НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ _____Голуб Б.Л._____
(підпис) (ПІБ)

"___" червня 2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Веб додаток інтернет магазину з продажу чайного товару та супроводження блогу покупців»

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.ф.-м.н., доцент____ _____ _Кириченко В.В._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _Смоляк Данило Валерійович____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОВИКОРИСТАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)

_____/ Голуб Б.Л. доцент к.т.н./
(підпис)

"___" _____2025р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

Смоляк Данило Валерійович

Спеціальність 122 – «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Веб додаток інтернет-магазину з продажу чайного товару та супроводження блогу покупців» затверджена наказом ректора НУБіП України №2246С від "16" грудня 2024 р.
2. Термін подання завершеної роботи на кафедру 2025 . 05 . _25
(рік, місяць, число)
3. Вихідні дані для роботи: надання інформації про роботу інтернет-магазину з продажу чайного товару у вигляді списків та таблиць.
4. Перелік питань, що розглядаються:
   1. Системний аналіз предметної області
   2. Проектування інформаційного та програмного забезпечення
   3. Розробка інформаційного та програмного забезпечення
   4. Рекомендації щодо впровадження та експлуатації системи
   5. Висновки

Керівник бакалаврської кваліфікаційної роботи_____/Кириченко В.В. /
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/ Смоляк Д.В/__
(підпис) (прізвище та ініціали)

Дата отримання завдання 2025.01.08
Рік місяць число

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій не стоїть на місці та невпинно розвивається, крім того його вплив на повсякденне життя становиться все більшим та більшим. Важко уявити будь-яку сферу без залучення інформаційних технологій, зокрема це стосується бізнесу. Веб технології стали невід'ємною частиною бізнесу дозволяючи підприємцям розширювати аудиторію, підвищувати прибутковість та надавати свої товари і послуги широкому колу користувачів - без обмежень у географічному розташуванні чи режимі роботи.

Онлайн-торгівля - одна з найбільш динамічних галузей, яка постійно змінюється відповідно до потреб ринку та користувачів. Інтернет-магазин це вже звичний інструмент не лише для великих корпорацій , а й для малого та середнього бізнесу. І при розробці кожен намагається зробити їх зручнішими як для себе, так і для клієнтів. В умовах високої конкуренції важливу роль відіграє не тільки якість товару, ще важливу роль має саме програмне забезпечення магазину - швидкість, простота, безпека, уважність до деталей та багато чого іншого.

Крім цього, розвинуті інтернет-магазини поєднують комерційну складову з тим що вони надають змогу користувачам обмінюватись інформацією через блоги, новини, статті, огляди. Саме це дозволяє налагодити комунікацію з аудиторією, підвищити не тільки обізнаність, ще й допомагає мотивувати клієнтів до покупки товару . Тому особливої актуальності набувають програмні рішення, що поєднують функціонал продажу з можливістю створення та управлінням контенту.

Тому розробка програмного забезпечення для інтернет-магазину з підтримкою блогу є актуальним завданням. Створене рішення повинно забезпечити ефективне управління товарним асортиментом і контентом, а також автоматизувати ключові бізнес-процеси. При цьому програмне забезпечення має відповідати наступним практичним критеріям:

1. Забезпечити зручний та інтуїтивно зрозумілий інтерфейс для кінцевого користувача та адміністратора системи.
2. Реалізувати функціонал повноцінного кошика з можливістю додавання, видалення товарів, обчислення загальної вартості та формування замовлення.
3. Реалізувати систему фільтрації, сортування та пошуку товарів, що дозволить користувачам знаходити потрібні продукти.
4. Інтегрувати блог-платформу, яка дозволить адміністратору створювати, редагувати та публікувати тематичні статті, новини, огляди тощо.
5. Забезпечити базовий функціонал авторизації та реєстрації користувачів.
6. Реалізувати збереження даних у базі даних.
7. Забезпечити можливість масштабування системи, що дозволить у майбутньому додавати новий функціонал.
8. Автоматизувати частину адміністративних дій.

Перш за все система повинна бути масштабованою, стабільною, та мати одночасно простий та зрозумілий інтерфейс для кожного користувача. Всі ці критерії дозволять не тільки надалі розвивати веб-додаток, а й отримати в кінці готовий автоматизований продукт для подальшого розвитку.

Щоб розробити такий веб-продукт, потрібно використовувати сучасні технології та методи розробки, які дозволять забезпечити стабільну роботу системи, гнучкість у розширенні функціоналу, ефективну взаємодію з базою даних та зручність користування як для клієнтів, так і для адміністратора та окрему увагу слід приділити питанням безпеки даних: забезпечення надійного збереження даних, захисту персональної інформації користувачів та запобіганню її несанкціонованому доступу . Методи та технології, які застосовувалися під час розробки програмного забезпечення, були обрані з урахуванням таких чинників, як надійність, поширеність, підтримка українськими інституціями, зручність для розробника та кінцевого користувача. Зокрема, були використані такі технології та інструменти:

1. Архітектура "клієнт-сервер" - чітке розділення фронтенду (Angular) і бекенду (ASP.NET Core Web API).
2. RESTful API - дозволяє ефективно взаємодіяти між клієнтською частиною та сервером, використовуючи стандартизовані HTTP-запити.
3. JWT (JSON Web Token) - використовується для реалізації автентифікації та авторизації користувачів, забезпечуючи контроль доступу до захищених ресурсів та захист персональних даних.
4. Specification Pattern - застосовується для реалізації гнучкої фільтрації, сортування, пагінації та пошуку товарів через запити до бази даних.
5. Хешування паролів- забезпечує збереження облікових даних у безпечному вигляді в базі даних.
6. Розмежування прав доступу - система ролей користувачів, яка дозволяє обмежити доступ до функціоналу відповідно до рівня прав.
7. Entity Framework Core - ORM для роботи з базою даних MSSQL, що дозволить ефективно реалізувати CRUD-операції.
8. MSSQL Server - надійна реляційна база даних, яка зберігає інформацію про товари, замовлення, користувачів, блог-записи тощо.
9. Механізм рефреш-токенів - дозволяє продовжити сесію користувача без повторної авторизації, зберігаючи при цьому безпечний доступ до ресурсу.
10. Документація API (Postman) - полегшує тестування та супровід API завдяки зручному інтерфейсу для виклику запитів.

Таким чином, розробка програмного забезпечення для інтернет-магазину з підтримкою блогу це сучасне рішення для задоволення потреб користувачів що цінують чай, та повністю відповідає викликам сучасного цифрового середовища та вимогам бізнесу.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Ринок чайної продукції має свої особливості як у звичайній торгівлі, так і в Інтернеті. Чай, як товар, представлений у широкому асортименті - за країною походження, видом (чорний, зелений, білий, улун тощо), смаковими характеристиками, запахом, історією та формою випуску. Крім того, популярність культури споживання чаю створює умови для розвитку спільнот, що об'єднуються навколо тем здорового способу життя, традицій заварювання, відкриття нових смаків і культурних практик. Це зумовлює зростаючий попит на якісну продукцію, достовірну інформацію та персоналізований підхід у продажу. Споживачі чайної продукції, як правило, орієнтовані на натуральність, якість, походження сировини, корисні властивості та етичність виробництва. Це формує потребу в детальному представленні кожного товару, що включає опис складу, рекомендації щодо заварювання, історичні факти про походження, методи збору та обробки. Таким чином, онлайн-магазини повинні не лише забезпечувати стандартну функцію купівлі, а й виступати джерелом нової інформації для користувачів, що підвищить довіру, лояльність клієнта до продукції та надасть змогу дізнатися більше про товар.

1.1.1 Візитка товару

Для користувачів інтернет-магазину ключове значення має якість та повнота інформації, представленої на сторінці кожного продукту, система повинна забезпечувати збереження та виведення даних про товар. До необхідних атрибутів, що формують повний опис чаю та допомагають користувачеві зробити усвідомлений вибір, належать: повна назва чаю, детальний опис, що розкриває його особливості, ціна, яка може варіюватися залежно від ваги або типу упаковки, та категорія (наприклад, чорний, зелений, фруктовий). Додатково, для цінителів чайної культури критично важливою є інформація про

походження або історію конкретного сорту, рекомендації щодо способу заварювання для досягнення найкращого смаку, а також властивості чаю (енергетичні, розслаблюючі, оздоровчі). Наявність якісних фотографій продукту є обов'язковою складовою, оскільки візуальний аспект значно впливає на рішення покупця. Сукупність цих даних дозволить користувачеві вибрати саме той товар, який найкраще відповідає його індивідуальним смакам, цілям та очікуванням від покупки.

1.1.2 Ціноутворення

Ціна на чайну продукцію залежить від багатьох факторів: типу чаю, регіону вирощування, екологічності, способу обробки, бренду, а також виду упаковки (розсипний, у пакетиках, пресований). Крім того, часто реалізується гнучка система фасування (наприклад, 50 г, 100 г, 250 г), що впливає на кінцеву вартість для споживача. В онлайн-магазині важливо реалізувати механізм вибору об’єму з автоматичним перерахунком ціни.

1.1.3 Значення блогу

Блог на сайті інтернет-магазину є важливим елементом взаємодії з клієнтом. Його основні функції:

- Формування експертного іміджу магазину
- Освітній контент - статті про історію чаю, способи заварювання, традиції різних країн
- Інформування - огляд новинок, акцій, сезонних пропозицій
- Побудова спільноти - спілкування через коментарі, поширення досвіду

Блог сприяє зростанню лояльності, довіри до бренду, а також допомагає відвідувачеві зробити більш усвідомлений вибір.

1.1.4 Коментарі та відгуки користувачів

Можливість залишати коментарі та відгуки є важливою функцією будь-якого сучасного інтернет-магазину. У контексті чайної продукції це дозволяє:

- Ділитися досвідом використання продукції

- Формувати соціальний доказ якості
- Отримувати зворотній зв’язок для покращення асортименту
- Впливати на прийняття рішення іншими покупцями

Механізм коментування повинен бути простим, безпечним, бажано з можливістю модерування, оцінювання (лайки, зірки) та сортування коментарів.

1.1.5 Кошик користувача

Кошик є одним з ключових елементів інтернет-магазину, що забезпечує зручність взаємодії з товаром перед оформленням замовлення. Для ринку чайної продукції важливо, щоб кошик враховував:

- Можливість додавати товари в різній фасовці (вага, тип пакування)
- Миттєве оновлення вартості кошика.
- Збереження стану кошика при будь-яких діях користувача
- Можливість редагування та видалення інформації
- Відображення додаткової інформації (назва, фото ,ціна за одиницю)

1.1.6 Панель адміністратора

Для ефективного управління онлайн-магазином необхідна зручна адміністративна панель, яка дає змогу:

- Додавати, редагувати та видаляти товари (назва, опис, фото, ціна, категорія, країна походження тощо)
- Керувати замовленнями: статуси, перегляд деталей
- Аналізувати продажі та перегляди
- Керувати контентом блогу: створення, редагування, публікація
- Змінювати, слідкувати та змінювати коментарі та блоги користувачів

Панель адміністратора повинна забезпечувати простоту, безпеку та функціональність, адаптовану під потреби малого або середнього бізнесу.

1.1.7 Основні проблеми предметної області

У сфері онлайн-продажу чайної продукції можна виділити кілька основних проблем:

- Складність сприйняття термінів (Нові користувачі, не обізнані з чайною культурою, стикаються з труднощами у розумінні специфічних термінів, що ускладнює вибір продукції.)
- Обмежена фільтрація: користувачі не можуть зручно знаходити продукцію за країною, властивостями або типом
- Відсутність якісної взаємодії з відвідувачами (немає блогу, коментарів, відгуків)
- Низький рівень персоналізації (немає рекомендацій на основі попередніх переглядів чи купівель)
- Обмежений контроль з боку адміністратора (ручна обробка замовлень, слабка аналітика)

1.2 Аналіз вимог до предметної області

Аналіз вимог до майбутньої системи критично важливе, воно дозволить проаналізувати потреби користувачів та бізнес-процесів. Вимоги поділяються на функціональні, що визначають передбачувані дії системи, та нефункціональні, що стосуються якісних аспектів її роботи. Додатково, вимоги можуть специфікуватися відповідно до прав доступу та обов'язків різних ролей користувачів, зокрема, для кінцевого споживача та адміністратора

1.2.1 Функціональні вимоги

Функціональні вимоги описують, що система повинна робити. Для онлайн-магазину чайної продукції з блогом:

1. **Реєстрація та автентифікація користувачів:**
   - Реєстрація з валідацією полів (email, пароль, ім’я).
   - Підтримка ролей (користувач, адміністратор).
   - Вхід та вихід з системи.
   - Зберігання сесії за допомогою JWT (автентифікація токеном).
2. **Перегляд та пошук товарів:**
   - Сортування та фільтрація товарів (за видом, країною, ціною тощо).

- Пошук за ключовими словами.
- Відображення детальної інформації про товар (назва, опис, фото, ціна, рейтинг, відгуки).

3. **Робота з кошиком:**
   - Додавання або видалення товарів з кошика.
   - Зміна кількості одиниць кожного товару.
   - Підрахунок загальної суми замовлення.
   - Збереження кошика між сесіями користувача.
4. **Оформлення замовлення:**
   - Заповнення контактної інформації.
   - Вибір способу доставки.
   - Створення замовлення та підтвердження.
   - Повідомлення користувача про статус замовлення (прийнято, в обробці, доставлено).
5. **Панель адміністратора:**
   - CRUD-операції над товарами, категоріями, блогами.• Управління наявністю товарів.
   - Перегляд та обробка замовлень (зміна статусів, видалення).
   - Видалення коментарів.
6. **Блог:**
   - Перегляд та фільтрація статей за темами або датою.
   - Прив’язка статей до товарів або категорій.
   - Можливість коментування статей зареєстрованими користувачами.
7. **Система коментарів / відгуків:**
   - Авторизовані користувачі можуть залишати відгуки.
   - Можливість поставити оцінку (рейтинг від 1 до 5).

1.2.2 Нефункціональні вимоги

1. **Продуктивність:**

- Швидке завантаження сторінок навіть при великій кількості товарів (>1000).
- Мінімальні затримки (<2 секунд) при додаванні в кошик, оформленні замовлення або авторизації.

2. **Безпека:**
   - Авторизація через JWT з оновленням токенів.
   - Захист від атак типу SQL-ін’єкцій, XSS, CSRF.
   - Обмеження доступу до адміністративного функціоналу.
3. **Масштабованість:**
   - Можливість розширення функціоналу (акції, бонусна система, інтеграція з CRM, додатки).
   - Підтримка великої кількості одночасних користувачів (від 100 до 1000+).
   - Гнучка архітектура, що дозволяє розділення на мікросервіси в майбутньому.
4. **Стиль та юзабіліті (UX/UI):**
   - Мінімалістичний, але інформативний дизайн з акцентом на візуалізацію товару.
   - Чітка навігація, інтуїтивно зрозумілий інтерфейс для користувача будь-якого рівня.
   - Єдина стилістика по всьому сайту (кольорова схема, шрифти, кнопки).
   - Використання анімацій та підсвіток для покращення взаємодії з користувачем.

1.2.3 Обмеження.

Система створюється в рамках навчального проекту, тому накладаються деякі обмеження:

- Відсутність мобільного застосунку, лише на десктопна веб-версія.

- Оплата товарів реалізується лише на концептуальному рівні без інтеграції з реальними платіжними системами.
- Система не передбачає багатомовності - використовується лише українська мова.
- Адміністративна панель має базовий функціонал і доступ через локальний обліковий запис.
- Всі дані зберігаються локально у базі даних - без інтеграції з хмарними сховищами
- Система не забезпечує повного дотримання законодавчих вимог щодо захисту персональних даних, оскільки розробляється з навчальною метою
- Проект не буде розгортатися на хостингу або сервері - кінцевий результат буде представлений лише як вихідний код у репозиторії GitHub

1.3 Моделювання предметної області

Моделювання предметної області є одним із ключових етапів розробки програмного забезпечення, особливо коли йдеться про створення складної системи, яка охоплює бізнес-логіку, взаємодію користувачів, управління даними та адміністрування. Це дозволяє:

- зрозуміти реальні потреби користувачів і бізнесу, які система має задовольнити;
- виявити основні об’єкти та процеси в межах предметної області, які необхідно підтримувати в програмному продукті;
- визначити ролі користувачів та сценарії їхньої взаємодії із системою, що забезпечує чітке бачення функціональності;
- зменшити кількість помилок у реалізації, оскільки модель дозволяє заздалегідь виявити логічні недоліки або суперечності;
- забезпечити основу для подальшого проектування структури бази даних та архітектури програмного продукту.

Модель предметної області - це умовне уявлення про реальний світ, яке дозволяє структурувати знання про систему й побудувати ефективне, масштабоване та зрозуміле рішення.

1.3.1 Діаграма прецедентів (Use Case Diagram).

Діаграма варіантів використання (Use Case) є важливим елементом моделювання програмного забезпечення, оскільки вона дає змогу візуалізувати функціональні вимоги до системи з точки зору користувачів. Вона показує, як саме користувачі (актори) взаємодіють із системою, та які завдання вони можуть у ній виконувати. Така діаграма визначає набір дій або сценаріїв, які система повинна підтримувати у співпраці з акторами, тим самим описуючи очікувану поведінку системи на рівні її зовнішніх функцій без розкриття внутрішньої реалізації.

Це дає змогу сформулювати чітке уявлення про функціональність майбутнього програмного продукту, а також слугує основою для подальшої розробки інших моделей, таких як діаграми класів, послідовностей, активностей тощо.

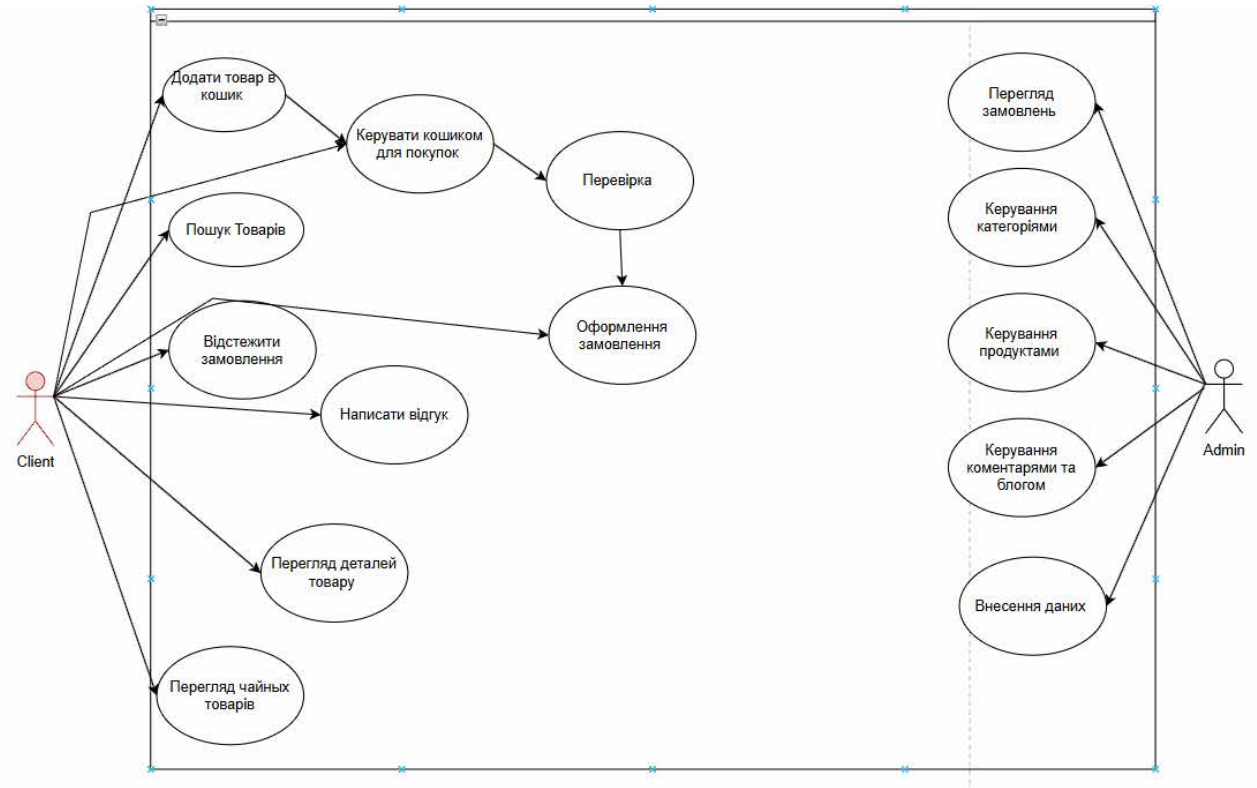

Рис. 1.1 – Діаграма прецедентів (Use Case Diagram)

На представленій діаграмі виділено два основні актори:

1. **Клієнт (Client)** - звичайний користувач системи, який може здійснювати покупки та взаємодіяти з електронним каталогом.
2. **Адміністратор (Admin)** - користувач з розширеними правами, відповідальний за керування системою та її контентом.

Прецеденти для актора "Клієнт"

1. **Додати товар в кошик**

- Дозволяє клієнту вибрати потрібний товар та додати його до віртуального кошика
- Передбачає можливість вказати кількість товару
- Зберігає вибір для подальшого оформлення замовлення

2. **Пошук товарів**

- Забезпечує можливість знаходити необхідні товари за ключовими словами
- Дозволяє застосовувати фільтри за різними параметрами (ціна, категорія, тощо)
- Надає результати у зручному для перегляду форматі

3. **Відстежити замовлення**

- Дозволяє клієнту контролювати процес виконання своїх замовлень
- Надає інформацію про статус (прийнято, обробляється, відправлено, доставлено)
- Містить деталі щодо способу та термінів доставки

4. **Написати відгук**

- Дає можливість залишити коментар та оцінку щодо придбаного товару
- Допомагає іншим користувачам у виборі товарів
- Забезпечує зворотний зв'язок для продавців та адміністраторів

5. **Перегляд деталей товару**

- Надає розгорнуту інформацію про обраний товар
- Включає технічні характеристики, фотографії, опис, ціну

- Показує наявність товару та можливі варіанти доставки

6. **Перегляд чайних товарів**

- Спеціалізований розділ для перегляду популярних товарів
- Може включати акційні пропозиції або товари зі знижками
- Формується на основі статистики продажів або рекомендацій адміністратора

7. **Керувати кошиком для покупок**

- Дозволяє редагувати вміст кошика (змінювати кількість, видаляти товари)
- Показує загальну вартість покупки та можливі знижки
- Є проміжним кроком перед оформленням замовлення

Прецеденти для актора "Адміністратор"

1. **Перегляд замовлень**

- Забезпечує доступ до всіх замовлень у системі
- Дозволяє фільтрувати та сортувати замовлення за різними параметрами
- Надає детальну інформацію про кожне замовлення та клієнта

2. **Керування продуктами**

- Дозволяє додавати нові товари до каталогу
- Забезпечує можливість редагування інформації про існуючі товари
- Дозволяє видаляти застарілі або недоступні товари

3. **Керування коментарями та блогом**

- Забезпечує модерацію відгуків користувачів
- Дозволяє публікувати статті та новини в блозі
- Надає інструменти для підтримки комунікації з клієнтами

4. **Внесення даних**

- Дозволяє оновлювати системну інформацію
- Забезпечує наповнення бази даних різними типами контенту
- Може включати роботу з імпортом/експортом даних

1.3.3 Діаграма послідовності (Sequence Diagram). Діаграма послідовності це елемент проектування, який показує, як різні частини системи (наприклад, користувач, інтерфейс, сервіси) взаємодіють між собою у часі. Вона допомагає зрозуміти послідовність дій під час основних бізнес-процесів (наприклад, оформлення замовлення), фіксує порядок викликів між компонентами, полегшує комунікацію в команді та слугує основою для розробки програмного коду. Наведена діаграма послідовності моделює процес оформлення замовлення клієнтом у системі.

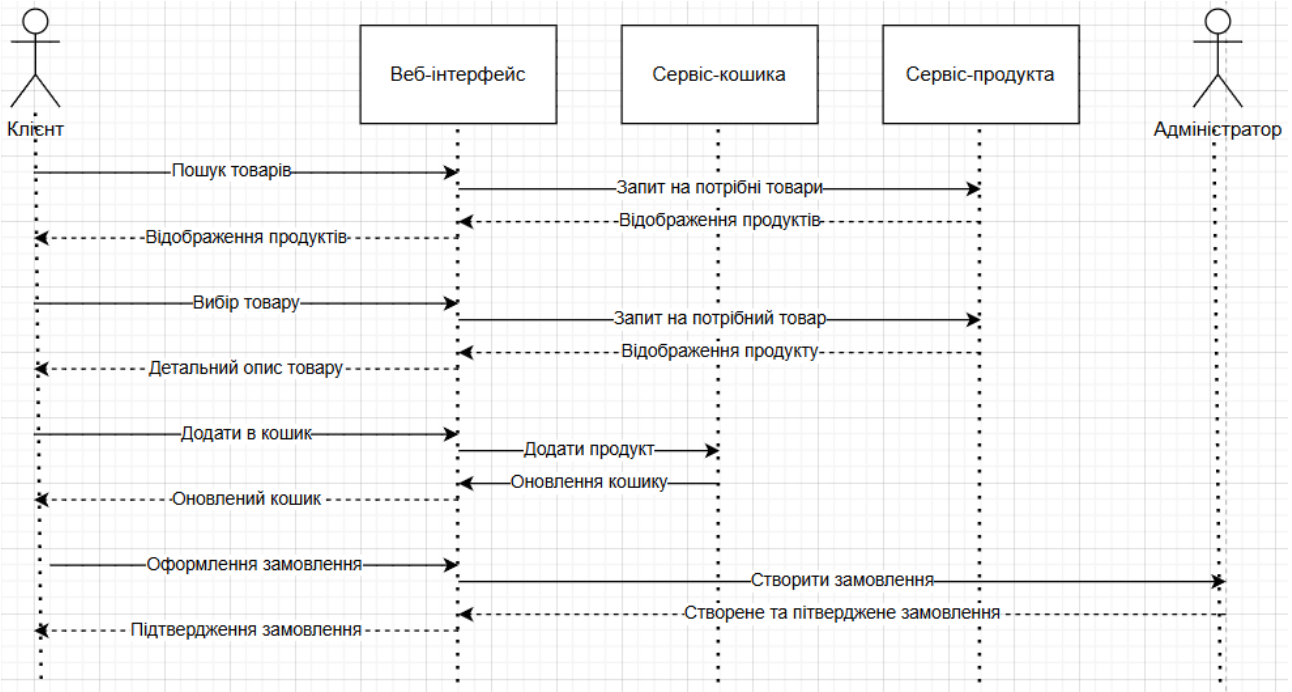

Рис. 1.2 – Діаграма послідовності

Учасники системи:

- Клієнт - шукає та купує товари (пошук, перегляд, додавання до кошика, оформлення замовлення).
- Адміністратор - керує товарами, обробляє замовлення, модерує контент.

Основні компоненти:

- Веб-інтерфейс - забезпечує взаємодію користувача із системою через браузер. Відображає дані, обробляє дії клієнта, передає запити до серверу.

- Сервіс продуктів - відповідає за каталог товарів: пошук, перегляд, перевірка наявності.
- Сервіс кошика - обробляє операції з кошиком: додавання, оновлення кількості, підрахунок вартості.

Основні потоки взаємодії:

- `Пошук товару: Клієнт → Веб-інтерфейс → Сервіс продуктів → Веб-інтерфейс → Клієнт.`
- Перегляд товару: Клієнт отримує детальну інформацію про вибраний товар.
- Додавання до кошика: Запит на додавання, оновлення стану кошика.
- Оформлення замовлення: Передача замовлення адміністратору для обробки та підтвердження

1.3.4 Діаграма активності (Activity Diagram).

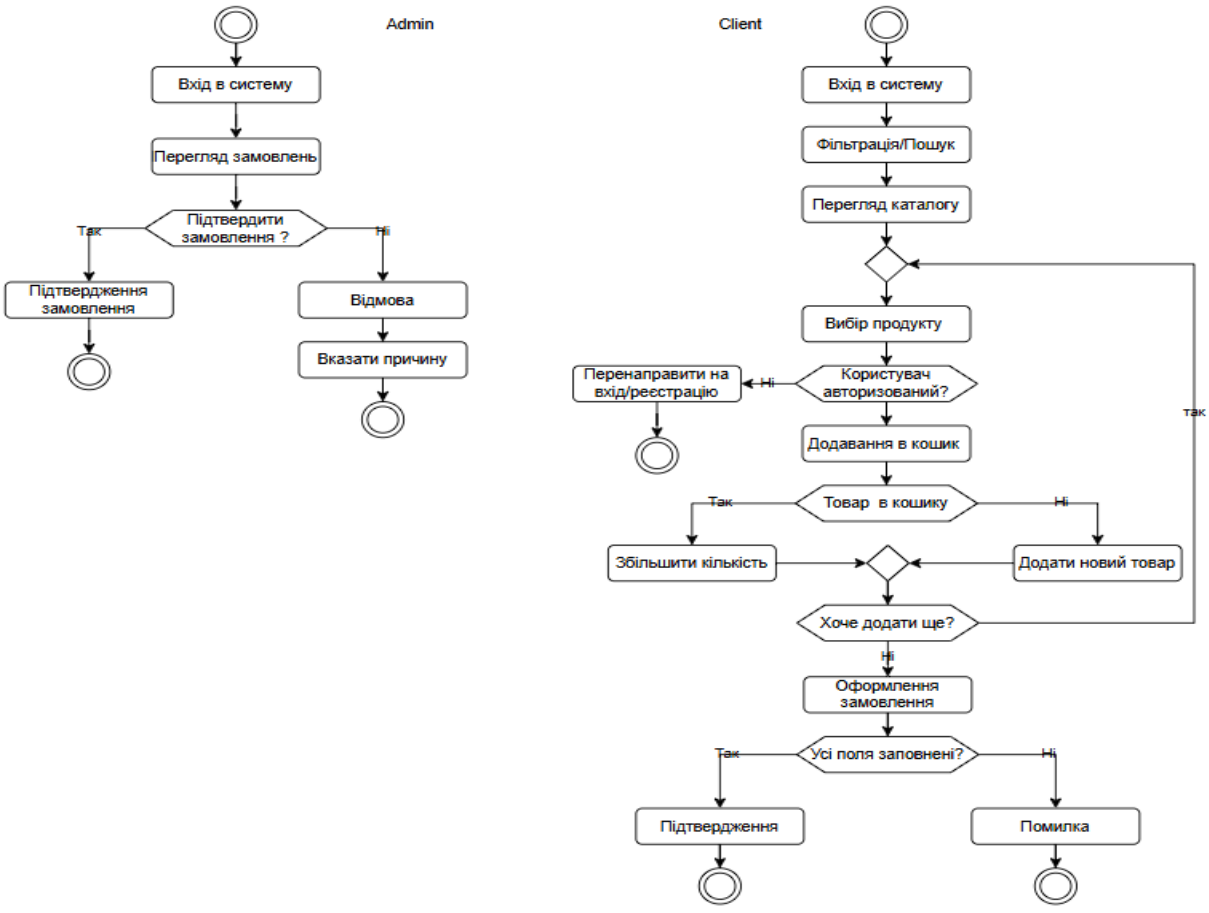

Рис. 1.3 – Діаграма активності

Представлена діаграма активностей відображає основні процеси взаємодії користувача та адміністратора на веб-платформі електронної комерції.

1. Потік Дій Користувача (права частина):

- Починається з "Вхід в систему", далі йде "Фільтрація/Пошук" та "Перегляд каталогу".
- Після перевірки авторизації користувач здійснює "Вибір продукту" та "Додавання в кошик" (з можливістю "Збільшити кількість" або "Додати новий товар").
- Цикл "Хоче додати ще?" дозволяє додавати кілька товарів.
- Завершується потік "Оформлення замовлення", перевіркою "Усі поля заповнені?" та переходом до "Підтвердження" або "Помилка".

2. Потік Дій Адміністратора (ліва частина):

- Починається з "Вхід в систему" адміністратора.
- Наступним кроком є "Перегляд замовлень".
- Адміністратор приймає рішення "Підтвердити замовлення?":
- У разі підтвердження – "Підтвердження замовлення".
- У разі відмови – "Відмова" та "Вказати причину".

1.3.5 Значення для реалізації.

Використання UML-діаграм на етапі проектування дає змогу:

- уникнути двозначностей у розумінні вимог;
- чітко окреслити функціональність та поведінку системи;
- спростити комунікацію між замовником і розробником;
- забезпечити основу для реалізації програмних модулів.

Завдяки створенню діаграм прецедентів, послідовності та активності, стає можливим точне моделювання логіки роботи системи, що дозволяє уникнути суттєвих помилок у процесі її реалізації.

1.4 Постановка завдання

1.5.1 Загальна характеристика задачі.

Сучасний ринок електронної комерції активно розвивається, і бізнесу потрібно швидко пристосовуватись до нових умов та ефективно керувати своїми процесами. Старі підходи або застарілі програми часто не дають змоги якісно автоматизувати роботу та контролювати важливі операції.

Це може призводити до таких проблем, як незручне управління товарами, складність у відстеженні замовлень, відсутність єдиної бази клієнтів або обмежені можливості аналізу поведінки покупців. Без сучасної цифрової системи компанії ризикують втратити конкурентні переваги, працювати неефективно та не встигати реагувати на зміни ринку.

Тому виникає потреба у створенні зручної, надійної та функціональної веб-системи для управління онлайн-магазином, яка вирішить ці проблеми та підтримає розвиток бізнесу.

1.5.2 Завдання Проекту.

Даний проект спрямований на створення інтегрованої веб-системи для онлайн-продажів, яка вирішуватиме наступні першочергові завдання:

- Повноцінно керувати каталогом товарів (додавання, редагування, архівація, категоризація);
- Обробляти замовлення від моменту їх створення до виконання або скасування;
- Реалізувати систему ролей (користувач, адміністратор) для контролю доступу;
- Створити модулі для аналізу продажів і активності користувачів;
- Забезпечити зручний і зрозумілий інтерфейс для всіх типів користувачів;
- Додати **функціонал блогу**, де можна публікувати новини, статті та огляди;

- Додати **систему коментарів**, щоб користувачі могли залишати думки під товарами та блогами.

1.5.3 Функціональні вимоги.

Функціональні вимоги визначають дії, які система має бути здатна виконувати. У таблиці нижче наведено функціональні системи.

Таблиця 1.1 Функціональні вимоги

Вимога	Опис вимоги
Авторизація та автентифікація	Вхід/реєстрація користувачів, розподіл ролей
Каталог продукції	Перегляд, фільтрація, сортування товарів
Кошик та оформлення замовлення	Додавання, редагування, оформлення покупок
Життєвий цикл замовлень	Зміна статусів: нове, обробка, виконано, скасовано
Адмін-панель	Керування товарами, замовленнями, користувачами, блогом
Блог	Публікація статей, оглядів від імені адміністратора
Коментарі	Можливість залишати коментарі під товарами та публікаціями блогу

1.5.4 Нефункціональні вимоги.

Система повинна мати зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко орієнтуватися та виконувати необхідні дії без тривалого навчання. Час відгуку системи має бути мінімальним – навіть при великому навантаженні сторінки та API-запити повинні завантажуватись не

довше ніж за 2 секунди. Особливу увагу приділено захисту даних: передбачені заходи проти поширених загроз, таких як SQL-ін’єкції, XSS та CSRF, а також використання токенів JWT для безпечної авторизації. Архітектура системи повинна бути гнучкою та підтримувати легке розширення – це дозволить у майбутньому додавати нові функції або змінювати існуючі без суттєвих змін у коді. Веб-додаток має коректно працювати у всіх популярних браузерах, включаючи Google Chrome, Firefox та Edge. Також система повинна бути стабільною і не “ламатися” навіть у випадках неправильних дій з боку користувача або нестабільного інтернет-з’єднання. Інтерфейс буде повністю українською мовою,

1.5.5 Вимоги до середовища розробки. Для розробки додатку планується використання таких інструментів:

- **Backend**: C# (.NET Core Web API)
- **Frontend**: Angular (TypeScript)
- **База Даних**: MSSQL
- **ORM**: Entity Framework Core
- **API Документація**: Postman
- **Тестування API**: Postman
- **Розробка**: Visual Studio 2022, Visual Studio Code
- **Контроль версій**: GitHub

1.6 Висновок по першому розділу

Представлений розділ "Системний аналіз предметної області" детально описує ринок чайної продукції та обґрунтовує необхідність створення сучасної веб-системи для онлайн-магазину.

У першому підрозділі, "Опис предметної області", розкрито особливості чайного ринку, що характеризується широким асортиментом продукції та

зростаючим попитом на якісну інформацію й персоналізований підхід. Визначено ключові аспекти для онлайн-магазину, такі як візитка товару, що включає його назву, походження, ціну, категорію, властивості та рекомендації щодо заварювання, а також якісні фотографії. Розглянуто ціноутворення, що залежить від різних факторів (типу, регіону, обробки, бренду, упаковки), та важливість реалізації гнучкої системи фасування з автоматичним перерахунком вартості. Підкреслено значення блогу у формуванні експертного іміджу, інформуванні користувачів та побудові спільноти, а також важливість механізму коментарів та відгуків для обміну досвідом та підвищення довіри до продукції. Описано функціональність кошика користувача, що враховує різну фасовку, миттєве оновлення вартості та збереження стану. Визначено необхідність зручного інструменту панелі адміністратора для ефективного управління товарами, замовленнями, блогом та коментарями. Насамкінець, виявлено основні проблеми предметної області, а саме складнощі сприйняття термінів, обмежену фільтрацію, відсутність взаємодії з відвідувачами, низький рівень персоналізації та обмежений контроль з боку адміністратора.

У підрозділі "Аналіз вимог до предметної області" було сформульовано функціональні та нефункціональні вимоги до майбутньої системи. Функціональні вимоги охоплюють реєстрацію та автентифікацію користувачів, перегляд і пошук товарів, роботу з кошиком, оформлення замовлень, функціонал панелі адміністратора, блогу та системи коментарів/відгуків. Нефункціональні вимоги акцентують увагу на продуктивності (швидкість завантаження сторінок та API-запитів), безпеці (захист від атак, авторизація через JWT), масштабованості, стилі та юзабіліті інтерфейсу. Також окреслено обмеження проєкту, пов'язані з його навчальним характером, зокрема відсутність мобільного застосунку, концептуальна оплата, одномовність, базовий функціонал адмін-панелі, локальне зберігання даних та недотримання всіх законодавчих вимог щодо захисту персональних даних.

Підрозділ "Моделювання предметної області" демонструє візуалізацію ключових процесів за допомогою UML-діаграм. Діаграма прецедентів (Use Case

Diagram) ілюструє взаємодію двох основних акторів ("Клієнт" та "Адміністратор") із системою, визначаючи їхні функції та сценарії використання. Діаграма послідовності (Sequence Diagram) показує послідовність взаємодії компонентів системи (Клієнт, Веб-інтерфейс, Сервіс продуктів, Сервіс кошика) під час таких процесів, як пошук, перегляд та оформлення замовлення. Діаграма активності (Activity Diagram) відображає основні бізнес-процеси, що виконуються користувачем та адміністратором, від входу в систему до оформлення замовлення та його підтвердження.

Наприкінці розділу, у "Постановці завдання", підсумовуються загальна характеристика задачі, основні завдання проєкту (керування каталогом, обробка замовлень, реалізація системи ролей, аналіз продажів, зручний інтерфейс, функціонал блогу та коментарів), а також функціональні та нефункціональні вимоги, наведені у табличній формі. Окрім цього, перелічено вимоги до середовища розробки (C# (.NET Core Web API), Angular, Ms SQL, Entity Framework Core, Postman, Visual Studio, GitHub), що забезпечить реалізацію проєкту відповідно до сучасних стандартів розробки. Цей розділ закладає міцний фундамент для подальшої розробки, надаючи комплексне розуміння предметної області, чітко визначені вимоги та візуалізовані моделі взаємодії.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних у вигляді ER-діаграми вирішує проблему розуміння та структурування інформації. Вона допомагає чітко побачити, які типи даних існують, які характеристики вони мають, і як вони пов'язані між собою. Це дозволяє уникнути плутанини при проєктуванні баз даних, забезпечити узгодженість інформації та спростити комунікацію між розробниками та замовниками, адже всі бачать єдину, наочну схему. Завдяки цьому можна виявити потенційні проблеми з даними ще до початку їх реалізації.

2.1.1 Основні сутності:

1. **AspNetUsers** - центральна сутність для зберігання даних користувачів системи. Містить особисту інформацію, контактні дані та дані для авторизації.
2. **Products** - товари, що представлені в системі, з описом, ціною, наявністю на складі.
3. **TeaDetail** - деталізована інформація про чай (особливості приготування, походження, інструкції зберігання).
4. **ProductPrice** - цінова політика для товарів, включаючи вагу та ціну.
5. **Categories** - категорії товарів для зручної навігації.
6. **Cart** і **CartItem** - кошик користувача та товари в кошику.
7. **Orders** і **OrderDetails** - замовлення користувачів та деталі кожного замовлення.
8. **Reviews** - відгуки користувачів про товари.
9. **Blogs** і **BlogComments** - блоги та коментарі до них.
10. **Photos** - зображення товарів.

11. **AspNet-сутності** - елементи системи авторизації та аутентифікації (ролі, токени, права).

2.1.2 Загальний опис взаємозв'язків між сутностями:

- **AspNetUsers → Orders** - один користувач може мати багато замовлень.
- **AspNetUsers → Cart** - один користувач має один кошик.
- **AspNetUsers → Reviews** - один користувач може залишити багато відгуків.
- **AspNetUsers → Blogs** - один користувач може створити багато блогів.
- **Products → TeaDetail** - один товар може мати детальну інформацію про чай.
- **Products → ProductPrice** - один товар може мати різні варіанти цін (наприклад, залежно від ваги).
- **Products → Categories** - товар належить до певної категорії.
- **Products → OrderDetails** - товар може бути в багатьох замовленнях.
- **Products → CartItem** - товар може бути в багатьох кошиках.
- **Products → Reviews** - товар може мати багато відгуків.
- **Products → Photos** - товар може мати багато фотографій.
- **Orders → OrderDetails** - одне замовлення містить багато товарів.
- **Cart → CartItem** - один кошик містить багато товарів.
- **Blogs → BlogComments** - один блог може мати багато коментарів.

2.1.3 Опис сутностей та їх атрибутів:

1. AspNetUsers: Id, UserName, NormalizedUserName, Email, NormalizedEmail, EmailConfirmed, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled

2. Products: Id, Name, Description, Stock, AvailableBuying

3. TeaDetail: Id, History, PreparationGuide, CultivationNotes, Origin, StorageInstructions

4. ProductPrice: Id, Price, WeightGrams, ProductId

5. Categories: Id, CategoryName, Deleted

6. Cart: Id, UserId

7. CartItem: Id, CartId, ProductId, Quantity, TotalPrice

8. Orders: Id, UserId, OrderDate, TotalAmount, ShippingAddress, PhoneNumber, Email, isConfirmed

9. OrderDetails: Id, OrderId, ProductId, Quantity, UnitPrice, Status

10. Reviews: Id, UserId, ProductId, Rating, Comment, isDeleted

11. Photos: Id, Image, ProductId

12. AspNetUserRoles: UserId, RoleId

13. AspNetRoles: Id, Name, NormalizedName

14. AspNetRoleClaims: Id, RoleId, ClaimType

15. AspNetUserClaims: Id, UserId, ClaimType

16. AspNetUserTokens: UserId, LoginProvider, Name

17. AspNetUserLogins: LoginProvider, ProviderKey, ProviderDisplayName

18. Blogs: Id, UserId, Title, Content

19. BlogComments: Id, BlogId, UserId, Comment

20. RefreshToken: Id, UserId, Token, ExpiresAt, isActive

2.1.4 Реалізація Soft Delete

У системі використовується метод "Soft Delete" (м'яке видалення), що означає, що дані фізично не видаляються з бази даних, а лише позначаються як неактивні або видалені. Це досягається через додавання спеціальних полів-індикаторів у таблицях:

- В таблицях є поле **isDeleted**, яке вказуэ на статус сутності.

Такий підхід має кілька переваг:

1. **Можливість відновлення даних** - інформація не втрачається назавжди і може бути відновлена.

2. **Аудит та історія** - зберігається історія всіх дій в системі для аналізу та моніторингу.

3. **Цілісність даних** - зберігаються зв'язки між сутностями, що запобігає порушенню цілісності даних.

4. **Статистика та аналітика** - доступ до повної інформації для формування звітів та аналізу.

Система також використовує поля як **CreatedAt** та **UpdatedAt** в різних таблицях для відстеження часу створення та останнього оновлення записів, що є частиною стратегії Soft Delete.

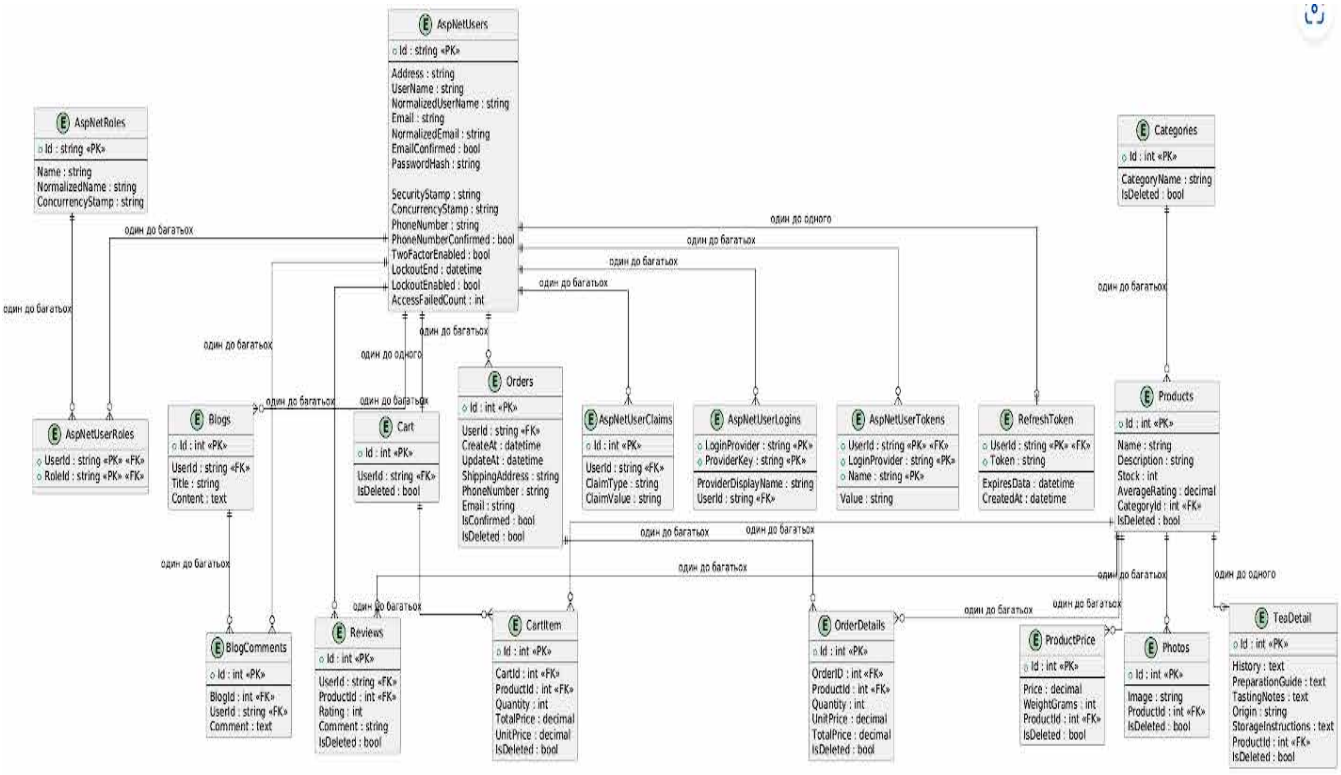

Рис. 2.1 – ER-діаграма (**Додаток А**)

2.2 Діаграма пакетів

У структурі веб-додатку важливою є чітка організація взаємодії між різними компонентами, що відповідають за окремі функціональні області. Діаграма пакетів демонструє логічне групування компонентів системи відповідно до їх призначення, а також описує основні зв’язки між ними. Вона

дозволяє на високому рівні зрозуміти, як побудована архітектура додатку та як взаємодіють різні його частини.

Загалом, архітектура веб-додатку складається з чотирьох основних пакетів: клієнтського, серверного, бази даних і файлового сховища.

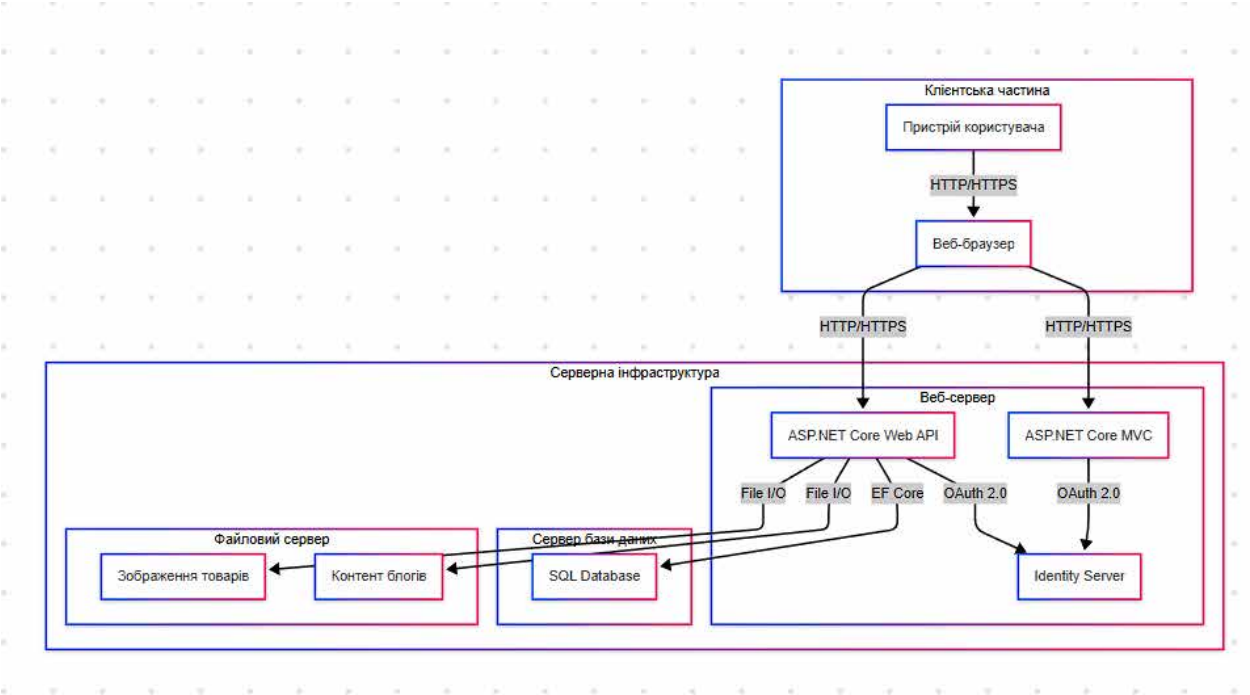

Рис. 2.2 – Діаграма пакетів

Клієнтський пакет включає в себе пристрій користувача та веб-браузер, який виступає як основний засіб доступу до функціональності додатку. Користувач взаємодіє з інтерфейсом через браузер, який у свою чергу формує HTTP або HTTPS-запити. Ці запити можуть бути адресовані як до Web API для отримання або обробки даних у форматі JSON, так і до MVC-компоненту, відповідального за генерацію HTML-сторінок, що динамічно рендеряться на стороні сервера.

Серверний пакет представлений трьома основними компонентами: Web API, MVC та сервером автентифікації Identity Server. Компонент Web API реалізований за допомогою ASP.NET Core і відповідає за прийом HTTP-запитів від клієнта, обробку логіки додатку, а також за взаємодію з базою даних та файловою системою. У випадку, якщо запит вимагає перевірки автентичності

або прав доступу, Web API надсилає запит до Identity Server, який реалізує протокол OAuth 2.0. Подібним чином, компонент MVC також звертається до Identity Server для автентифікації користувача перед рендерингом сторінки. Таким чином, серверна частина забезпечує як API-доступ до ресурсів, так і традиційне HTML-представлення.

Identity Server виконує критично важливу функцію управління безпекою, забезпечуючи централізовану автентифікацію та авторизацію користувачів. Він інтегрується з обома компонентами серверної частини - як із Web API, так і з MVC - що дозволяє уніфікувати підхід до перевірки прав доступу незалежно від типу запиту.

База даних, представлена SQL-сервером, є централізованим сховищем усіх структурованих даних системи: від інформації про користувачів до даних про товари, замовлення, коментарі, блоги тощо. Web API і MVC отримують доступ до бази даних за допомогою ORM-фреймворку Entity Framework Core, що дозволяє реалізовувати доступ до даних через об'єктно-орієнтовану модель без необхідності використання SQL-запитів.

Файлове сховище також є важливою складовою інфраструктури веб-додатку. Воно включає в себе два логічно розділені сховища - для зображень товарів і для контенту блогів. Обидва ці сховища використовуються Web API для здійснення файлових операцій: зчитування, запису, оновлення або видалення файлів. Наприклад, при додаванні нового товару API може обробити завантажене зображення, зберегти його у відповідному каталозі файлової системи та записати до бази даних шлях до файлу. Аналогічно, при створенні нової публікації в блозі, текстовий і мультимедійний контент також зберігається у файловій системі, а посилання на нього передаються через API до клієнта.

Щодо взаємодій між компонентами, варто відзначити кілька важливих напрямків: по-перше, веб-браузер формує запити або до MVC для отримання рендерених сторінок, або до Web API для отримання даних у форматі JSON. По-друге, як Web API, так і MVC використовують Identity Server для перевірки користувачів і авторизації доступу. По-третє, обидва серверні компоненти

активно працюють з базою даних для зберігання і отримання даних, а також з файловим сховищем для обробки файлів, що завантажуються або відображаються.

Таким чином, діаграма пакетів чітко демонструє логічну структуру веб-додатку, де кожен компонент виконує свою визначену роль і взаємодіє з іншими через стандартизовані інтерфейси. Такий підхід дозволяє досягти високого рівня масштабованості, підтримуваності та безпеки всієї системи.

2.3 Діаграма розгортання

Діаграма розгортання в UML використовується для моделювання фізичного розміщення компонентів програмного забезпечення на апаратних вузлах (серверах, комп'ютерах, пристроях) у розподіленій системі. Вона дозволяє візуалізувати архітектуру системи на етапі розгортання, показуючи, де саме будуть працювати різні частини програмного забезпечення.

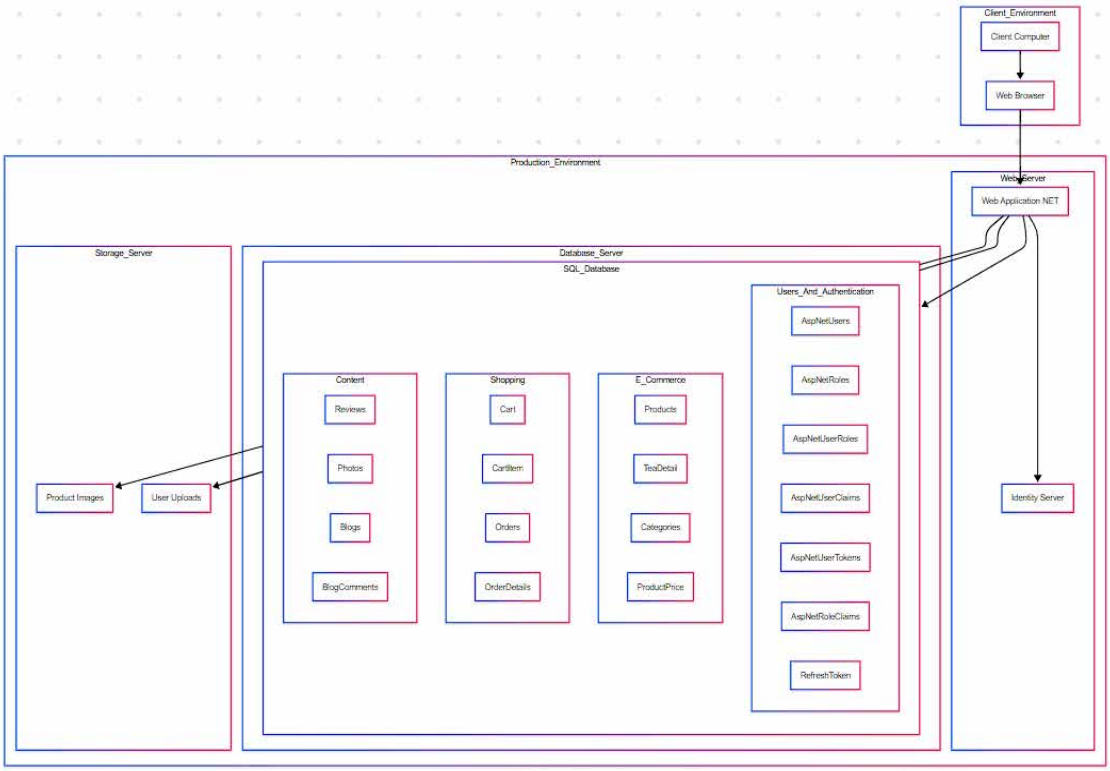

Рис. 2.3 – Діаграма розгортання

2.3.1 Опис представленої діаграми розгортання:

На цій діаграмі розгортання показана архітектура розподіленої системи онлайн-магазину (E-Commerce), що включає кілька ключових апаратних вузлів та програмних компонентів.

1. **Client Environment (Середовище клієнта):**
   - **Client Computer (Комп'ютер клієнта):**
   - **Web Browser (Веб-браузер):** Програмне забезпечення, що працює на комп'ютері клієнта і використовується для взаємодії з веб-сервером.
2. **Production Environment (Виробниче середовище):** Це основний контейнер, що представляє загальне середовище, де розгортаються всі сервери системи.
3. **Storage Server (Сервер сховища):**
   - Відповідає за зберігання статичних файлів, таких як зображення.
   - Містить компоненти: **Product Images (Зображення продуктів)** та **User Uploads (Завантаження користувачів)**, що свідчить про те, що користувачі також можуть завантажувати певний контент.
4. **Database Server (Сервер баз даних):**
   - Центральний вузол для зберігання всіх даних застосунку.
   - Виділено такі групи за функціональним призначенням:
     - **Content :** Включає таблиці для **Reviews** , **Photos** , **Blogs** та **BlogComments .**
     - **Shopping :** Містить таблиці для **Cart** , **CartItem**, **Orders** та **OrderDetails .**
     - **E_Commerce :** Включає таблиці для **Products** , **BoxDetail**, **Categories** та **ProductPrice**.
     - **Users_And_Authentication:** Містить таблиці для управління користувачами та їхніми правами, що характерно для ASP.NET Identity
5. **Web Server (Веб-сервер):**
   - Відповідає за обробку HTTP-запитів від веб-браузера.

- Веб-сервер взаємодіє з:
  - **Database Server:** Для отримання та збереження всіх даних, пов'язаних з контентом, покупками, та користувачами.
  - **Identity Server (Сервер ідентифікації):** Цей окремий компонент відповідає за автентифікацію та авторизацію користувачів, використовуючи протоколи OAuth 2.0 з JWT токенами.

2.5 Висновок по другому розділу

Другий розділ розкриває етапи проектування інформаційного та програмного забезпечення, надаючи комплексне бачення внутрішньої будови системи онлайн-магазину. Логічна модель даних (ER-діаграма) забезпечує чітке розуміння структури даних та їхніх взаємозв'язків, включаючи сучасний підхід до м'якого видалення. Діаграма пакетів деталізує логічну архітектуру та взаємодію між основними функціональними модулями (клієнтським, серверним, базою даних, файловим сховищем), підкреслюючи модульність та масштабованість. Нарешті, діаграма розгортання візуалізує фізичне розміщення компонентів на апаратних вузлах, що є незамінним для планування інфраструктури та управління експлуатаційними аспектами системи. Сукупність цих діаграм та їх описів демонструє продуману архітектуру, готову до подальшої реалізації.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглядається процес розробки інформаційної системи, починаючи з побудови архітектури програмного забезпечення, створення інформаційної моделі, застосування шаблонів проєктування та закінчуючи реалізацією механізмів безпеки. Основна мета розробки - забезпечити масштабованість, зручність супроводу, безпеку та чітке розмежування відповідальностей між компонентами системи. Для цього було використано багаторівневу архітектуру з виокремленням логіки у три основні проекти - API, Core та Infrastructure.

3.1 Загальна структура програмного забезпечення

3.1.1 Опис багаторівневої архітектури.

Розроблене програмне забезпечення базується на принципах **чистої архітектури (Clean Architecture)**, яка дозволяє чітко розмежувати бізнес-логіку, інфраструктурну реалізацію та зовнішній API-інтерфейс. Такий підхід покращує підтримку, тестованість та розширюваність додатку. Основними логічними рівнями є:

- **Core** - містить інтерфейси, DTO-моделі, специфікації:
    1. **Interfaces** - інтерфейси сервісів, репозиторіїв.
    2. **Models** - основні сутності, що відображають структуру предметної області.
    3. **Specifications** - шаблони специфікацій для фільтрації, сортування, пагінації.
    4. **Extensions** - розширення для реалізації технології SoftDelete.
- **Infrastructure** - відповідає за доступ до бази даних, реалізацію інтерфейсів, репозиторії та інші зовнішні сервіси.

1. **Context.cs** - реалізація контексту бази даних з використанням Entity Framework.
2. **Migrations/** - автоматично згенеровані міграції бази даних.
3. **Seeders/** - початкова ініціалізація обов’язкових або тестових даних.
4. **Repository.cs** - реалізація generic-репозиторію
5. **SpecificationEvaluator.cs** - застосування специфікацій до запитів.

- **API** - забезпечує обробку HTTP-запитів, автентифікацію, конфігурацію сервісів, роботу з AutoMapper, маршрутизацію тощо.

1. **Controllers/** - реалізація HTTP-контролерів для обробки запитів від користувача.
2. **Mapper/** - налаштування AutoMapper для перетворення між DTO та сутностями.
3. **appsettings.json** - конфігураційний файл, у якому задаються параметри підключення, JWT, шляхи тощо.
4. **Program.cs** - точка входу в програму, де виконується конфігурація залежностей, middleware та служб.

3.1.2 Причини вибору Clean Architecture

Clean Architecture, вона забезпечує чіткий поділ обов’язків між різними рівнями системи.

Основні причини вибору:

- **Незалежність від технологій.** Чиста архітектура дозволяє легко замінювати зовнішні бібліотеки або фреймворки (наприклад, замінити базу даних чи механізм) без суттєвих змін у бізнес-логіці.
- **Поділ на рівні.** Розмежування між доменною логікою (Core), інфраструктурою (Infrastructure) та API сприяє модульності й зменшує зв’язаність компонентів.

- **Спрощене тестування**. Завдяки використанню інтерфейсів та ізоляції бізнес-логіки, більшість компонентів системи можна буде протестувати незалежно від бази даних чи зовнішніх сервісів.
- **Принцип інверсії залежностей (Dependency Inversion Principle)**. Верхні рівні системи не залежать від нижчих. Замість цього, залежності інвертовано за допомогою інтерфейсів.
- **Масштабованість та гнучкість**. Додаток легко масштабувати, додаючи нові функціональні можливості, не зачіпаючи вже існуючий код.

Таким чином, Clean Architecture була обрана як найбільш відповідна для середніх і великих систем, що потребують високої гнучкості, підтримки та незалежності між логікою, зберіганням даних і представленням.

3.2 Інформаційне забезпечення

3.2.1 Опис моделей, реалізованих у Core

У бібліотеці **Core** зосереджено основні сутності предметної області, які використовуються для побудови бізнес-логіки. Кожна модель наслідує базову сутність BaseEntity, яка містить спільні атрибути, зокрема унікальний ідентифікатор Id.

Серед ключових моделей:

- User, Role – реалізують базову систему авторизації.
- Product, Category – описують товари й категорії.
- Order, OrderDetail – зберігають інформацію про замовлення й деталі замовлень.
- Review, Blog, BlogComment – дозволяють користувачам взаємодіяти з контентом через відгуки та коментарі.

Моделі визначають зв’язки між собою: один до багатьох (Category – Products, Order – OrderDetails) та багато до багатьох (User – Role через проміжну таблицю).

3.2.2 Причини вибору Entity Framework Core та MS SQL Server

Для роботи з базою даних було обрано **Entity Framework Core (EF Core)** як ORM-фреймворк. Основні причини:

- **Зв’язок із C#-об'єктами** – дозволяє працювати з базою даних через C# класи, без написання сирих SQL-запитів.
- **Міграції** – дають змогу створювати й оновлювати схему бази даних на основі змін у моделях.
- **Гнучкість і підтримка LINQ** – дає можливість формувати запити у вигляді C#-виразів.
- **Інтеграція зі специфікаціями** – EF Core добре працює із шаблоном Specification, що дозволяє ефективно реалізовувати фільтрацію, сортування тощо.

Як СУБД було обрано **MS SQL Server** з таких причин:

- **Надійність та продуктивність** – підходить для веб-додатків, де потрібна стабільна робота з великими обсягами даних.
- **Підтримка складних транзакцій та зв'язків** – реалізація складних зв’язків між таблицями (1:n, m:n).
- **Гнучкість у розгортанні** – має підтримку як локального, так і хмарного хостингу (наприклад, Azure SQL).

3.2.3 Generic Repository: реалізація, переваги і недоліки

У проєкті реалізований **Generic Repository** (Переглянити можна в дотатку Б) - це надає змогу використовувати базові методи до будь-якої сутності (CRUD-методів), використовуючи узагальнені типи. Це дозволяє не дублювати код для кожної моделі, а використовувати одну логіку.

Переваги Generic Repository:

- Універсальність і повторне використання коду.
- Проста реалізація і тестування.
- Відповідає принципу DRY (Don't Repeat Yourself).

Недоліки:

- Обмеження при складних запитах (наприклад, фільтрація, сортування, включення пов'язаних сутностей).
- Потреба дублювати методи або переписувати репозиторій для більш специфічної логіки.
- Порушення принципу **Single Responsibility** при надмірній генералізації.

Саме тому для подолання недоліків Generic Repository у проекті використовується **патерн Specification**.

3.2.4 Specification Pattern: суть, реалізація та переваги

Specification Pattern дозволяє створювати окремі специфікації (умови), які можна багаторазово застосовувати для фільтрації, сортування, включення пов’язаних даних та іншої логіки в запитах.

У проекті реалізовані:

- базовий клас BaseSpecification<T>, який визначає:

1. фільтраційні критерії (Criteria);
2. включення пов’язаних таблиць (Includes);
3. сортування (OrderBy, OrderByDescending);
4. пагінацію (Skip, Take);

- клас SpecificationEvaluator, який бере IQueryable<T> і застосовує до нього всі правила зі специфікації.

Переваги Specification Pattern:

- Чиста, структурована логіка запитів.
- Відокремлення умов від реалізації репозиторію.
- Легке повторне використання та тестування умов запиту.

Використання цього підходу дозволяє не розширювати репозиторій новими методами для кожного запиту, а замість цього створювати специфікації у вигляді окремих класів (наприклад, ProductWithFiltersSpecification або OrdersWithDetailsSpecification), що робить код гнучким і масштабованим.

3.3 Функціональні можливості системи

Розроблена система є серверною частиною (REST API) для онлайн-магазину, що дозволяє реалізувати повний цикл взаємодії користувача з інтернет-магазином. Основні функціональні можливості охоплюють як базову поведінку користувача, так і адміністративні функції. Система побудована за принципами модульності, розширюваності та безпеки. Нижче наведено основні функціональні блоки:

3.4.1 Автентифікація та авторизація

У веб-додатку реалізована система автентифікації та авторизації з використанням технологій ASP.NET Core Identity та JWT (JSON Web Token)**.** Мета реалізації - забезпечити безпечний доступ до захищених ресурсів та гнучке управління ролями користувачів.

JWT та його переваги

Для автентифікації обрано JWT-токени, оскільки цей підхід дозволяє створити систему без збереження сесій на сервері. Після успішного входу користувач отримує токен доступу, який включає в себе інформацію про користувача, його унікальний ідентифікатор, email, список ролей та інші claims.

Основні переваги JWT:

- **Простота передачі** - токен передається в заголовку запиту (Authorization: Bearer ...).
- **Незалежність від сесій** - не вимагає збереження інформації на сервері.
- **Швидкість перевірки** - використання цифрового підпису дозволяє верифікувати токен без доступу до БД.

Refresh Token

Оскільки токен доступу має обмежений термін дії (наприклад, 15–30 хвилин), реалізовано також механізм оновлення токенів за допомогою refresh token.

Це унікальний довготривалий токен, який зберігається на стороні клієнта в cookies і в базі даних - прив’язаний до користувача. Він використовується для:

- автоматичного оновлення токена доступу без повторного входу;
- підтримки безперервного сеансу користувача;
- додаткового контролю безпеки (наприклад, можливість відкликати refresh token при виході користувача).

Реєстрація та вхід

При реєстрації нового користувача в систему створюється обліковий запис з валідацією унікальності email, після чого пароль хешується з використанням механізмів, наданих Identity. Також автоматично додається роль за замовчуванням (наприклад, User).

При вході перевіряється правильність облікових даних, і у випадку успішної перевірки користувач отримує:

- JWT-токен для доступу до захищених ресурсів;
- Refresh токен для продовження сесії без повторного логіну.

Ролі та обмеження доступу

У системі реалізовано дві основні ролі**:**

- **User** - звичайний користувач з обмеженим доступом (перегляд товарів, оформлення замовлень, написання відгуків).
- **Admin** - адміністратор з повним доступом до керування продуктами, категоріями, замовленнями тощо.

Ролі зберігаються у відповідній таблиці бази даних (AspNetRoles) і пов’язані з користувачем через таблицю AspNetUserRoles. Для обмеження доступу до певних endpoint’ів використовуються атрибути авторизації**.** Це дозволяє обмежити доступ лише для авторизованих користувачів з відповідною роллю.

Фронтенд: збереження та передача токенів

На стороні клієнта (Angular-додаток) реалізовано перехоплювач запитів (**interceptor**), який автоматично додає токен у заголовки кожного HTTP-запиту

до захищених ресурсів. Це дозволяє забезпечити централізовану авторизацію без необхідності додавати токен вручну в кожному запиті

3.4.2 Використання AutoMapper у проєкті

У процесі розробки веб-додатку виникає необхідність працювати з різними представленнями даних: внутрішніми моделями (сутностями бази даних) та зовнішніми об'єктами, які передаються через API (DTO - Data Transfer Object). Щоб уникнути ручного копіювання полів між цими об'єктами, у проєкті використовується бібліотека AutoMapper.

Проблема, яку вирішує AutoMapper

У класичних CRUD-операціях часто виникає потреба:

- Перетворювати сутності БД у DTO для відправки на клієнт (наприклад, не включати зайві поля, або змінювати назви властивостей);
- Перетворювати DTO в сутності для збереження в базу (наприклад, при створенні нового продукту або редагуванні замовлення).

Ручне копіювання полів в обох напрямках є:

- **Трудомістким** - особливо якщо моделей багато і вони змінюються;
- **Схильним до помилок** - можна випадково не скопіювати якесь поле або переплутати типи;
- **Незручним для підтримки** - при зміні моделі потрібно оновлювати всі мапінги вручну.

Переваги використання AutoMapper

AutoMapper автоматизує процес перетворення об'єктів і значно знижує кількість шаблонного коду. Основні його переваги:

- **Зменшення дублювання коду** - немає потреби вручну писати копіювання кожного поля.
- **Централізоване управління мапінгами** - всі правила перетворення налаштовуються в одному місці (у профілях).
- **Гнучкість** - AutoMapper підтримує налаштування мапінгу, включаючи ігнорування полів, перейменування, умовне мапінгування тощо.

- **Покращена підтримуваність коду** - при зміні моделі достатньо оновити мапінг один раз, замість внесення змін у десятки місць.

Роль у проєкті

У цьому веб-додатку AutoMapper використовується:

- Для перетворення сутностей Product, Order, Review, User тощо в DTO, які повертаються з API;
- Для обробки вхідних DTO при створенні або редагуванні сутностей;
- Для забезпечення чистоти архітектури, розділяючи модель бази даних та модель представлення на клієнті.

Наприклад, модель Product може містити багато службової інформації, яка не потрібна користувачу (наприклад, дата створення, ідентифікатори зв'язків), тому для передачі на клієнт формується окрема DTO-модель, що включає лише необхідні поля (назва, опис, ціна, категорія тощо).

3.4.3 Сервісний рівень

У рамках проєкту реалізовано чітке розділення логіки за допомогою сервісного рівня. Такий підхід дозволяє винести бізнес-логіку з контролерів та репозиторіїв, зберігаючи чисту архітектуру та дотримуючись принципу єдиної відповідальності. Розміщення логіки в окремих сервісах спрощує підтримку та масштабування застосунку, а також значно покращує тестованість. Кожен сервіс відповідає за певну частину функціональності.

У системі реалізовано декілька основних сервісів: ProductService, OrderService. PhotoService, RatingService, UserService. ProductService відповідає за роботу з товарами, включаючи пошук, фільтрацію, сортування та реалізацію пагінації. OrderService реалізує логіку створення замовлення, перевірку авторизації користувача та збереження деталей замовлення. UserService

обробляє запити, пов’язані з автентифікацією та авторизацією користувачів: генерацію JWT і рефреш-токенів, вхід та реєстрацію, перевірку даних користувача та його ролей.

Усі сервіси працюють із DTO (об’єктами передачі даних), які дозволяють передавати лише необхідну інформацію між шарами системи. Це допомагає

захистити внутрішню структуру моделі бази даних та спростити форматування відповіді для клієнта. Додатково, всі методи в сервісах реалізовано з використанням асинхронності. Завдяки async/await система здатна обробляти запити до бази даних без блокування потоків, що підвищує загальну продуктивність застосунку та дозволяє краще масштабувати його при високому навантаженні.

Таким чином, сервісний рівень відіграє ключову роль у побудові логіки застосунку, забезпечуючи чисту структуру, спрощене тестування, гнучкість розвитку та ефективну взаємодію між контролерами, репозиторіями та базою даних.

3.4.4 Контролери та endpoints

У системі контролери відіграють ключову роль в обробці HTTP-запитів, отриманих від клієнтів, і є посередниками між клієнтською частиною та бізнес-логікою застосунку. Кожен контролер відповідає за певний аспект функціональності та обслуговує конкретний набір ресурсів, надаючи доступ до них через відповідні endpoints. В основному, контролери реалізують базові CRUD-операції (створення, читання, оновлення, видалення), а також додаткові специфічні дії, які залежать від типу сутності.

ProductController відповідає за керування продуктами в магазині. Він дозволяє отримувати список продуктів із підтримкою пагінації та фільтрації, отримувати продукт за ідентифікатором, додавати нові продукти, прикріплювати до них фотографії, додавати ціни та деталі чаю. Контролер інтегрується зі службою фотографій для збереження та отримання зображень товарів, а також працює з репозиторіями для взаємодії з базою даних.

UserController відповідає за роботу з обліковими записами користувачів. Він обробляє реєстрацію, авторизацію (включаючи видачу access- і refresh-токенів), оновлення профілю, завантаження фотографій користувачів, додавання ролей, оновлення токенів, а також надає захищені маршрути, доступні лише для авторизованих користувачів. Через роботу з JWT (JSON Web Token) забезпечується автентифікація та контроль доступу.

Інші контролери, такі як BlogController**,** BlogCommentController**,** CartController**,** CategorieController**,** OrderController**,** OrderDetailController**,** ReviewController, реалізують подібну структуру: кожен обробляє запити, пов’язані зі своєю сутністю. Наприклад, ReviewController дозволяє користувачам залишати відгуки на товари, а OrderController керує створенням та переглядом замовлень. CategorieController забезпечує логіку для роботи з категоріями товарів, CartController – із кошиком користувача, а OrderDetailController – із деталями окремих замовлень.

Загалом, архітектура контролерів організована у відповідності до принципів REST, що забезпечує простоту, передбачуваність та масштабованість API. Всі контролери успадковують ControllerBase, що дозволяє використовувати атрибути маршрутизації, моделі валідації запитів та HTTP-статуси у відповідях.

3.4.5 Контролери та endpoints

Для розробки клієнтської частини онлайн-магазину було обрано фреймворк **Angular**. Основною причиною такого вибору стала потреба у створенні динамічного, масштабованого та зручного для підтримки односторінкового застосунку (SPA). Angular надає потужний інструментарій для побудови компонентної архітектури, що дозволяє чітко структурувати код, спростити розробку повторно використовуваних елементів, реалізувати ефективну маршрутизацію між сторінками та забезпечити гнучку роботу з формами, подіями та асинхронними запитами.

Клієнтська частина реалізована як набір **сторінок** і **перевикористовуваних компонентів**, згрупованих за відповідними каталогами.

До **основних сторінкових компонентів** належать:

- home – головна сторінка з коротким оглядом магазину;
- login – форма входу до системи для авторизованих користувачів;
- register – форма реєстрації нового користувача;
- product-page – детальна сторінка окремого товару;

- shopping-cart – кошик користувача з можливістю змінювати кількість товарів і оформлювати замовлення;
- teas – сторінка з переліком чаїв, можливістю сортування, фільтрації та пагінації;
- user-page – особистий кабінет користувача;
- news – відображення блогу або новин.

Крім сторінок, реалізовано низку **перевикористовуваних компонентів**, які винесено у спільну папку shared:

- categorie – компонент для відображення списку або фільтрації за категоріями;
- footer – футер сайту;
- nav-bar – навігаційна панель, що включає авторизаційні кнопки, посилання та корзину;
- product-list – компонент для виведення списку товарів у вигляді карток;
- app.component – головний компонент, що є контейнером усього застосунку.

Для обробки логіки, роботи з API та управління станом реалізовано низку **Angular сервісів**:

- product.service – відповідає за отримання списку товарів, фільтрацію, пагінацію, пошук;
- product-page.service – використовується для отримання детальної інформації про окремий товар;
- cart.service – керує додаванням, видаленням товарів у кошику, обрахунком вартості та збереженням даних;
- UserServices – робота з даними користувача, наприклад, отримання профілю;
- auth.interceptor – Angular Interceptor, що автоматично додає токен авторизації до HTTP-запитів, а також може обробляти помилки або перенаправляти неавторизованих користувачів.

Архітектура застосунку базується на чіткому розділенні відповідальностей. Дані, які надходять із серверної частини через REST API, спочатку обробляються у відповідних сервісах, де виконуються запити, обробка відповіді та можливе кешування. Далі компоненти, які відображають дані на екрані, отримують цю інформацію через залежності від сервісів. Таким чином, компоненти зосереджені виключно на логіці представлення, що робить код більш зрозумілим, розширюваним і тестованим.

Таким чином, Angular дозволив реалізувати сучасний, зручний у користуванні та легко підтримуваний клієнтський інтерфейс, який забезпечує інтерактивну взаємодію з користувачем, гнучку структуру коду та ефективну інтеграцію з серверною частиною застосунку.

3.5 Висновок по третьому розділу

Розробка інформаційного та програмного забезпечення для онлайн-магазину базується на принципах чистої архітектури, що забезпечить модульність, незалежність від технологій шляхом шляхом поділу системи на рівні Core (бізнес-логіка), Infrastructure (доступ до даних) та API (зовнішній інтерфейс). Описано аспекти інформаційного забезпечення, що включають моделі даних, вибір Entity Framework Core та MS SQL Server, а також реалізацію шаблонів Generic Repository та Specification Pattern для гнучких запитів. Деталізовано функціональні можливості системи: безпечна автентифікація та авторизація з використанням ASP.NET Core Identity та JWT-токенів, застосування AutoMapper для автоматизації перетворення об'єктів, структура сервісного рівня для виділення бізнес-логіки, а також функціонал контролерів та їхніх Endpoints. Розроблено клієнтську частину на Angular, яка використовує компонентну архітектуру та взаємодію з бекендом. Таким чином, застосовані архітектурні підходи та технології забезпечують гнучкість, безпеку та масштабованість веб-додатку.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною розробки, спрямованою на виявлення дефектів та забезпечення високої якості продукту, а також відповідності реалізованого функціоналу вимогам та надійній роботі в реальному середовищі. Для досягнення цієї мети було застосовано декілька типів тестування. Модульне тестування проводилось вручну для перевірки коректності окремих функцій та методів API. Інтеграційне тестування перевіряло взаємодію між клієнтською (Angular) та серверною (ASP.NET Core API) частинами, зокрема, процеси авторизації, роботи з токенами та обробки помилок. Системне тестування охоплювало перевірку всієї системи як єдиного цілого, включаючи взаємодію бази даних, серверної логіки, API та користувацького інтерфейсу.

Тестування було спрямоване на перевірку відповідності системи вимогам кінцевого користувача шляхом моделювання типових сценаріїв використання, таких як створення, редагування та коментування блогів, а також перегляд їх списку.

4.1.1 Практичне тестування вебзастосунку

Середовище тестування:

- Операційна система: Windows 10
- Сервер: .NET SDK 8.0, SQL Server 2022
- Клієнт: Angular 16

4.1.2 Практичне тестування вебзастосунку

1. **Аутентифікація** - Перевірити, як система обробляє реєстрацію користувача, вхід до системи, та створення/видачу токенів доступу. Особлива увага приділяється валідації полів (email, пароль), повідомленням про помилки та реакції клієнта.

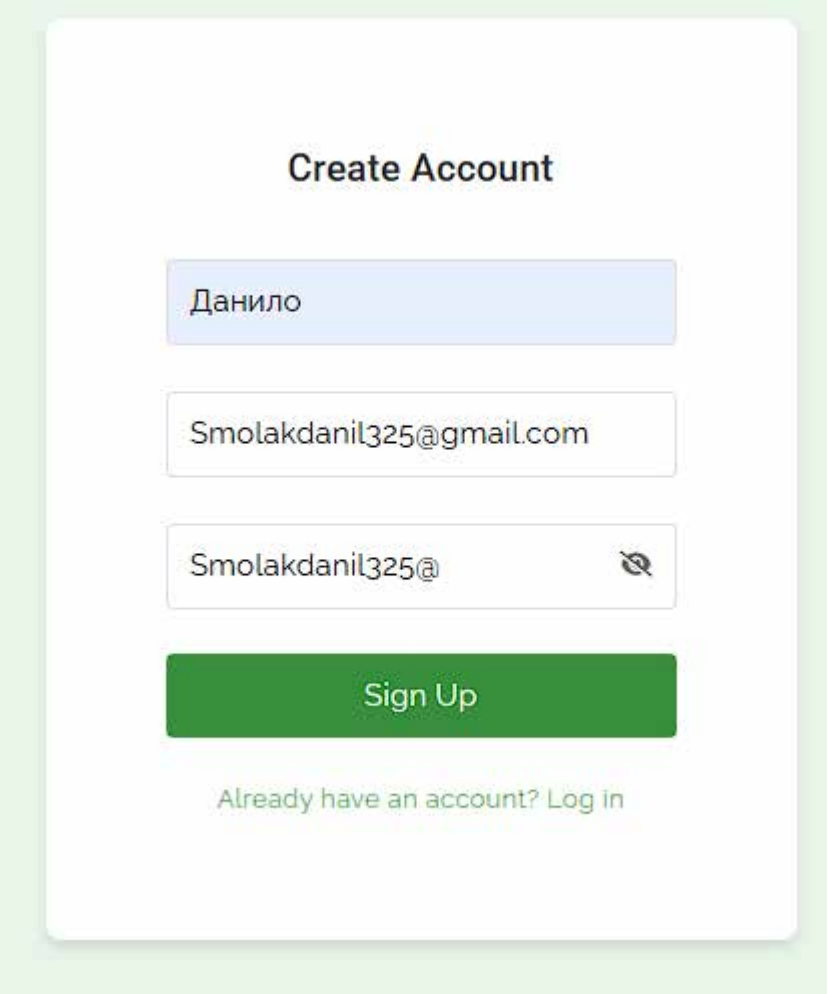

Рис. 4.1 – Реєстрація

Програма має правильно виконувати:

- Email має формат email@example.com. Якщо структура неправильна - з'являється повідомлення про помилку.
- Пароль має бути складним: мінімум 6 символів, латинські літери, цифри, спеціальні символи.

При успішному створенні нового користувача переводить на сторінку входу в систему.

2. Вхід у систему

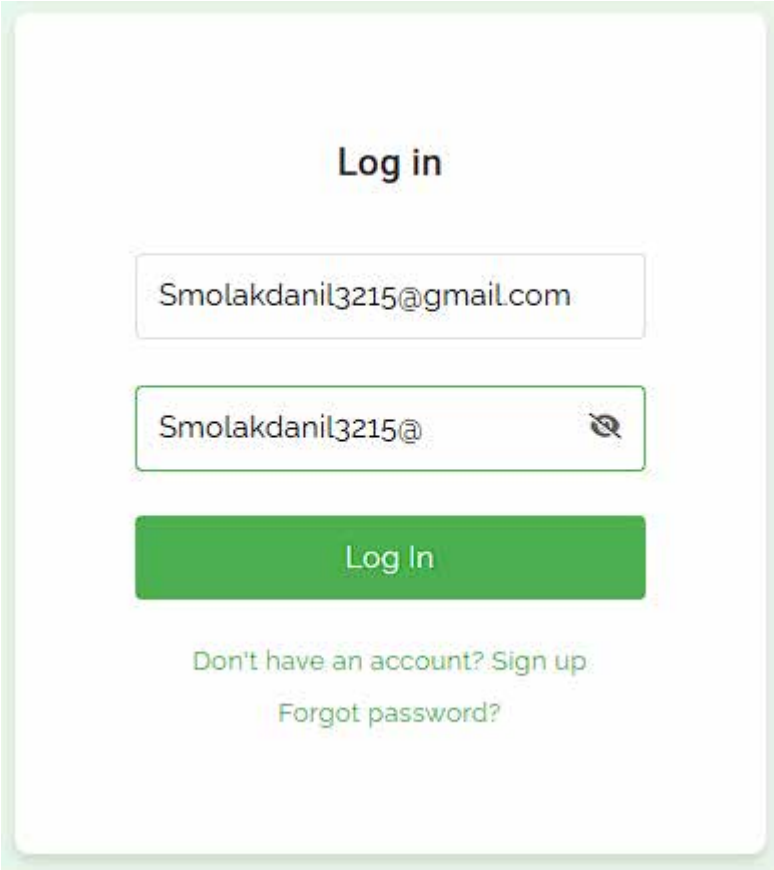

Рис. 4.2 – Вхід в систему

- Успішний вхід - користувач отримає токен доступу і буде перенаправлений на головну сторінку
- `Невірні облікові дані → повертається статус 401 Unauthorized, і користувачу виводиться повідомлення "Invalid email or password".`
- Користувачу надається Jwt токен та Refresh токен.

При успішному вході можна побачити профіль користувача.

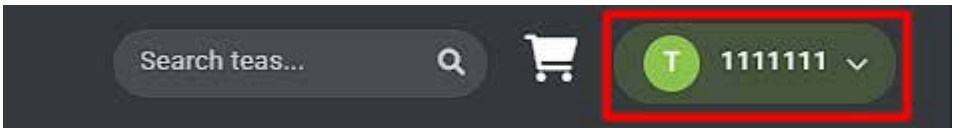

Рис. 4.3 – Профіль користувача.

3. Коментарі – Перевірити функціональність додавання та перегляду коментарів до товарів, а також перевірку доступу до цієї функції лише для автентифікованих користувачів. Перегляд коментарів для всіх користувачів (у тому числі неавторизованих):

- Доступний список коментарів під кожним товаром.
- Коментарі містять текст відгуку, ім’я користувача або псевдонім, а також оцінку.
- Коментарі відображаються в хронологічному порядку.

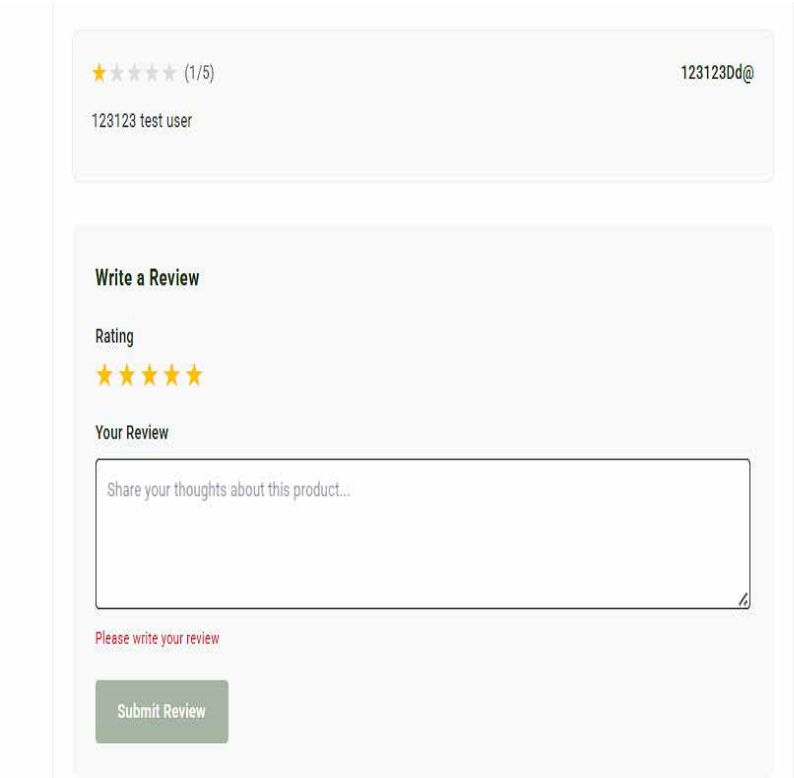

Рис. 4.4 – Відгуки товару

4. Кошик

Тестування функціональності кошика підтвердило його коректну роботу, що є критично важливим для онлайн-магазину. На початковому етапі, коли кошик порожній, система інформує про це користувача, відображаючи повідомлення "Your cart is empty. Continue shopping" (Рис. 1.3 - Порожній кошик). При цьому загальна кількість товарів та загальна вартість коректно дорівнюють нулю.

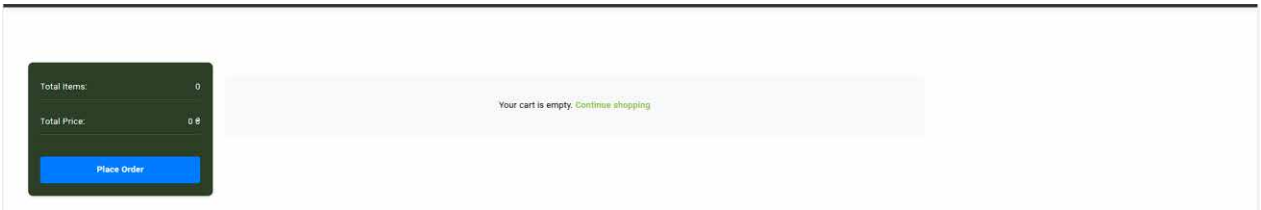

Рис. 4.5 – Порожній кошик

Після додавання товарів до кошика (Рис. 1.3 – Один товар але за різною ціною), система правильно відображає додані позиції, їхню кількість, ціну за одиницю та сумарну вартість для кожної позиції. Загальна кількість товарів та сумарна вартість кошика також оновлюються в реальному часі. Була перевірена можливість зміни кількості товарів у кошику, що призводить до автоматичного перерахунку загальної суми. Функціонал видалення товарів з кошика також працює коректно, відображаючи зменшення кількості та суми.

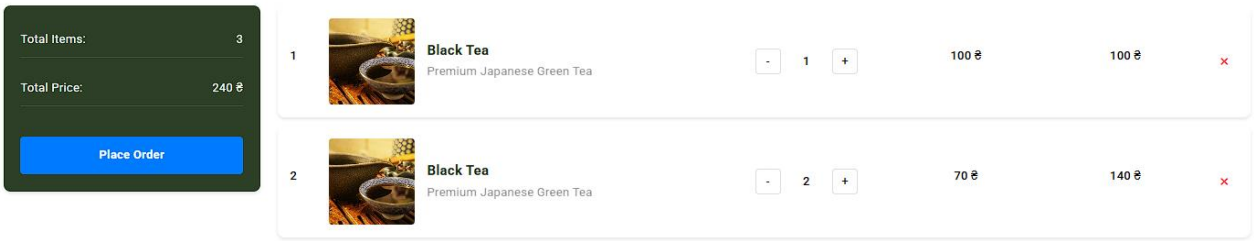

Рис. 1.3 – Один товар але за різною ціною через грами

5. Оформлення замовлення

Тестування оформлення замовлення підтвердило можливість користувача перейти від наповненого кошика до етапу введення інформації, необхідної для доставки. На сторінці оформлення замовлення (Рис. 1.3 - Оформлення замовлення,) користувачеві пропонується ввести дані для доставки, такі як адреса, номер телефону та електронна пошта. Система забезпечує відображення раніше доданих товарів у кошику з їхньою кількістю та загальною ціною, що дозволяє користувачеві перевірити своє замовлення перед підтвердженням. Після успішного введення даних та підтвердження замовлення система коректно обробляє його, що є завершальним етапом покупки.

Рис. 4.6 – Оформлення замовлення

6. Блог

На сторінці "Blog Posts" (Рис. 1.3 - Блог) відображається список публікацій у блозі. Кожна публікація містить заголовок, короткий опис та ім'я автора. Кнопка "Create New Post", доступна для авторизованих користувачів, що дозволяє створювати нові записи.

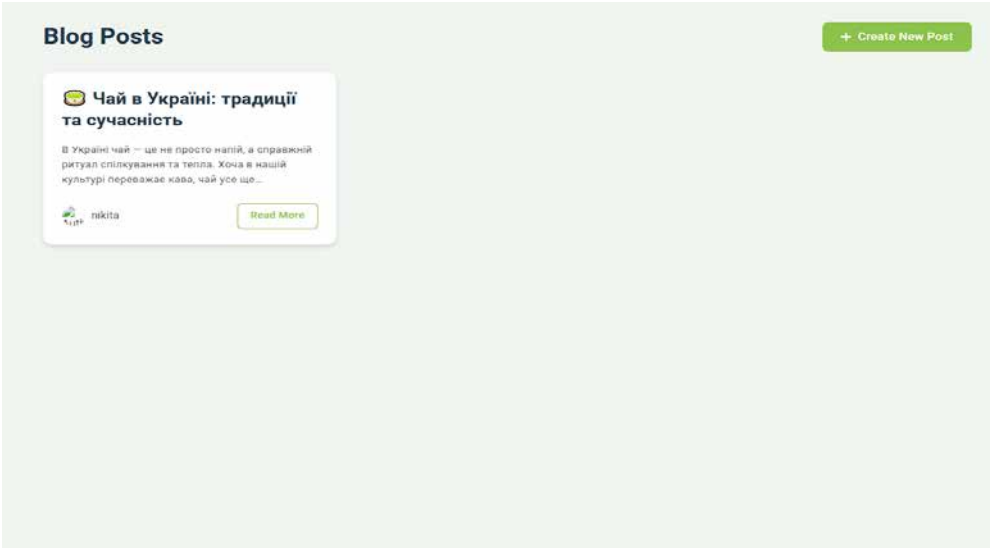

Рис. 4.7 – Блог

При перегляді окремого блогу (Рис. 1.3 - Перегляд блогу) відображається повний текст публікації, її заголовок та ім'я автора. Важливою перевіркою було підтвердження того, що користувач, який створив блог, має можливість його редагувати та видаляти. Ці кнопки відображаються лише для автора публікації, що забезпечує належне розмежування прав доступу та контроль над власним контентом. Користувачі, які не є авторами, можуть лише переглядати вміст блогу та писати відкуги.

Рис. 4.8 – Перегляд блогу

4.2 Вимоги до апаратного та програмного забезпечення

Для успішного функціонування розробленого вебзастосунку необхідне відповідне апаратне та програмне середовище. Вимоги до нього поділяються на серверні та клієнтські.

Серверні вимоги:

- **Операційна система:** Рекомендовано використання операційної системи Windows 10 або новішої версії, що забезпечить сумісність та оптимальну продуктивність для розробленої серверної частини на .NET.
- **Платформа розробки та виконання:** Необхідно мати встановлений .NET SDK версії 8.0 або новішої. Це забезпечить коректне компілювання, запуск та виконання серверної частини API.
- **Система управління базами даних (СУБД):** Для зберігання та управління даними застосунку потрібен SQL Server 2022 або сумісна версія. Він забезпечує надійність, продуктивність та підтримку всіх необхідних реляційних операцій.

Клієнтські вимоги:

- **Операційна система:** Застосунок розроблено як веб-додаток, тому він сумісний з більшістю сучасних операційних систем, таких як Windows, macOS, Linux.
- **Веб-браузер:** Для доступу до функціоналу магазину необхідний будь-який сучасний веб-браузер (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) останньої стабільної версії.
- **Платформа розробки (для розробки/компіляції клієнтської частини):** Для роботи з клієнтським кодом, розробленим на Angular, необхідні такі компоненти:
  - **Angular CLI:** Версія 19.2.1 або новіша, яка є інструментом командного рядка для ініціалізації, розробки та підтримки Angular-застосунків.
  - **Node.js:** Версія 18.19.0 або новіша, що є середовищем виконання JavaScript і необхідний для роботи Angular CLI та керування залежностями.
  - **npm (Node Package Manager):** Версія 10.2.3 або новіша, використовується для встановлення та управління пакетами JavaScript.

Ці вимоги до апаратного та програмного забезпечення гарантують стабільну роботу системи як на етапі розробки, так і після її розгортання, забезпечуючи сумісність між усіма компонентами системи.

4.3 Установлення програми

Для успішного запуску та функціонування вебзастосунку необхідно послідовно виконати кроки з налаштування середовища розробки та розгортання серверної і клієнтської частин.

1. **Встановлення Node.js та Angular CLI**

- **Node Version Manager (NVM) та Node.js:** Слід завантажити та встановити nvm-setup.exe з офіційного репозиторію NVM на GitHub. Після встановлення NVM, використовуючи командний рядок (CMD або PowerShell), виконати команди nvm install 18.10.0 та nvm use 18.10.0 для інсталяції та активації необхідної версії Node.js.
- **Angular CLI:** Глобальне встановлення Angular CLI здійснюється командою npm install -g @angular/cli@15.2.0.У випадку необхідності оновлення до найновішої версії використовується npm install -g @angular/cli@latest.

2. **Встановлення .NET SDK**

- Необхідно переконатися, що на системі встановлено .NET SDK версії 8.0.0. Завантажити його можна з офіційного сайту Microsoft. Перевірка встановленої версії виконується командою dotnet --version.

3. **Налаштування серверної частини**

- **Клонування репозиторію:** Початковий код проєкту необхідно склонувати з GitHub за допомогою команди git clone https://github.com/DanylSmolyak/TeaShopByDan та перейти до кореневої папки проєкту командою cd TeaShopByDan.
- **Налаштування підключення до бази даних:** У файлі appsettings.Development.json, розташованому в корені проєкту, потрібно замінити значення DefaultConnection на актуальний рядок підключення до локального або віддаленого SQL Server.

- **Застосування міграцій:** Для створення та оновлення схеми бази даних відповідно до моделей проєкту необхідно виконати команду dotnet ef database update.

4. **Запуск клієнтської частини (Angular)**

- **Перехід до папки клієнта:** У командному рядку потрібно перейти до папки `client` за допомогою `cd client`.
- **Встановлення залежностей:** Усі необхідні для Angular-додатку залежності встановлюються командою `npm install`.
- **Запуск сервера розробки:** Клієнтська частина запускається командою `ng serve`, після чого додаток буде доступний за адресою `http://localhost:4200`.

5. **Запуск серверної частини (.NET API)**

- **Запуск Backend:** У кореневій папці проєкту, після налаштування бази даних, серверна частина запускається командою `dotnet run`

4.4 Висновок по четвертого розділу

"Впровадження та експлуатації системи" підсумовує етапи підготовки розробленого вебзастосунку до його використання. Комплексне тестування системи, що включало модульне, інтеграційне, системне та приймальне тестування дозволило перевірити коректність роботи окремих компонентів, взаємодію між клієнтською (Angular) та серверною (ASP.NET Core API) частинами, функціональність всієї системи як єдиного цілого та її відповідність вимогам кінцевого користувача. Практичне тестування підтвердило надійну роботу механізмів аутентифікації (реєстрація, вхід, видача токенів), функціоналу коментарів (додавання, перегляд), коректну обробку операцій з кошиком (додавання, зміна кількості, видалення, перерахунок суми) та успішне оформлення замовлень. Також було підтверджено функціональність блогу, включаючи створення, перегляд та можливість редагування/видалення публікацій автором. Визначено чіткі вимоги до апаратного та програмного

забезпечення, що охоплюють операційні системи, версії .NET SDK та SQL Server для серверної частини, а також необхідні версії Angular CLI, Node.js та npm для клієнтської розробки. Це забезпечує сумісність та стабільну роботу застосунку. Надано детальну інструкцію з встановлення програми, яка включає кроки з налаштування Node.js та Angular CLI, встановлення .NET SDK, конфігурації серверної частини (клонування репозиторію, налаштування підключення до бази даних, застосування міграцій) та запуску клієнтської і серверної частин. Таким чином, розділ демонструє готовність системи до розгортання та експлуатації, підтверджену ретельним тестуванням та чітко визначеними вимогами до середовища та процесу встановлення.

ВИСНОВОК

У даній дипломній роботі було успішно розроблено та впроваджено вебзастосунок інтернет-магазину для продажу чайної продукції з інтегрованим функціоналом блогу. Проєкт пройшов усі необхідні етапи від системного аналізу до тестування та підготовки до експлуатації.

Було детально проаналізовано ринок чайної продукції, визначено ключові особливості та потреби онлайн-магазину, включаючи гнучке ціноутворення, систему фасування, важливість блогу та механізму коментарів. Сформульовані функціональні та нефункціональні вимоги до майбутньої системи, а також окреслені обмеження проєкту. Візуалізація ключових процесів за допомогою діаграм дозволила чітко представити взаємодію користувачів із системою та її бізнес-логіку, а постановка завдання визначила основні цілі та вимоги до середовища розробки.

Внутрішня архітектура системи була ретельно спроектована, що включає логічну модель даних із застосуванням сучасного підходу до м'якого видалення. Діаграма пакетів деталізувала логічну архітектуру, підкресливши модульність та масштабованість, а діаграма розгортання візуалізувала фізичне розміщення компонентів.

Практична реалізація проєкту базувалася на принципах чистої архітектури, забезпечуючи модульність, незалежність та легкість тестування шляхом поділу системи на рівні бізнес-логіки, доступу до даних та зовнішнього інтерфейсу. Описано аспекти інформаційного забезпечення, вибір інструментів для роботи з базою даних, а також реалізацію шаблонів для гнучких запитів. Функціональні можливості системи включають безпечну автентифікацію та авторизацію, застосування засобів для автоматизації перетворення об'єктів, структуру сервісного рівня та функціонал контролерів. Клієнтська частина розроблена з використанням компонентної архітектури та взаємодії з бекендом.

Проведено комплексне тестування, яке включало різні типи перевірок, підтвердивши коректність роботи всіх ключових функціональних модулів:

аутентифікації, коментування, кошика, оформлення замовлень та блогу. Визначені чіткі вимоги до апаратного та програмного забезпечення. Надано детальну інструкцію з встановлення програми, що охоплює налаштування середовища розробки та розгортання всіх її частин.

Таким чином, у ході виконання дипломної роботи було створено повноцінний, функціональний та безпечний вебзастосунок інтернет-магазину чайної продукції. Застосовані архітектурні підходи забезпечують гнучкість, масштабованість та легкість у супроводженні системи, що робить її готовою до подальшого розвитку та інтеграції з додатковими сервісами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Angular. *Angular. The web development framework for building the perfect app*. URL: https://angular.io/ (дата звернення: 03.05.2025).
2. Голуб Б. Л. Методичний посібник до вивчення дисципліни «Програмування та алгоритмічні мови». Навчальне видання. Голуб Б. Л., Щукайло Є. М. Київ: Видавничий центр НАУ, 2003. 64 с.
3. Офіційний сайт Microsoft .NET. *Welcome to .NET. The developer platform*. URL: https://dotnet.microsoft.com/ (дата звернення: 23.05.2025).
4. Паттерн "Специфікація" в .NET. *Medium*. URL: https://medium.com/some-article-about-specification-pattern (дата публікації: 01.01.2023; дата звернення: 23.05.2025).
5. Ролі та автентифікація в ASP.NET Core Identity. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity (дата оновлення: 15.03.2024; дата звернення: 23.05.2025).
6. Бондар В. М. *Теоретичні основи інформатики*: підручник. Київ: Кондор, 2019. 450 с.
7. Коваленко І. П. *Бази даних та інформаційні системи*: навчальний посібник. Львів: Новий Світ, 2021. 380 с.
8. Мороз О. В. *Веб-технології та веб-дизайн*: курс лекцій. Одеса: ОНПУ, 2022. 210 с.
9. Степаненко Г. С. *Програмування мовою C#*: методичні вказівки. Запоріжжя: ЗНУ, 2020. 150 с.
10. Іванов П. Р. *Основи розробки програмного забезпечення*: навчальний посібник. Харків: ХНУРЕ, 2023. 290 с.
11. ДСТУ 2850-94. *Програмні засоби оброблення інформації. Терміни та визначення*. Київ: Держстандарт України, 1994. 20 с.
12. Кравчук А. В. *Проектування та розробка RESTful API*: практичний посібник. Дніпро: Ліра, 2024. 270 с.

13. Офіційний сайт Node.js. *Node.js*. URL: https://nodejs.org/en (дата звернення: 23.05.2025).

14. Офіційний сайт Visual Studio. *Visual Studio. The IDE for every developer*. URL: https://visualstudio.microsoft.com/ (дата звернення: 23.05.2025).