300px;
        }
        .cart-item {
            flex-direction: column;
            align-items: flex-start;
        }
            .cart-item > * {
                margin-bottom: 10px;
            }
            .cart-item .item-image {
                margin-right: 0;
            }
            .cart-item .item-total {
```

```
            margin-right: 0;
        }
    .order-header {
        flex-direction: column;
        align-items: flex-start;
    }
    .order-status {
        margin-top: 10px;
    }
    .order-item {
        flex-wrap: wrap;
    }
}
/* Стилі для оновленого хедера з логотипом */
.header-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 15px;
    height: 80px; /* Збільшуємо висоту хедера */
}
.logo-container {
    flex: 0 0 200px; /* Резервуємо місце для логотипу */
}
.logo {
    display: flex;
    align-items: center;
    height: 100%;
}
    .logo img {
        max-height: 60px;
        max-width: 100%;
    }
/* Стилі для модальних вікон додавання товарів і категорій */
.modal {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: rgba(0, 0, 0, 0.5);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
    animation: fadeIn 0.3s ease;
}
@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}
.modal-content {
    background-color: var(--color-light);
    border-radius: 8px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.15);
    padding: 25px;
    width: 100%;
    max-width: 500px;
    max-height: 90vh;
    overflow-y: auto;
    position: relative;
```

```
    animation: slideIn 0.3s ease;
}
@keyframes slideIn {
    from {
        transform: translateY(-20px);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}
.modal-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 25px;
    padding-bottom: 15px;
    border-bottom: 1px solid var(--color-cream);
}
    .modal-header h2 {
        font-family: var(--font-heading);
        color: var(--color-gumidnt);
        margin: 0;
        font-size: 24px;
    }
.close-btn {
    background: none;
    border: none;
    font-size: 24px;
    cursor: pointer;
    color: var(--color-gray);
    transition: var(--transition);
    width: 30px;
    height: 30px;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 50%;
}
    .close-btn:hover {
        background-color: var(--color-cream);
        color: var(--color-dark);
    }
.product-form,
.category-form {
    display: flex;
    flex-direction: column;
    gap: 20px;
}
    .product-form .form-group,
    .category-form .form-group {
        margin-bottom: 0;
    }
    .product-form label,
    .category-form label {
        display: block;
        margin-bottom: 8px;
        font-weight: 500;
        color: var(--color-dark);
    }
    .product-form input,
    .product-form textarea,
    .product-form select,
    .category-form input,
```

```
    .category-form textarea,
    .category-form select {
        width: 100%;
        padding: 12px 15px;
        border: 1px solid #ddd;
        border-radius: 6px;
        font-size: 15px;
        transition: var(--transition);
    }
        .product-form input:focus,
        .product-form textarea:focus,
        .product-form select:focus,
        .category-form input:focus,
        .category-form textarea:focus,
        .category-form select:focus {
            border-color: var(--color-gumidnt);
            outline: none;
            box-shadow: 0 0 0 2px rgba(154, 109, 56, 0.1);
        }
.form-check {
    display: flex;
    align-items: center;
    margin-bottom: 0;
}
    .form-check input[type="checkbox"] {
        width: auto;
        margin-right: 10px;
        cursor: pointer;
        width: 18px;
        height: 18px;
    }
    .form-check label {
        margin-bottom: 0;
        cursor: pointer;
    }
.form-actions {
    display: flex;
    justify-content: flex-end;
    gap: 15px;
    margin-top: 10px;
}
        .form-actions button {
        min-width: 100px;
        padding: 10px 20px;
    }
/* Адаптивність для модальних вікон */
@media (max-width: 576px) {
    .modal-content {
        width: 90%;
        padding: 20px;
    }
    .form-actions {
        flex-direction: column;
    }
        .form-actions button {
            width: 100%;
        }
}
/* Фонове зображення для заголовків сторінок */
.page-title {
    font-family: var(--font-heading);
    text-align: center;
    margin-bottom: 30px;
    color: var(--color-light);
    padding: 40px 0;
```

```
    position: relative;
    background-image: url('https://images.unsplash.com/photo-1470509037663-
253afd7f0f51?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1920&q=80');
    background-size: cover;
    background-position: center;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.7);
    border-radius: 8px;
}
    .page-title::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background-color: rgba(0, 0, 0, 0.4);
        border-radius: 8px;
        z-index: 0;
    }
    .page-title h1, .page-title {
        position: relative;
        z-index: 1;
    }
/* Паралакс-ефект для каталогу */
.catalog-page {
    padding: 40px 0;
    position: relative;
    overflow: hidden;
}
    .catalog-page::before {
        content: "";
        position: fixed;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background-image: url('https://images.unsplash.com/photo-1462275646964-
a0e3386b89fa?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1920&q=70');
        background-size: cover;
        background-position: center;
        background-attachment: fixed;
        opacity: 0.15;
        z-index: -1;
    }
/* Покращення фільтрів для кращого відображення на паралаксі */
.filters-panel {
    background-color: rgba(230, 217, 192, 0.9);
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
}
/* Покращення стилів товарів для кращого відображення на паралаксі */
.product-card {
    background-color: var(--color-light);
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.15);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
    .product-card:hover {
        transform: translateY(-10px);
        box-shadow: 0 12px 30px rgba(0, 0, 0, 0.2);
    }
/* Анімація для паралакс-ефекту при прокрутці */
```

```
@media (min-width: 992px) {
    .catalog-page .container {
        position: relative;
        z-index: 1;
    }
    .products-grid {
        transition: transform 0.1s ease-out;
    }
}
/* Загальні стилі для фонів з паралакс-ефектом */
body {
    position: relative;
    background-color: var(--color-light);
}
    body::before {
        content: "";
        position: fixed;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background-image: url('https://images.unsplash.com/photo-1462275646964-
a0e3386b89fa?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1920&q=70');
        background-size: cover;
        background-position: center;
        background-attachment: fixed;
        opacity: 0.08;
        z-index: -1;
    }
/* Стилі для заголовків сторінок з фоновим зображенням */
.page-title {
    font-family: var(--font-heading);
    text-align: center;
    margin-bottom: 30px;
    color: var(--color-light);
    padding: 40px 0;
    position: relative;
    background-image: url('https://images.unsplash.com/photo-1470509037663-
253afd7f0f51?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D&auto=format&fit=crop&w
=1920&q=80');
    background-size: cover;
    background-position: center;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.7);
    border-radius: 8px;
    overflow: hidden;
}
    .page-title::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        right: 0;
        bottom: 0;
        background: linear-gradient(to right, rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.3));
        border-radius: 8px;
        z-index: 0;
    }
    .page-title h1, .page-title {
        position: relative;
        z-index: 1;
    }
/* Фони для різних сторінок */
```

```
.home-page,
.catalog-page,
.product-detail-page,
.cart-page,
.profile-page,
.orders-page,
.chat-page,
.auth-page,
.admin-products-page,
.admin-categories-page,
.admin-users-page {
    padding: 40px 0;
    position: relative;
    overflow: hidden;
}
/* Покращення контейнерів для кращого відображення на паралаксі */
.container {
    background-color: rgba(255, 255, 255, 0.9);
    border-radius: 10px;
    box-shadow: 0 5px 20px rgba(0, 0, 0, 0.08);
    padding: 20px;
    position: relative;
    z-index: 1;
}
/* Специфічні стилі для героя на головній сторінці */
.hero {
    background-attachment: fixed;
    position: relative;
    padding: 100px 0;
    margin-top: -20px;
    margin-bottom: 30px;
}
    .hero .container {
        background-color: transparent;
        box-shadow: none;
    }
.hero-content {
    background-color: rgba(0, 0, 0, 0.5);
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 5px 25px rgba(0, 0, 0, 0.2);
}
/* Покращення карток для категорій та товарів */
.category-card, .product-card {
    background-color: var(--color-light);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    border-radius: 10px;
    overflow: hidden;
    height: 100%;
}
    .category-card:hover, .product-card:hover {
        transform: translateY(-10px) scale(1.02);
        box-shadow: 0 15px 30px rgba(0, 0, 0, 0.15);
    }
.category-image, .product-image {
    height: 220px;
    overflow: hidden;
    position: relative;
}
        .category-image img, .product-image img {
        width: 100%;
        height: 100%;
        object-fit: cover;
        transition: transform 0.5s ease;
```

```
    }
.category-card:hover .category-image img,
.product-card:hover .product-image img {
    transform: scale(1.1);
}
/* Фільтри в каталозі */
.filters-panel {
    background-color: rgba(230, 217, 192, 0.9);
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
    margin-bottom: 30px;
    transition: transform 0.3s ease;
}
    .filters-panel:hover {
        transform: translateY(-3px);
        box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
    }
/* Стилі для форм авторизації та реєстрації */
.auth-form-container {
    background-color: rgba(255, 255, 255, 0.95);
    border-radius: 12px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
    transition: transform 0.3s ease;
}
    .auth-form-container:hover {
        transform: translateY(-5px);
        box-shadow: 0 15px 40px rgba(0, 0, 0, 0.2);
    }
/* Анімація для модальних вікон */
.modal-content {
    background-color: rgba(255, 255, 255, 0.98);
    box-shadow: 0 15px 50px rgba(0, 0, 0, 0.25);
    animation: modalZoomIn 0.3s ease;
}
@keyframes modalZoomIn {
    from {
        transform: scale(0.9);
        opacity: 0;
    }
    to {
        transform: scale(1);
        opacity: 1;
    }
}
/* Адаптивність для різних розмірів екрану */
@media (max-width: 992px) {
    .page-title {
        padding: 30px 0;
    }
    .container {
        padding: 15px;
    }
    .hero {
        padding: 60px 0;
    }
}
@media (max-width: 768px) {
    body::before {
        opacity: 0.05; /* Зменшуємо прозорість для малих екранів */
    }
}
```

package.json

```
{
  "name": "flower-shop",
  "version": "1.0.0",
  "description": "Інтернет-магазин Квіткова Лавка",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "server": "nodemon server.js",
    "dev": "concurrently \"npm run server\""
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2",
    "react-router-dom": "^7.5.0",
    "socket.io": "^4.8.1",
    "socket.io-client": "^4.8.1",
    "sqlite3": "^5.1.7"
  },
  "devDependencies": {
    "webpack-cli": "^6.0.1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

README.md

Додаток "Квіткова Лавка"

Цей веб-додаток представляє собою повноцінний онлайн-магазин квітів з підтримкою різних ролей користувачів. Додаток дозволяє клієнтам переглядати каталог товарів, додавати їх до кошика, оформлювати замовлення та спілкуватися з менеджерами через чат. Менеджери можуть керувати каталогом товарів, категоріями, переглядати замовлення та спілкуватися з клієнтами.

Можливості

Додаток реалізує функціонал електронної комерції, включаючи реєстрацію та авторизацію, перегляд каталогу, оформлення замовлень, управління контентом та чат між користувачами.

Для клієнтів доступні: перегляд каталогу, фільтрація та пошук товарів, додавання до кошика, оформлення замовлень, перегляд історії замовлень та спілкування з менеджерами.

Для менеджерів доступні: управління товарами (додавання, редагування, видалення), управління категоріями, перегляд замовлень клієнтів, зміна статусів замовлень та спілкування з клієнтами.

Запуск та використання

Для запуску додатку необхідно мати Node.js та NPM. Спочатку запустіть сервер через термінал, перейшовши в директорію з проектом та виконавши команду node server.js. Сервер буде доступний за адресою http://localhost:3001. Після цього відкрийте у браузері файл index.html або перейдіть за адресою http://localhost:3001 для використання додатку.

Для входу можна зареєструвати новий обліковий запис. При реєстрації можна вибрати роль "клієнт" або "менеджер". Після авторизації інтерфейс адаптується відповідно до ролі користувача.

Каталог товарів доступний усім користувачам. Для оформлення замовлення необхідно авторизуватися як клієнт, додати товари до кошика та перейти до оформлення. Для управління товарами та категоріями необхідно авторизуватися як менеджер.

Чат доступний авторизованим користувачам. Клієнти можуть спілкуватися з менеджерами, а менеджери – з клієнтами. Для початку чату необхідно перейти до відповідного розділу в меню та вибрати співрозмовника зі списку.

Додаток має адаптивний дизайн та зручний користувацький інтерфейс, що дозволяє комфортно користуватися ним як на десктопних, так і на мобільних пристроях.

ДОДАТОК Б

ОСНОВНІ ФОРМИ ЗАСТОСУНКУ

Рис. Б.1 Головна сторінка відображає привітальний банер, категорії товарів і популярні продукти з можливістю переходу до каталогу.

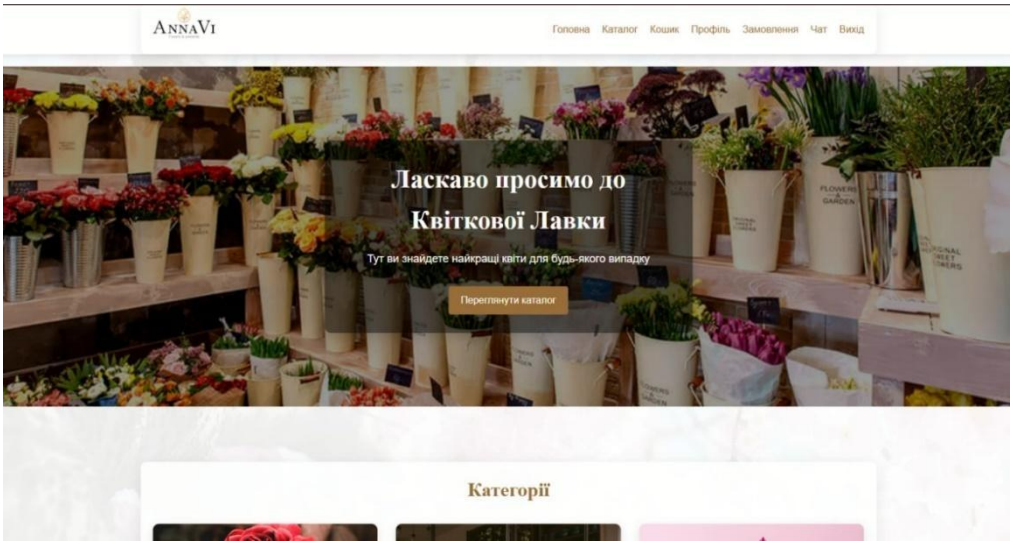

Рис. Б.1 Головна сторінка

Рис. Б.2 Сторінка каталогу. надає список товарів із фільтрами за категоріями, пошуком і сортуванням для зручного перегляду.

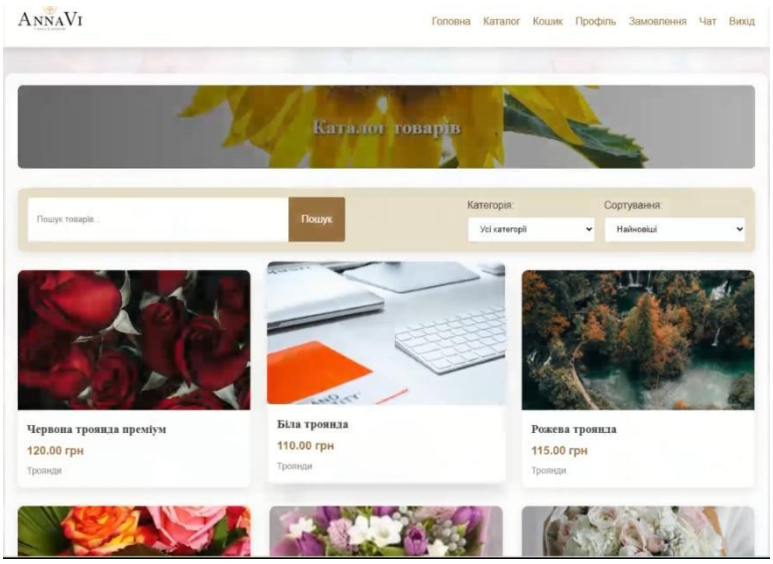

Рис. Б.2 Сторінка каталогу

Рис. Б.3 Картка товару показує детальну інформацію про товар, включаючи зображення, ціну, опис і можливість додавання до кошика

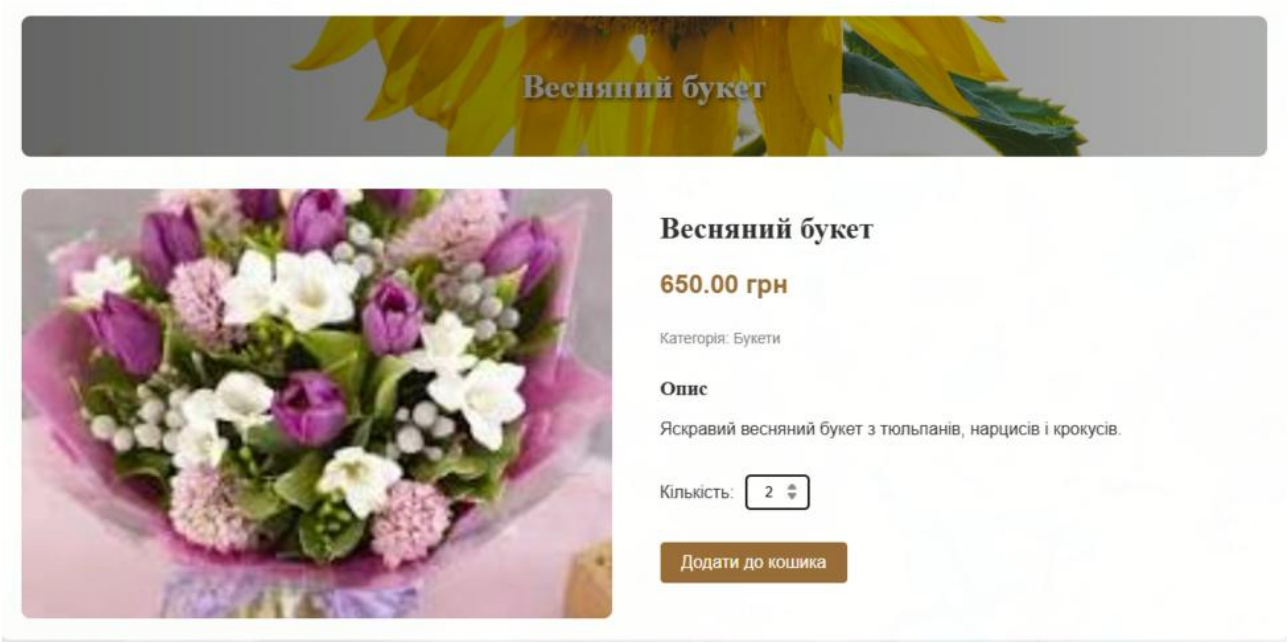

Рис. Б.3 Картка товару

Рис. Б.4 Кошик дозволяє переглядати, редагувати та видаляти товари в кошику, а також оформлювати замовлення.

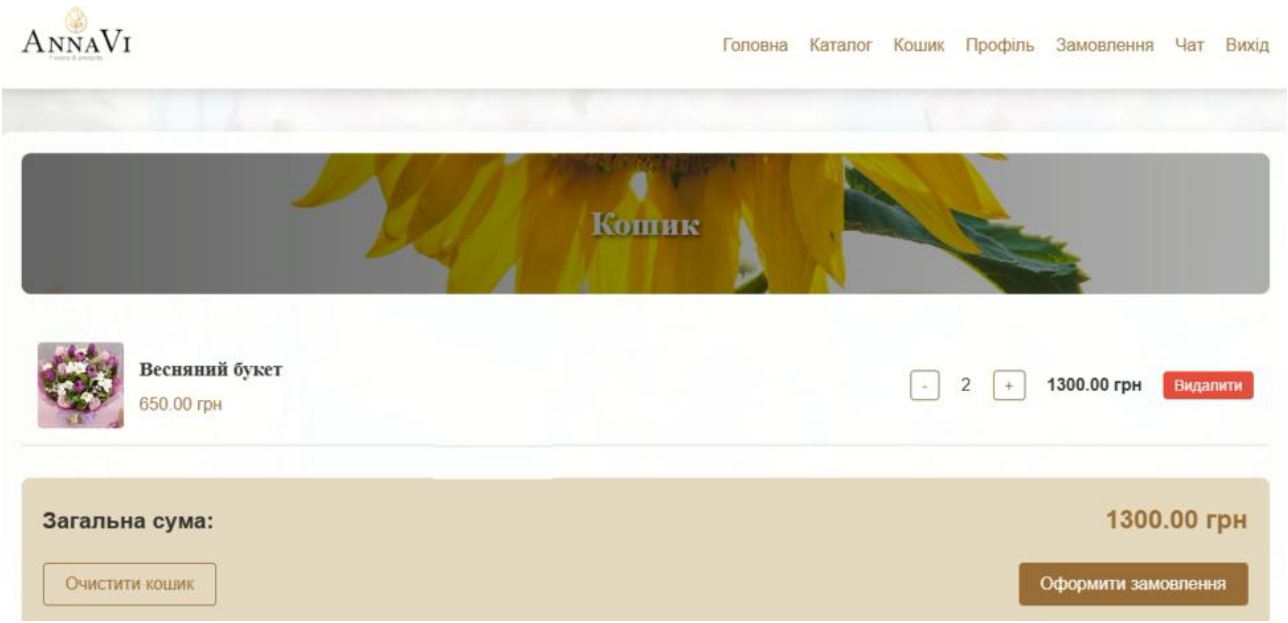

Рис. Б.4 Кошик

Рис. Б.5 Сторінка замовлень відображає історію замовлень користувача з деталями кожного замовлення

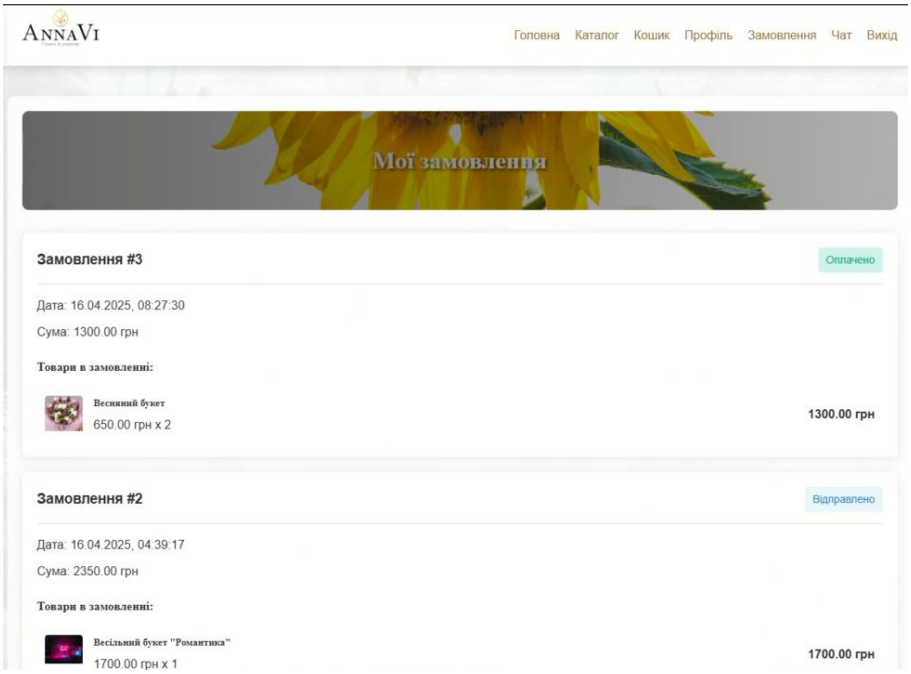

Рис. Б.5 Сторінка замовлень

Рис. Б.6 Профіль користувача. Особисті дані надає форму для перегляду та редагування особистих даних користувача, таких як ім’я, телефон і адреса.

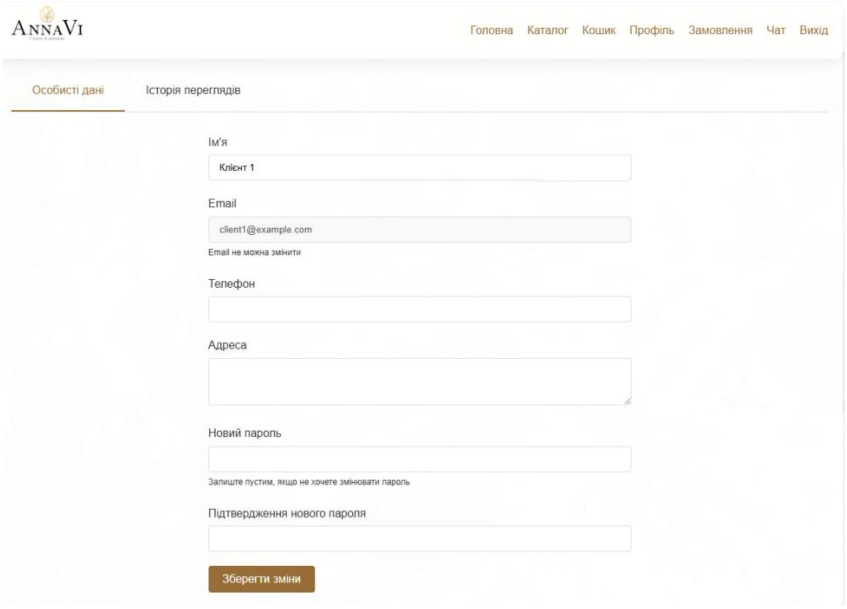

Рис. Б.6 – Профіль користувача. Особисті дані

Рис. Б.7 Профіль користувача. Історія переглядів показує список раніше переглянутих товарів для клієнтів.

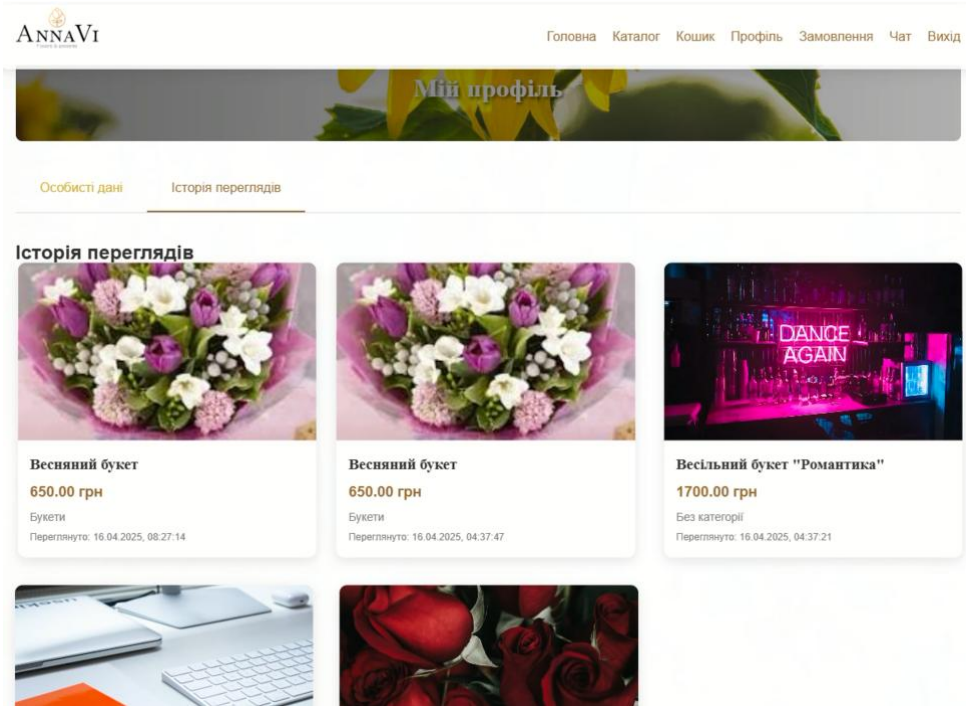

Рис. Б.7 – Профіль користувача. Історія переглядів

Рис. Б.8 Чат клієнта з адміністратором забезпечує інтерфейс для спілкування клієнтів із адміністратором у реальному часі через socket.io.

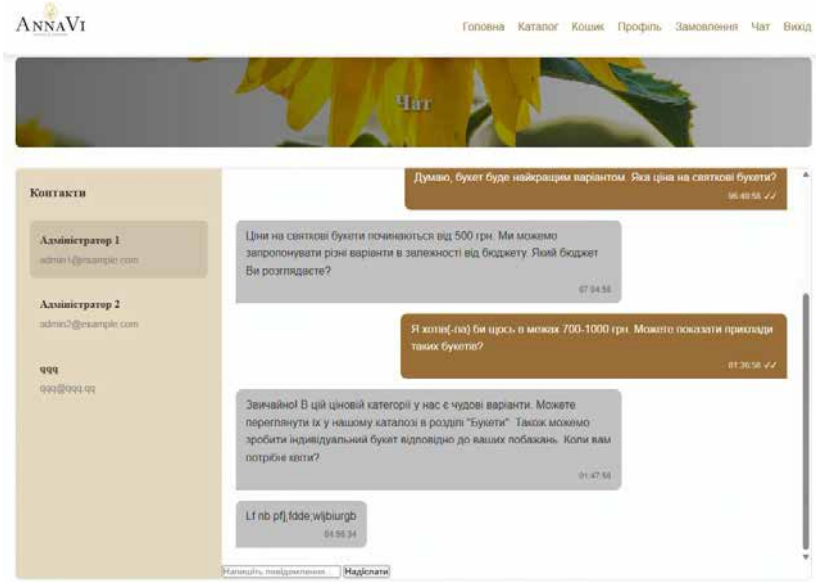

Рис. Б.8 Чат клієнта з адміністратором

Рис. Б.9 Чат для адміністратора дозволяє адміністратору керувати чатами з клієнтами та відповідати на їх повідомлення

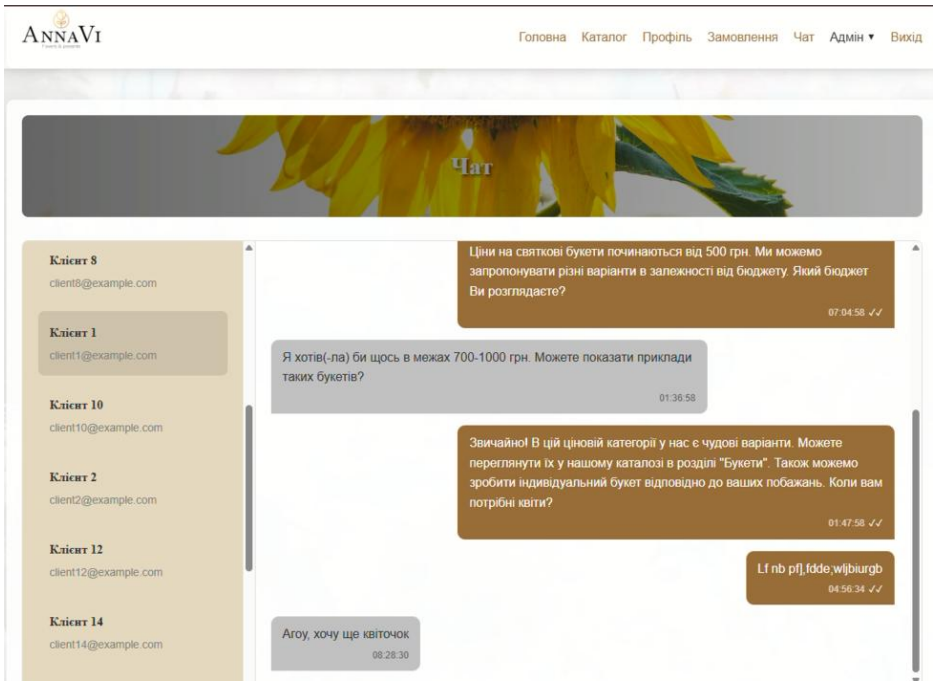

Рис. Б.9 Чат для адміністратора

Рис. Б.10 Сторінка замовлень для адміністратора надає адміністратору список усіх замовлень із можливістю перегляду та керування статусами.

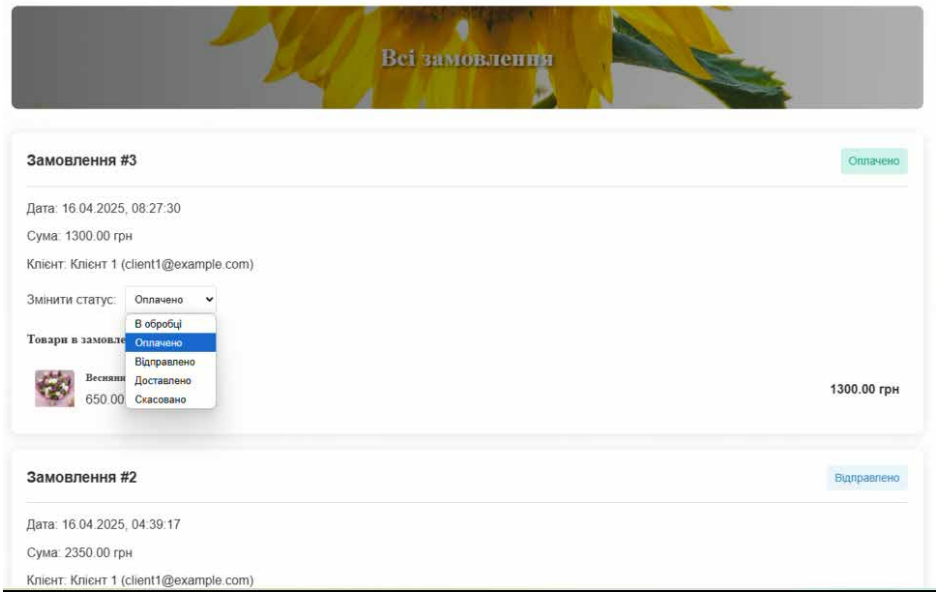

Рис. Б.10 Сторінка замовлень для адміністратора

Рис. Б.11 Сторінка товари (тільки для адміністратора) відображає список усіх товарів із можливістю їх редагування чи видалення.

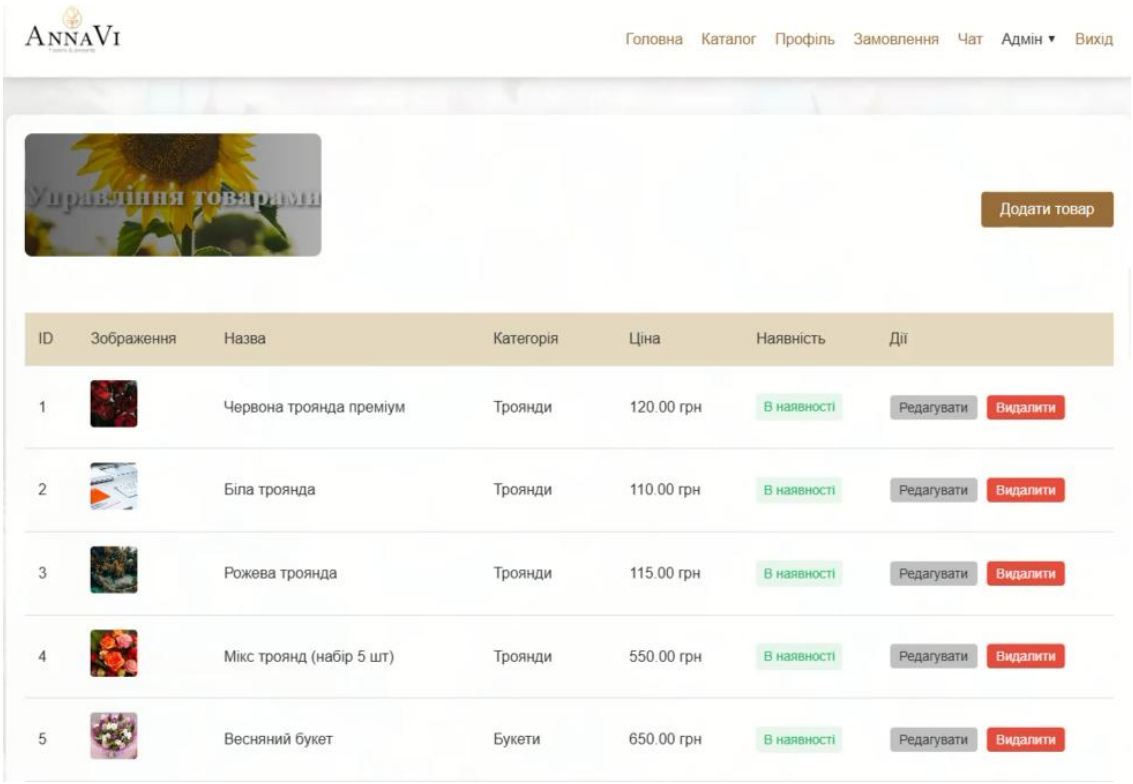

Рис. Б.11 Сторінка товари (тільки для адміністратора)

Рис. Б.12 Додавання нового товару (тільки для адміністратора) містить форму для створення нового товару з полями для назви, ціни, опису та зображення

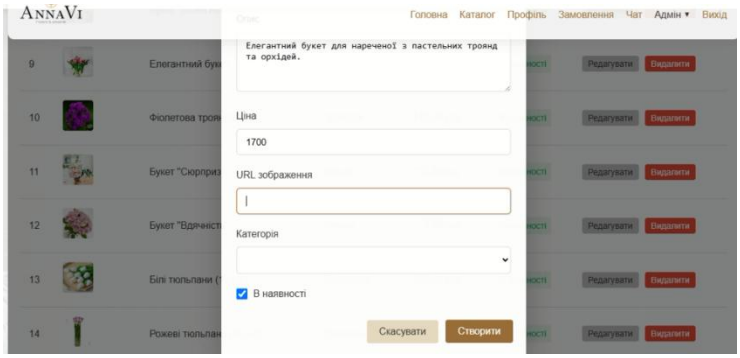

Рис. Б.12 Додавання нового товару (тільки для адміністратора)

Рис. Б.13 Редагування товарів (тільки для адміністратора) дозволяє змінювати дані існуючих товарів, включаючи ціну, опис і категорію.

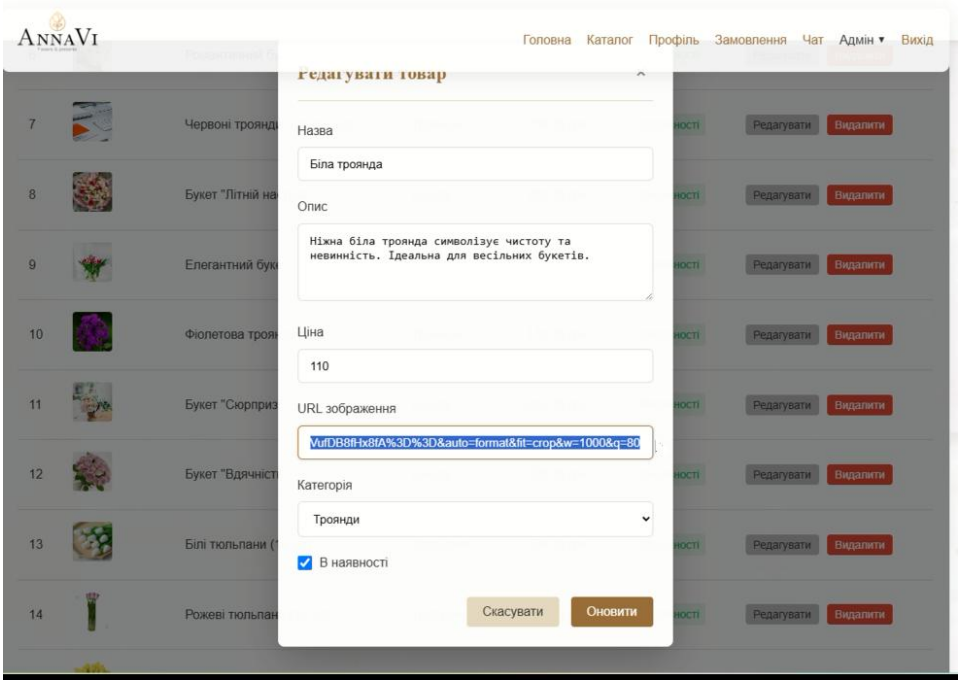

Рис. Б.13 Редагування товарів (тільки для адміністратора)

Рис. Б.14 Сторінка категорій (тільки для адміністратора) показує список категорій товарів із можливістю їх додавання, редагування чи видалення.

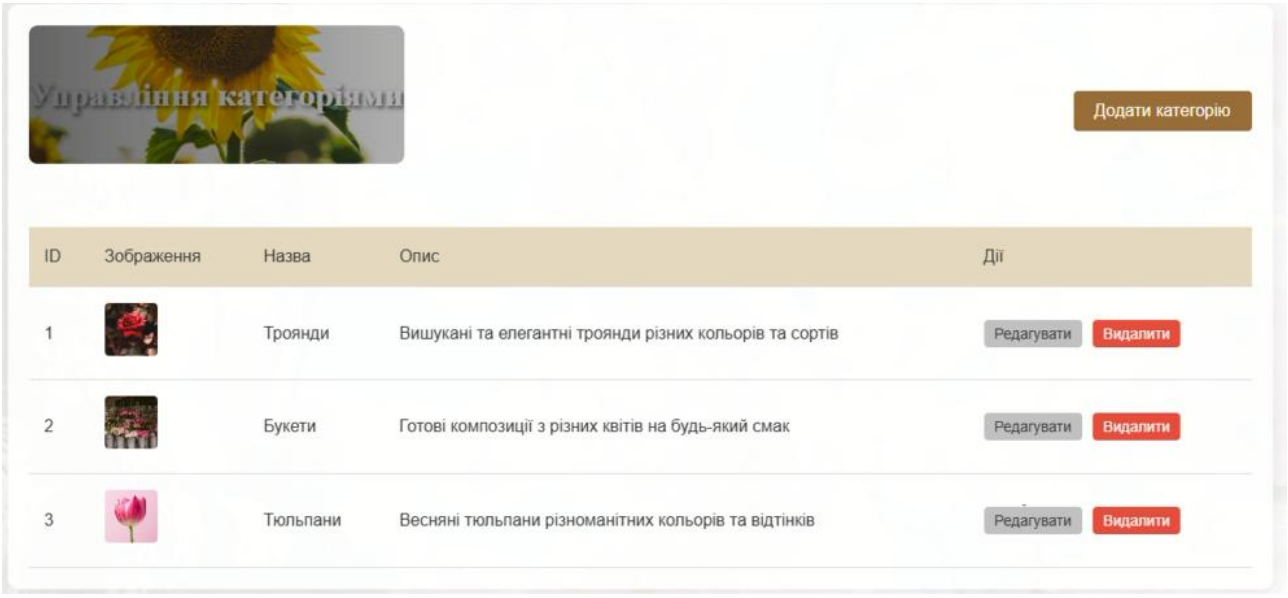

Рис. Б.14 Сторінка категорій (тільки для адміністратора)

Рис. Б.15 Додавання категорії (тільки для адміністратора) надає форму для створення нової категорії товарів.

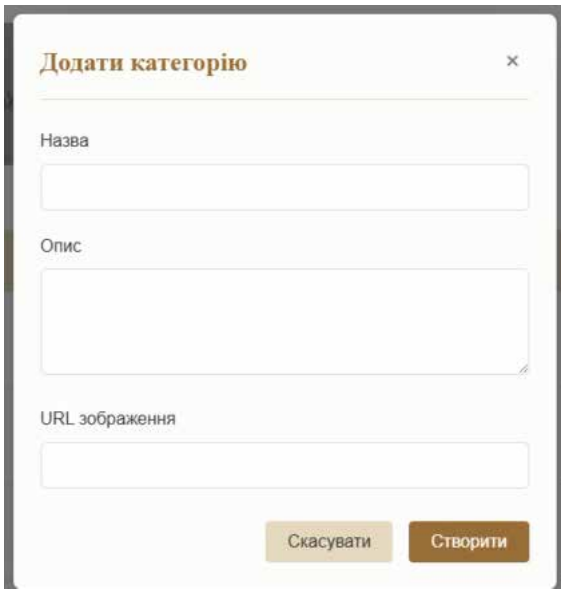

Рис. Б.15 Додавання категорії (тільки для адміністратора)

Рис. Б.16 Редагування категорії (тільки для адміністратора) дозволяє змінювати назву та інші параметри існуючої категорії.

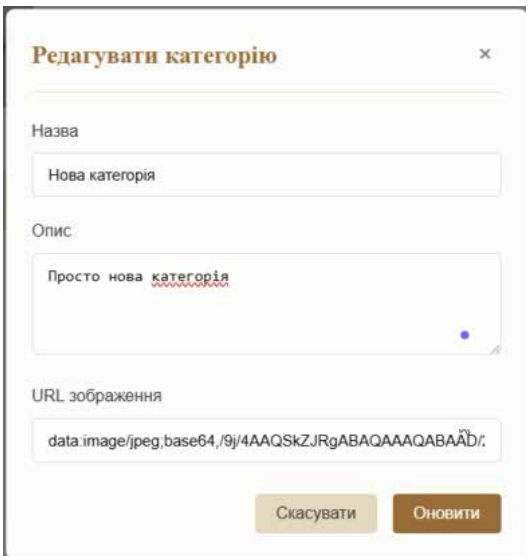

Рис. Б.16 Редагування категорії (тільки для адміністратора)

Рис. Б.17 Сторінка користувачі (тільки для адміністратора) відображає список усіх користувачів із можливістю керування їх даними та ролями.

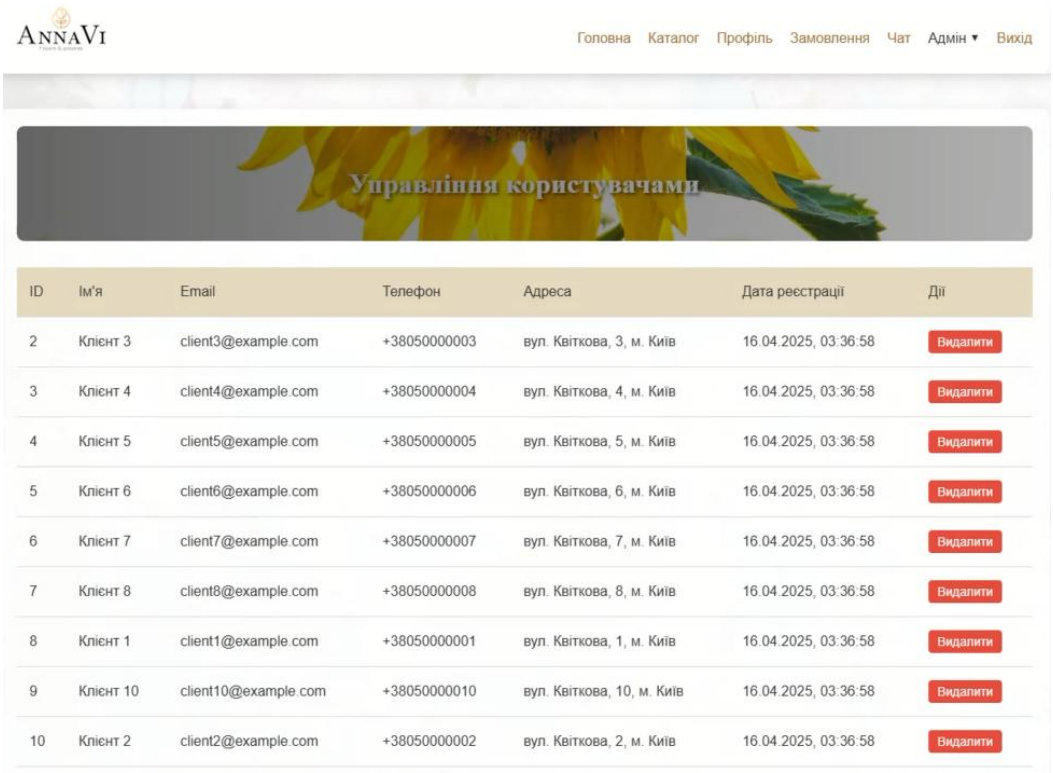

Рис. Б.17 Сторінка користувачі (тільки для адміністратора)

Рис. Б.18 Нижня панель інтернет магазину містить контактну інформацію, посилання на соціальні мережі та копірайт магазину.

Рис. Б.18 Нижня панель інтернет-магазину