НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"02" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему
Програмне забезпечення з розробки веб-додатку для магазину послуг з організації заходів
Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

_____ Степанов О.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Опря Артем Олександрович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
Голуб Б.Л.

"16" грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Опрі Артему Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи Програмне забезпечення з розробки веб-додатку для магазину послуг з організації заходів
затверджена наказом ректора НУБіП України від "16" грудня 2024р. №2249С

Термін подання завершеної роботи на кафедру 2025. 05. 25
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
опис програмного забезпечення

Перелік питань, які потрібно розробити:

- системний аналіз предметної області;
- проєктування інформаційного та програмного забезпечення;
- розробка інформаційного та програмного забезпечення;
- рекомендації щодо впровадження та експлуатації системи.

Дата видачі завдання "16" грудня 2025 р.
Керівник бакалаврської кваліфікаційної роботи

_____ _____ Степанов О.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Опря А.О.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

MVC – Model-View-Controller

MVT – Model-View-Template

ORM – Object-oriented model

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

SSL – Secure Sockets Layer

HTTPS – Hypertext transfer protocol secure

OMT – Object Modeling Technique

UML – Unified Modeling Language

RUP – Rational Unified Process

XML – eXtensible Markup Language

XMI – XML Metadata Interchange

OOSE – Object-Oriented Software Engineering

ER-model – Entity-Relationship model

JSON – JavaScript Object Notation

SQL – Structured Query Language

CTE – Common Table Expression

НФ – Нормальна форма

ВСТУП

Організація свят – це не просто бізнес. Це про емоції, враження, кольори, деталі, що роблять моменти незабутніми. Весільні букети, арки з живих квітів, повітряні кулі на день народження чи оформлення залів для корпоративів – усе це створюється руками людей, які добре розуміють, як виглядає щастя. Проте за кожною гарною фотосесією або розкішно оформленим банкетом ховається рутина: прийом замовлень, уточнення побажань клієнтів, погодження вартості, облік матеріалів, планування доставки й монтажу. Ці процеси часто залишаються «за лаштунками», проте саме вони визначають якість і своєчасність послуги.

На жаль, багато компаній, які працюють у сфері організації святкових заходів, змушені справлятися з цими завданнями вручну або за допомогою тимчасових рішень, таких як Google-форми, соціальні мережі, блокноти, телефонні дзвінки. Це не лише незручно, але й підвищує ризик помилок, втрати клієнтських даних або дублювання замовлень. У якийсь момент бізнес стикається з обмеженням, яке не вирішується новими квітами чи дизайнерськими ідеями. Тут на допомогу приходить автоматизована система як для менеджерів, так і для клієнтів.

Метою цієї дипломної роботи є розробка сучасного, адаптивного, веб-орієнтованого програмного продукту для магазину послуг з організації святкових заходів. Система має забезпечувати зручний інтерфейс для клієнтів, можливість перегляду каталогу послуг і товарів (букети, декорації, подарункові набори тощо), оформлення замовлень, адміністрування контенту з боку менеджерів та автоматизоване збереження даних у базі.

Щоб створити ефективну веб-систему, насамперед потрібно детально проаналізувати предметну область: зрозуміти потреби користувачів, бізнес-процеси компанії, типи послуг і сценарії взаємодії з клієнтами. Далі слід вивчити існуючі аналоги, порівняти їхній функціонал, інтерфейси, переваги й недоліки — це дозволить уникнути шаблонних рішень і зробити продукт зручнішим та

актуальнішим. На основі отриманої інформації потрібно змоделювати предметну область за допомогою UML-діаграм, зокрема створити діаграми варіантів використання, послідовності, компонентів, щоб візуально представити майбутню архітектуру системи та логіку її роботи.

Наступним кроком є аналіз і вибір технологій, які найкраще підходять для реалізації проєкту — як на рівні бекенду, так і фронтенду. Потрібно спроєктувати базу даних, розробити програмну частину, реалізувати функціональність для споживачів і менеджерів, передбачити особисті кабінети, форму замовлення, каталог товарів і послуг, можливість керування контентом. Після цього має бути проведене всебічне тестування програмного продукту, щоб переконатися в його стабільності, безпечності та відповідності вимогам.

Для реалізації поставленої мети було обрано стек сучасних веб-технологій. Основою проєкту є високорівневий Python-фреймворк Django, який дозволяє швидко створювати масштабовані веб-застосунки з чіткою архітектурою MVC. Для зберігання даних використовується надійна СУБД PostgreSQL, що добре поєднується з Django ORM. Для розробки було використано середовище PyCharm, яке забезпечує зручну роботу з кодом, відлагодженням і базами даних.

За матеріалами дипломної роботи було підготовлено наукові тези, які були подані та прийняті до публікації у збірнику наукових праць VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2025», що відбулася 24 квітня 2025 року. У публікації були висвітлені основні етапи створення інформаційної системи, обґрунтовано вибір технологій, окреслено архітектурні рішення та ключові функціональні можливості розробленого програмного продукту. [13]

До складу дипломної роботи входять наступні розділи: вступ, чотири розділ в складі основної частини, висновок, список джерел та додатки. У першому розділі «Системний аналіз предметної області» розглянуто специфіку діяльності магазину послуг, вимоги до функціональності майбутньої системи, проведено моделювання бізнес-процесів і огляд існуючих аналогів. У другому

розділі «Проєктування інформаційного та програмного забезпечення» побудовано логічну модель даних, діаграми класів, пакетів, компонентів, що відображають архітектуру проєкту. Третій розділ присвячено безпосередній розробці програмного продукту, а саме описано вибір інструментів, створення бази даних, алгоритмізацію основних модулів і реалізацію їх у коді. Четвертий розділ містить рекомендації щодо впровадження: описано процес тестування системи, наведено вимоги до середовища запуску та структуру інсталяційного пакету.

У пояснювальній записці використано 24 джерела літератури, додатки містять фрагменти основної логіки веб-додатку.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сфера організації святкових заходів – це один із напрямів сервісного бізнесу, що охоплює широкий спектр послуг, пов’язаних з підготовкою та проведенням урочистостей. До таких послуг відносяться створення квіткових композицій, оформлення залів і локацій, підбір святкових аксесуарів, оренда декорацій, створення фотозон, доставка подарунків та багато іншого. Основна мета такої діяльності – забезпечити клієнту максимально комфортну організацію свята, враховуючи його індивідуальні побажання, бюджет, стиль і формат заходу.

Клієнтами подібних сервісів можуть бути як приватні особи (молодята, іменинники, батьки дітей, які святкують хрестини чи випускний), так і організації (школи, фірми, салони краси, ресторани тощо), які потребують декорування приміщень або зовнішніх локацій. Як правило, робота з клієнтом передбачає кілька етапів: ознайомлення з каталогом товарів і послуг, вибір необхідного набору, оформлення замовлення, погодження деталей, передплата (часткова або повна), виконання замовлення і надання послуги.

Окрему роль у роботі таких магазинів відіграє асортимент, як правило, це не лише фізичні товари (букети, композиції, прикраси), але й послуги (доставка, монтаж, індивідуальне оформлення, консультації, виїзне обслуговування тощо). Асортимент постійно оновлюється, залежно від сезонів, трендів, запасів матеріалів та наявності персоналу.

Наразі більшість малих та середніх компаній, що працюють у цій сфері, використовують неструктуровані засоби обліку та комунікації: обговорення деталей через месенджери, публікації в Instagram чи Facebook замість повноцінного сайту, ручне ведення таблиць із замовленнями. Це призводить до неефективного використання робочого часу, ймовірних помилок, складності в обліку товарів і наданих послуг, відсутності зручної аналітики й труднощів зі збереженням історії взаємодії з клієнтами.

Автоматизація цього процесу може значно покращити ситуацію. Веб-додаток, створений для потреб магазину послуг з організації заходів, дозволяє централізовано керувати асортиментом, обробляти замовлення, зберігати клієнтські дані, відслідковувати статуси виконання робіт і взаємодіяти з клієнтами через зручний інтерфейс. З точки зору користувача, це означає швидкий доступ до пропозицій магазину, можливість створення персонального замовлення без телефонних дзвінків, перегляд попередніх замовлень, а також зручність у спілкуванні з менеджером.

Предметна область, в якій реалізується програмна система, охоплює бізнес-процеси, пов'язані з наданням послуг у сфері організації заходів, включаючи облік товарів, управління замовленнями, клієнтську взаємодію та забезпечення інформаційної підтримки. Розробка веб-додатку є відповіддю на потреби такої сфери в цифрових інструментах, які здатні оптимізувати роботу, покращити якість обслуговування та підтримати розвиток бізнесу.

1.2 Аналіз вимог до програмної системи

1.2.1 Бізнес-вимоги

Збільшення доходів через розширення ринку

Мета: проект має на меті залучення нових сегментів клієнтів шляхом виходу на нові ринки та регіони.

Завдяки онлайн-платформі можна обслуговувати клієнтів без географічних обмежень, що дозволяє збільшити охоплення нових ринків. Інвестиції у SEO, рекламні кампанії та соціальні мережі сприятимуть підвищенню впізнаваності бренду та збільшенню обсягу продажів.

Збільшення середнього чека через крос-продажі

Мета: збільшити середній чек за рахунок рекомендацій супутніх товарів.

Впровадження системи рекомендацій, яка буде пропонувати клієнтам додаткові товари під час оформлення замовлення. Це сприятиме збільшенню обсягу продажів кожного окремого замовлення.

Оптимізація витрат

Мета: знизити операційні витрати компанії, пов'язані з управлінням замовленнями та обслуговуванням клієнтів.

Автоматизація процесів обробки замовлень, відстеження доставок і спілкування з клієнтами через інтегровані сервіси, наприклад, автоматизовані email-повідомлення, дозволить скоротити витрати на операційний персонал, а також зменшить кількість ручних помилок, що призводять до додаткових витрат.

Підвищення конверсії через покращення користувацького досвіду

Мета: збільшити конверсію відвідувачів сайту в покупців.

Оптимізація веб-додатку для зручності користувачів, скорочення часу на пошук і покупку товарів, швидкий та безпроблемний процес оплати сприятимуть зростанню кількості завершених покупок. Впровадження зручних мобільних версій та інтеграція з соціальними мережами також дозволить легше залучати та конвертувати мобільних користувачів.

1.2.2 Функціональні вимоги

У межах розробки веб-додатку для магазину послуг з організації свят особлива увага приділяється зручності користувача як звичайного споживача, так і менеджера. Ключовий елемент системи – це каталог послуг і товарів. Споживач має змогу переглядати пропозиції в зручному онлайн-форматі: із фото, описами, інформацією про наявність та ціни. Для того щоб користувач швидше знаходив потрібне, передбачено фільтрацію за категоріями та ціновим діапазоном. Також реалізований пошук за ключовими словами, наприклад, якщо потрібно швидко знайти «букет нареченої» або «повітряні кулі».

Ще одна базова функція – кошик. Користувач може додати туди вибрані товари або послуги, змінювати їх кількість, при необхідності видаляти позиції, а також бачити загальну вартість у режимі реального часу. Це дозволяє сформувати замовлення з урахуванням бюджету та потреб.

Після того як усе вибрано, користувач переходить до оформлення замовлення. Тут він вказує свої контактні дані, обирає спосіб доставки і

визначається з методом оплати: або одразу онлайн, або при отриманні. Після підтвердження замовлення користувач отримує квитанцію на email, це зручно для збереження підтвердження та подальшої комунікації.

Щоб спростити роботу постійним клієнтам, реалізовано особистий кабінет. У ньому можна переглядати історію попередніх замовлень, оновлювати особисту інформацію (контактні дані, адреси доставки), а також у будь-який момент змінити пароль або оновити профіль. Реєстрація та авторизація реалізовані через email.

Для менеджерів передбачена окрема панель керування. Тут можна додавати нові товари, редагувати їх опис чи фото, змінювати ціни, керувати наявністю. Також у панелі доступна аналітика по замовленнях, можливість переглядати кожне замовлення, змінювати його статус («очікує на обробку», «у роботі», «відправлено», «завершено»). Менеджерський персонал має змогу гнучко керувати категоріями та керувати базою клієнтів.

1.2.3 Нефункціональні вимоги

Окрім функціональності, яку безпосередньо бачить користувач, важливою складовою є ті властивості системи, що забезпечують її надійну, безпечну та зручну роботу в довгостроковій перспективі. У цьому проєкті було визначено кілька основних нефункціональних вимог, які впливають на якість та ефективність роботи всієї системи.

Продуктивність. Веб-додаток повинен працювати швидко – незалежно від того, скільки людей заходить на сайт одночасно. Особливо в пікові періоди, коли всі купують подарунки або оформлюють замовлення на весілля, не можна допустити зупинки роботи сайту. Кожна сторінка має відкриватися не довше 2–3 секунд, інакше можна втратити потенційного користувача. Цього вдається досягти завдяки оптимізації запитів до бази даних, кешуванню частини інформації та мінімізації кількості зовнішніх ресурсів.

Масштабованість. Бізнес може рости, спочатку це буде 20 замовлень на місяць, потім 200, а можливо і 2000. Архітектура системи повинна бути готова

до цього. Тобто, у разі збільшення кількості споживачів, повинна бути можливість швидко під'єднати нові сервери або переналаштувати балансувальник навантаження, щоб усе працювало так само стабільно, як і на початку.

Безпека. Оскільки в системі обробляються персональні дані користувачів, адреси доставки, електронні скриньки, то важливо захистити всі ці дані. Весь обмін даними між клієнтом і сервером повинен бути зашифрований за допомогою SSL (HTTPS). Паролі зберігаються тільки у вигляді хешів, що унеможливлює їх витік у випадку компрометації бази.

Надійність. Будь-який онлайн-сервіс має працювати безперебійно. У разі виникнення помилки або збою серверного обладнання дані не повинні зникнути. Саме тому передбачено регулярне резервне копіювання (хоча б раз на добу), а також механізми відновлення з резервної копії. Це дозволить швидко повернути систему до робочого стану з мінімальними втратами, навіть у разі серйозних технічних проблем.

Юзабіліті (Зручність використання). Інтерфейс користувача також має бути зручним, не лише красивим, але й логічним. Це не просто «щоб було зрозуміло», а про те, щоб споживач міг оформити замовлення в кілька кліків, не розгубився серед десятків форм і не мусив шукати кнопку «Підтвердити» півгодини. Сайт адаптований під різні типи пристроїв: від великих моніторів до маленьких смартфонів. Це важливо, адже більшість користувачів заходить із телефону, особливо в ситуаціях, коли терміново потрібно щось замовити.

1.3 Моделювання предметної області

Перш ніж перейти безпосередньо до технічної реалізації веб-додатку, важливо сформувати чітке уявлення про логіку його роботи, структуру об’єктів та взаємозв’язки між ними. Для цього проводиться моделювання предметної області, що дозволяє представити ключові елементи майбутньої системи у

вигляді схем, діаграм та моделей, що значно спрощує розуміння її внутрішнього устрою як для розробника, так і для всіх зацікавлених сторін.

У цьому розділі розглянуто основні способи формального опису предметної області, зокрема з використанням методів та інструментів UML.

Модель даних зазвичай має форму діаграми, яка супроводжується текстовим описом. Вона візуально представляє елементи, важливі для бізнесу (наприклад, люди, місця, речі та бізнес-транзакції), атрибути, пов'язані з цими елементами, та важливі взаємозв'язки між ними. [9]

UML забезпечує стандартну нотацію для багатьох типів діаграм, які можна умовно розділити на 3 основні групи: діаграми поведінки, діаграми взаємодії та структурні діаграми. [16]

Логічні та фізичні діаграми "сутність-зв'язок" використовуються для реалізації реляційної бази даних, тоді як логічна або фізична діаграма класів використовується для підтримки об'єктно-орієнтованої розробки програмного забезпечення. [9]

Моделювання

Важливо розрізняти UML-модель і набір діаграм системи. Діаграма – це часткове графічне представлення моделі системи. Набір діаграм не обов'язково повинен повністю покривати модель, і видалення діаграми не змінює модель. Модель може також містити документацію, яка керує елементами моделі та діаграмами. [16]

Така діаграма дозволяє візуалізувати, які саме дії користувачі можуть виконувати в межах системи, як вони взаємодіють із різними компонентами, а також які ролі бере на себе кожен тип користувача. Завдяки цьому розробник краще розуміє, якими мають бути функціональні сценарії, а замовник може перевірити, чи реалізовано очікувану логіку.

На рис. 1 представлена діаграма варіантів використання, яка описує основні варіанти взаємодії користувачів із системою магазину послуг з

організації заходів. У системі чітко розрізняються дві ключові ролі: Споживач та Менеджер. Кожен з них має власні задачі й набір доступних дій.

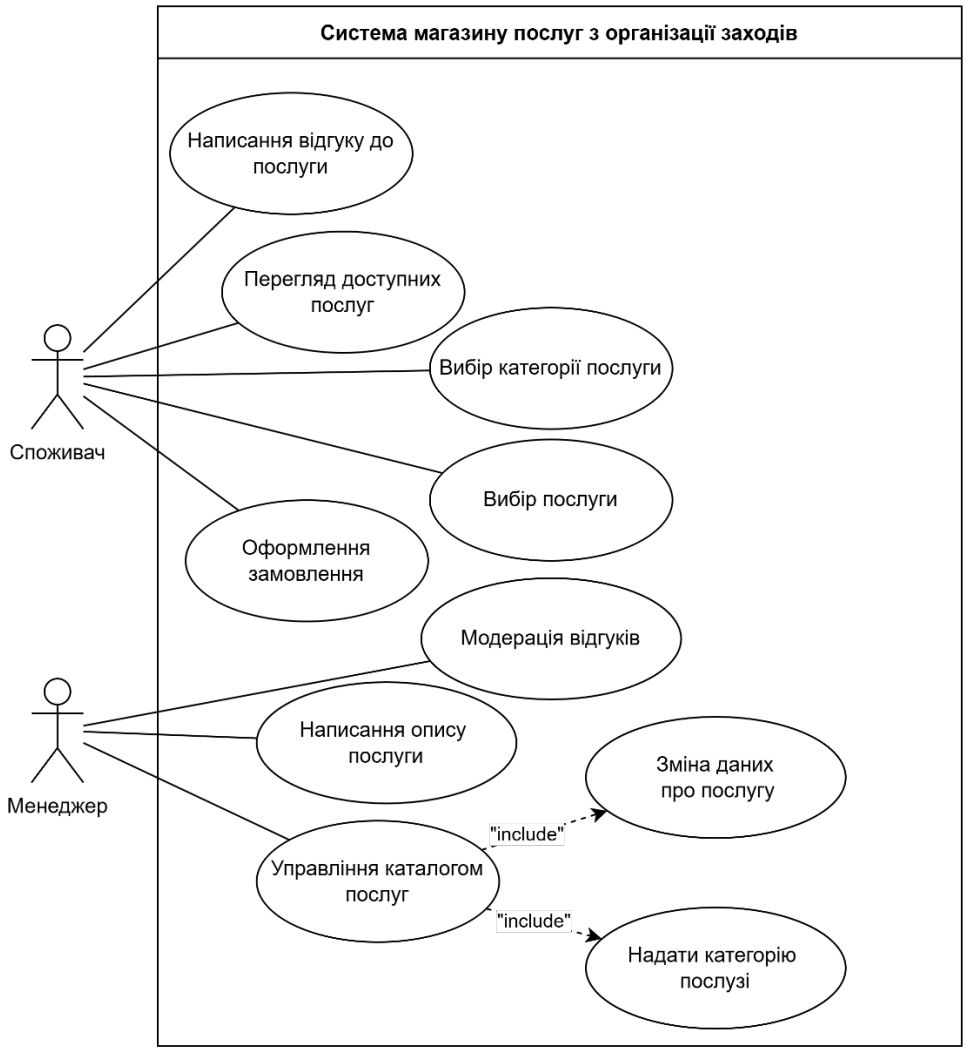


Рис 1. Діаграма прецедентів

Для звичайного споживача основним сценарієм є взаємодія з каталогом послуг. Все починається з перегляду доступних пропозицій користувач може ознайомитись із наявними варіантами оформлення, прочитати описи, переглянути фотографії та відфільтрувати послуги за певними критеріями (категоріями, ціною, наявністю). Щоб спростити пошук необхідної послуги, система надає можливість виконати пошук за ключовими словами.

Після того, як обрано категорію, система відображає перелік відповідних послуг. Далі користувач переходить до наступного етапу – оформлення

замовлення. Цей процес передбачає заповнення контактних даних, вибору способу доставки та оплати.

Ще однією важливою функцією для користувача є можливість залишити відгук про послугу. Цей елемент додає елемент взаємодії між клієнтами та допомагає новим відвідувачам формувати враження про якість сервісу. Водночас система передбачає механізм модерації відгуків, який реалізується на боці менеджера.

Менеджер – це окремий тип користувача, який має доступ до функціоналу, недоступного звичайним клієнтам. Його основне завдання — підтримка структури каталогу послуг у актуальному стані.

На діаграмі передбачено і взаємозв'язки між окремими випадками використання. Наприклад, при управлінні каталогом послуг менеджер може як змінювати опис чи ціну послуги, так і призначати її до певної категорії. Умовні позначки типу «include» вказують на те, що певна дія є частиною іншого процесу й виконується щоразу під час нього.

Для глибшого розуміння логіки взаємодії між окремими об'єктами системи, а також для уточнення послідовності виконання дій під час обробки замовлень, створено діаграму послідовності (Sequence diagram). Цей тип UML-діаграм дозволяє детально простежити часову послідовність обміну повідомленнями між сутностями, які беруть участь у виконанні певного сценарію. На відміну від діаграми варіантів використання, яка фокусується на ролях та їх можливостях, діаграма послідовності занурює нас у конкретну логіку реалізації функціоналу на прикладі послідовних викликів операцій.

Діаграма послідовності дозволяє візуалізувати процеси взаємодії між користувачами системи (споживачами та менеджерами) та компонентами системи (каталогом послуг та зошитом замовлень). Це допомагає краще зрозуміти логіку роботи системи та виявити потенційні проблеми на ранніх етапах проектування.

На представленій діаграмі послідовності (рис. 2) відображено взаємодію між чотирма основними сутностями: Споживачем, Каталогом послуг, Зошитом замовлень та Менеджером. Діаграма демонструє повний цикл процесу замовлення товару чи послуги.

Процес починається з того, що споживач переглядає доступні послуги в каталозі. Каталог послуг відображає послуги, доступні для замовлення. Далі споживач обирає категорію послуг, після чого система відображає послуги в обраній категорії. Споживач обирає конкретну послугу, і система надає детальну інформацію про неї.

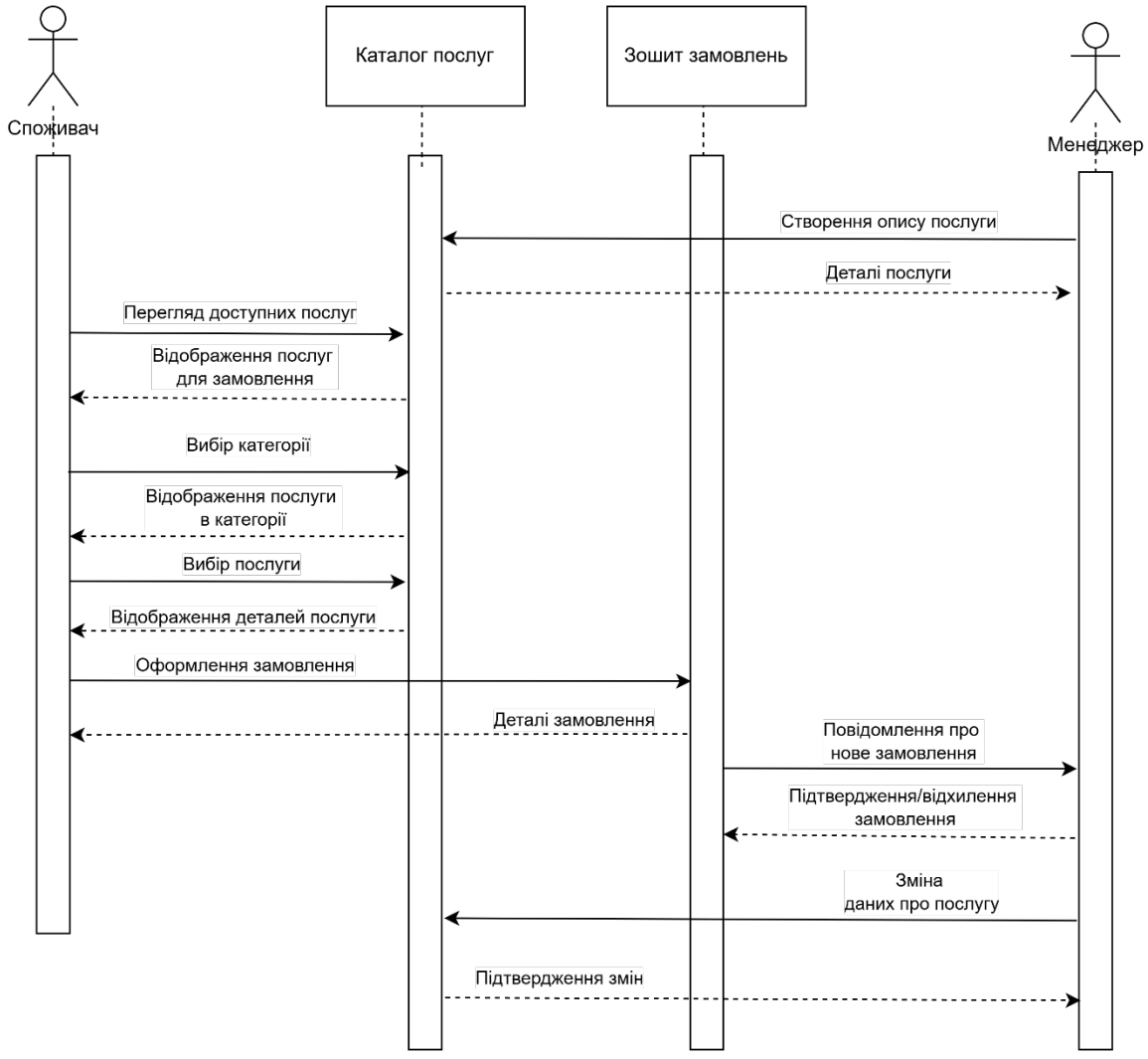


Рис 2. Діаграма послідовності

Після ознайомлення з деталями послуги споживач оформлює замовлення, яке передається до зошита замовлень. Зошит замовлень обробляє інформацію та надсилає деталі замовлення назад споживачу для підтвердження. Одночасно з цим зошит замовлень надсилає повідомлення менеджеру про нове замовлення.

Менеджер має можливість створювати описи послуг та змінювати дані про послуги в каталозі. Після внесення змін до каталогу надсилається підтвердження змін до менеджера.

1.4 Огляд інформаційних джерел та існуючих рішень

У цьому розділі проведено аналіз існуючих веб-рішень у сфері флористики та організації заходів. Розглянуто функціональні можливості, переваги та недоліки кожного з них, що дозволить визначити ключові аспекти для розробки власного веб-додатку.

FloralStore – це інтернет-магазин штучних квітів, який пропонує широкий асортимент декоративних рослин та композицій. Сайт орієнтований на клієнтів, які шукають довговічні альтернативи живим квітам для декорування інтер'єрів, проведення свят та інших подій.

Переваги:

- Спеціалізація на штучних квітах, що забезпечує довговічність продукції та відсутність проблем з алергенами
- Детальний опис переваг продукції з акцентом на якість, реалістичність та гіпоалергенність
- Широкий спектр застосування продукції (інтер'єри, одяг, зачіски, ритуальні та національні костюми, свята)

Недоліки:

- Обмеженість асортименту лише штучними квітами, що не задовольняє потреби клієнтів, які шукають живі рослини

- Відсутність комплексних послуг з організації заходів, фокус лише на продажу товарів

Flora de luxe – це флористична студія, яка спеціалізується на створенні витончених букетів та композицій з живих квітів. Компанія позиціонує себе як місце, де "квіти говорять більше, ніж слова", підкреслюючи емоційну складову своїх послуг.

Переваги:

- Професійний підхід до флористики з акцентом на естетику та емоційну складову
- Індивідуальний підхід до кожного замовлення з урахуванням специфіки події
- Широкий спектр послуг для різних подій (весілля, ювілеї, побачення)

Недоліки:

- Обмеженість послуг лише флористикою без комплексного підходу до організації заходів
- Відсутність детальної інформації про асортимент та ціни на сайті

Bloom.ua – це київська флористична студія, яка спеціалізується на створенні оригінальних квіткових композицій з доставкою. Компанія позиціонує свої роботи як "витвори флористичного мистецтва", підкреслюючи унікальність та естетичну цінність кожної композиції.

Переваги:

- Різноманітність форматів композицій (коробки, кошики, вази, крафтові ящики)
- Зручна система онлайн-замовлення з можливістю доставки по Києву та з-за кордону

Недоліки:

- Фокус лише на квіткових композиціях без додаткових послуг з організації заходів
- Відсутність можливості повної кастомізації композицій онлайн

На основі проведеного аналізу існуючих рішень можна визначити ключові аспекти, які варто врахувати при розробці веб-додатку магазину послуг з організації заходів:

1. Комплексний підхід: На відміну від розглянутих рішень, які фокусуються лише на флористиці, розроблюваний веб-додаток повинен пропонувати комплексні послуги з організації заходів, включаючи флористику та декор.

2. Емоційна складова: Варто перейняти підхід Flora de luxe та Bloom.ua щодо акценту на емоційній складовій послуг, підкреслюючи їх значущість для важливих подій у житті клієнтів.

3. Різноманітність форматів: Доцільно запропонувати різні формати послуг (пакетні пропозиції, індивідуальні замовлення, експрес-послуги) для задоволення потреб різних категорій клієнтів.

4. Інформативність: Важливо забезпечити детальний опис кожної послуги з фотографіями, цінами та відгуками клієнтів для формування довіри та полегшення вибору.

Врахування цих аспектів дозволить створити конкурентоспроможний веб-додаток, який буде відповідати потребам сучасних клієнтів та пропонувати унікальні послуги на ринку організації заходів.

1.5 Постановка завдання

У рамках дипломної роботи необхідно розробити веб-орієнтовану систему для магазину послуг з організації заходів. Основною метою проекту є створення функціонального веб-додатку на базі фреймворку Django, який дозволить ефективно керувати асортиментом послуг та взаємодіяти з клієнтами.

Система має забезпечувати можливість перегляду каталогу послуг з організації заходів, таких як продаж букетів, різноманітних прикрас для весіль, днів народження та інших святкових подій. Користувачі повинні мати змогу

переглядати доступні послуги за категоріями, отримувати детальну інформацію про кожну послугу та оформлювати замовлення.

Для адміністративної частини системи необхідно реалізувати функціонал, який дозволить менеджерам обробляти замовлення, підтверджувати або відхиляти їх, а також керувати каталогом послуг – додавати нові послуги, редагувати існуючі та видаляти застарілі.

Веб-додаток має бути розроблений з використанням сучасних технологій та підходів до веб-розробки, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс як для клієнтів, так і для адміністраторів системи. Важливими аспектами розробки є забезпечення надійності, безпеки та масштабованості системи.

Для досягнення поставленої мети необхідно провести аналіз предметної області, спроектувати архітектуру системи, розробити базу даних, реалізувати серверну та клієнтську частини веб-додатку, а також провести тестування системи для виявлення та усунення можливих помилок.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

ER-модель (Entity-relationship model) – це семантична модель даних, яка призначена для спрощення процесу проектування бази даних. З ER-моделі можуть бути породжені всі види баз даних: реляційні, ієрархічні, мережні, об’єктні. [5]

Основними компонентами ER-діаграми є:

- Сутності (Entities) - об'єкти реального світу, про які зберігається інформація в базі даних
- Атрибути (Attributes) - характеристики або властивості сутностей
- Зв'язки (Relationships) - логічні асоціації між сутностями.

Представлена ER-діаграма на рис. 3 відображає логічну модель даних для веб-додатку. Модель складається з 8 основних сутностей: Категорія, Послуга, Відгук, Користувач, Профіль, Замовлення, Позиція замовлення, Категорія послуги.

Нормалізація - це процес перетворення структури бази даних до нормальних форм. Нормальні форми - це певні правила побудови якісних баз даних. [11]

Загальне призначення процесу нормалізації полягає в наступному: [12]

- виключення деяких типів надмірності;
- усунення деяких аномалій оновлення;
- розробка проекту бази даних, який є досить «якісним» поданням реального світу, інтуїтивно зрозумілий і може бути гарною основою для подальшого розширення;
- спрощення процедури застосування необхідних обмежень цілісності.

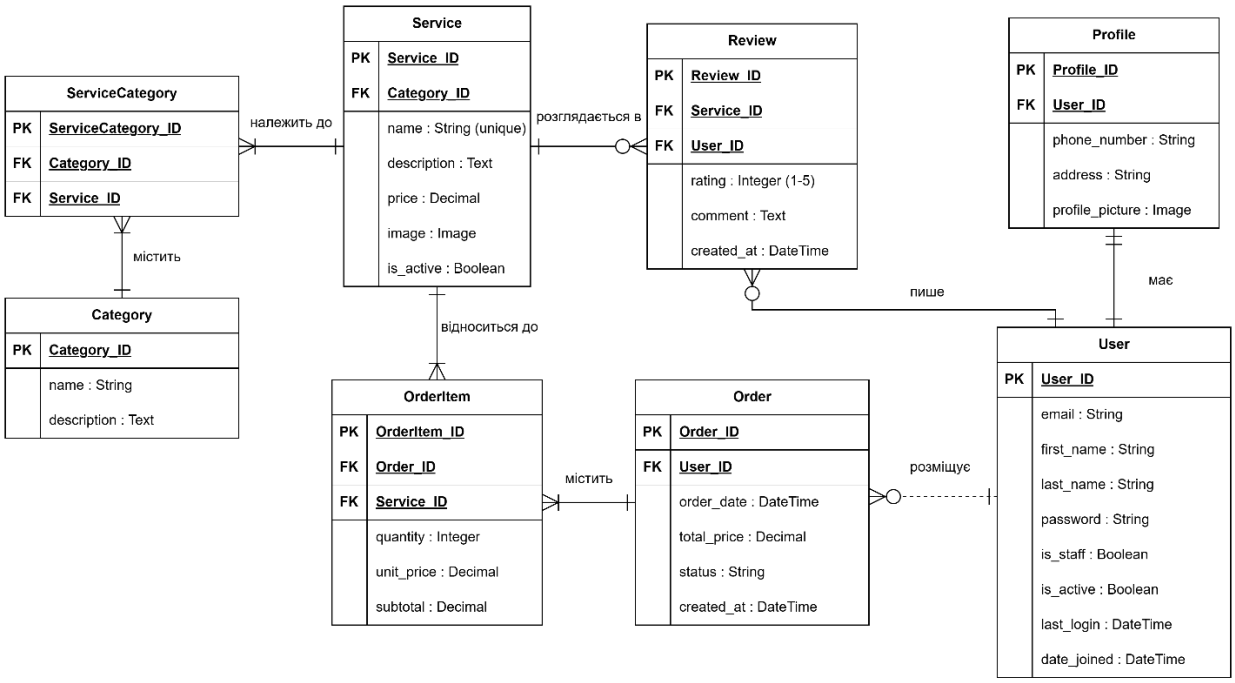


Рис 3. Логічна модель даних у вигляді ER-діаграми

Користувач — це основна сутність, яка містить унікальний ідентифікатор, ім’я користувача, електронну пошту та хеш пароля. Усі атрибути атомарні, не мають повторюваних груп, а кожен атрибут залежить винятково від первинного ключа — отже, сутність відповідає третій нормальній формам.

Профіль — містить додаткову інформацію про користувача, таку як ім’я, прізвище, адреса та номер телефону. Має зовнішній ключ до користувача, що забезпечує залежність один до одного. Всі дані атомарні та логічно пов’язані з користувачем. Сутність перебуває у ЗНФ, бо не має транзитивних залежностей — усі атрибути напряму залежать від ключа (ідентифікатора користувача).

Категорія — описує категорії послуг, містить назву та опис. Кожна послуга може належати до кількох категорій через зв’язкову сутність. Атрибути атомарні та логічно обґрунтовані, всі залежності — лише від первинного ключа.

Послуга — описує конкретні послуги, які можуть бути замовлені. Містить назву, опис, ціну, активність, дату створення та зовнішній ключ на користувача. Дані унікально визначають послугу, кожен атрибут залежить лише від

первинного ключа. Відсутні часткові або транзитивні залежності, отже, послуга перебуває у ЗНФ.

Категорія послуги — є проміжною сутністю між категорією та сервісом. Складається з двох зовнішніх ключів. Це повністю нормалізована таблиця багатозначного зв’язку, що гарантує відсутність дублювання і залежностей, що порушують ЗНФ.

Відгук — це відгук користувача на послугу. Містить рейтинг, текст відгуку, дату створення, зовнішній ключ на користувача та сервіс. Усі атрибути прямо залежать від первинного ключа відгуку, транзитивних залежностей немає.

Замовлення — представляє замовлення, зроблене користувачем. Включає статус, дату створення, дату оновлення, зовнішній ключ на користувача. Усі атрибути залежать лише від первинного ключа замовлення, а не одне від одного — форма відповідає ЗНФ.

Позиція замовлення — відображає перелік послуг у конкретному замовленні. Містить зовнішні ключі на замовлення та послугу, кількість, ціну. Всі атрибути атомарні та логічно залежні тільки від комбінації зовнішніх ключів, які утворюють складений первинний ключ. Транзитивні залежності відсутні.

Усі сутності мають атомарні атрибути (1НФ), усі атрибути залежать лише від первинного ключа (2НФ), і немає транзитивних залежностей між неключовими атрибутами (ЗНФ). Схема повністю відповідає вимогам третьої нормальної форми, що забезпечує мінімізацію надлишковості даних і забезпечує логічну цілісність.

2.2 Діаграма класів та кооперацій

Діаграма класів відображає статичну структуру системи через класи, їх атрибути, методи та взаємозв'язки між ними. У контексті розробки веб-додатку діаграма класів дозволяє наочно представити основні компоненти системи та їх взаємодію. [7]

Зв’язки між класами показують взаємозв’язки між об’єктами класів. Це можуть бути зв’язки типу «агрегація», «композиція», «унаслідування» або «асоціація». [7]

Асоціація показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності. Якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. [17]

Агрегації є особливим типом асоціацій, за якого два класи, які беруть участь у зв’язку не є рівнозначними, вони мають зв’язок типу «ціле-частина». За допомогою агрегації можна описати, яким чином клас, який грає роль цілого, складається з інших класів, які грають роль частин. У агрегаціях клас, який грає роль цілого, завжди має численність рівну одиниці. [14]

Композиція має жорстку залежність часу існування екземплярів класу контейнера та екземплярів класів що містяться в ньому. Якщо контейнер буде знищений, то весь його вміст буде також знищено. [17]

На представленій діаграмі класів (рис. 4) відображено вісім основних класів системи: Споживач, Менеджер, Каталог, Послуга, Замовлення, Кошик, Платіжна система та Квитанція про оплату. Кожен клас містить атрибути та методи, що визначають його функціональність.

Клас "Споживач" представляє користувача системи з атрибутами ім'я, email, пароль, телефон та адреса, а також методами для реєстрації, входу, оновлення профілю та створення замовлень. Клас "Менеджер" відповідає за управління послугами та замовленнями.

Класи "Каталог" та "Послуга" забезпечують функціональність перегляду та управління асортиментом послуг. Класи "Замовлення" та "Кошик" відповідають за процес оформлення та обробки замовлень.

Класи "Платіжна система" та "Квитанція про оплату" забезпечують функціональність оплати замовлень та генерації підтверджень оплати.

Діаграма також відображає взаємозв'язки між класами, такі як "Керує", "Містить", "Створює", "Має" та "Генерує", що демонструють логічні зв'язки між компонентами системи та їх взаємодію в процесі функціонування веб-додатку.

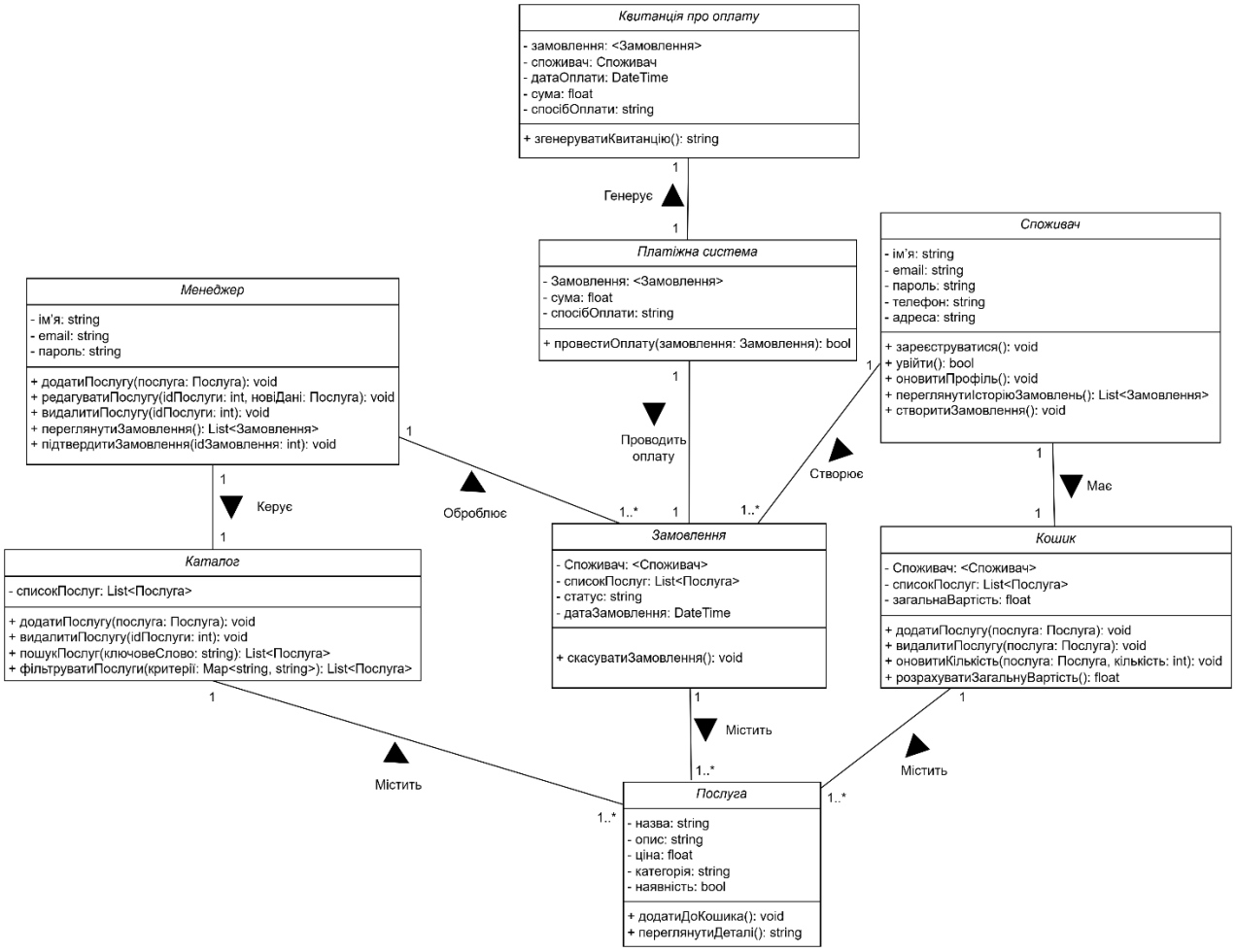


Рис 4. Діаграма класів

Проста кооперація: Взаємодія споживача з кошиком

Кооперація представлена на рис. 5 та демонструє взаємозв'язки між чотирма ключовими класами: Споживач, Кошик, Послуга та Знижка.

Клас "Споживач" містить основні атрибути користувача системи (ім'я, email, пароль, телефон, адреса) та методи для реєстрації, входу, оновлення профілю та перегляду історії замовлень. Кожен споживач має один кошик, що відображено зв'язком "1-1" між цими класами.

Клас "Кошик" відповідає за зберігання обраних послуг та розрахунок загальної вартості замовлення. Він містить атрибути, що вказують на споживача, список послуг та загальну вартість, а також методи для додавання, видалення послуг, оновлення кількості та розрахунку загальної вартості.

Клас "Послуга" представляє послуги, що пропонуються магазином, з атрибутами назва, опис, ціна, категорія та наявність. Між кошиком та послугою встановлено зв'язок "1-1..*", що означає, що кошик може містити одну або більше послуг.

Клас "Знижка" дозволяє застосовувати знижки до кошика, містить атрибути код та величина, а також метод для застосування знижки. Зв'язок між кошиком та знижкою показано пунктирною стрілкою, що вказує на можливість застосування знижки до кошика.

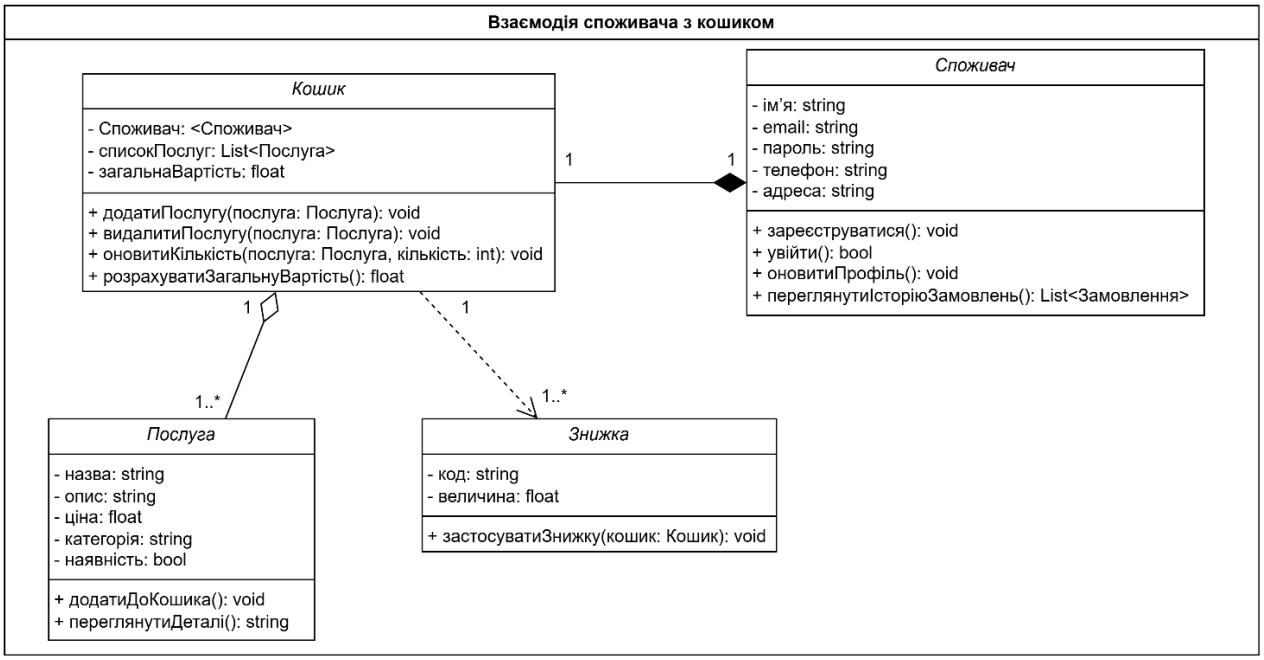


Рис 5. Проста кооперація «Взаємодія споживача з кошиком»

Проста кооперація: Будова каталогу послуг

Кооперація представлена на рис. 6 та демонструє взаємозв'язки між чотирма ключовими класами: Каталог, Категорія, Послуга та Менеджер.

Клас "Каталог" є центральним елементом цієї кооперації та містить список послуг, а також методи для додавання, видалення, пошуку та фільтрації послуг.

Каталог пов'язаний з класом "Категорія" зв'язком "1-1..*", що означає, що каталог може містити одну або більше категорій.

Клас "Категорія" містить атрибути назва та опис, а також методи для додавання та видалення послуг. Категорія пов'язана з класом "Послуга" зв'язком "1-1..*", що вказує на те, що кожна категорія може містити одну або більше послуг.

Клас "Послуга" представляє послуги, що пропонуються магазином, з атрибутами назва, опис, ціна, категорія та наявність, а також методами для додавання до кошика та перегляду деталей.

Клас "Менеджер" відповідає за управління каталогом послуг та має методи для додавання, редагування та видалення послуг, а також для перегляду та підтвердження замовлень. Менеджер пов'язаний з класом "Послуга" зв’язком залежність, що вказує на можливість управління послугами.

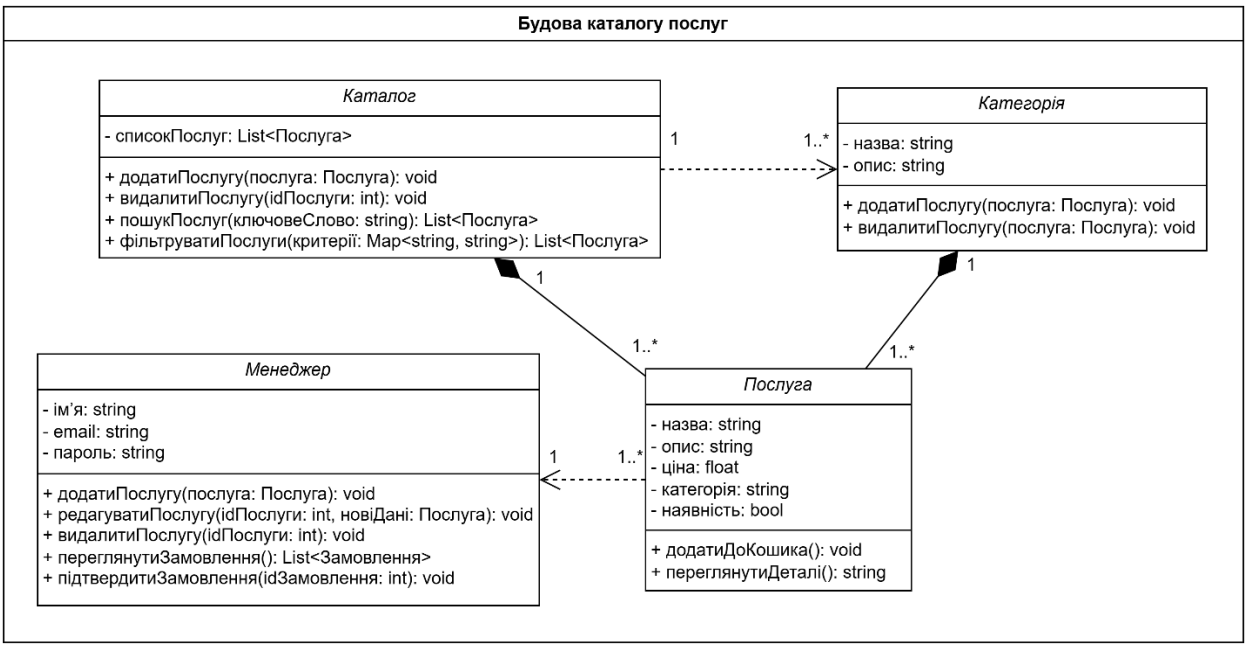


Рис 6. Проста кооперація «Будова каталогу послуг»

2.3 Діаграма пакетів

Діаграмою пакетів є діаграма, що містить пакети класів і залежності між ними. Строго кажучи, пакети та залежності є елементами діаграми класів, тобто

діаграма пакетів – це всього лише форма діаграми класів. Однак на практиці причини побудови таких діаграм різні. [8]

На представленій діаграмі пакетів на рис. 7 відображено основні компоненти веб-додатку та їх взаємодію. Центральним елементом є пакет WEB-SITE, який містить два основні інтерфейси: User Interface та Manager Interface.

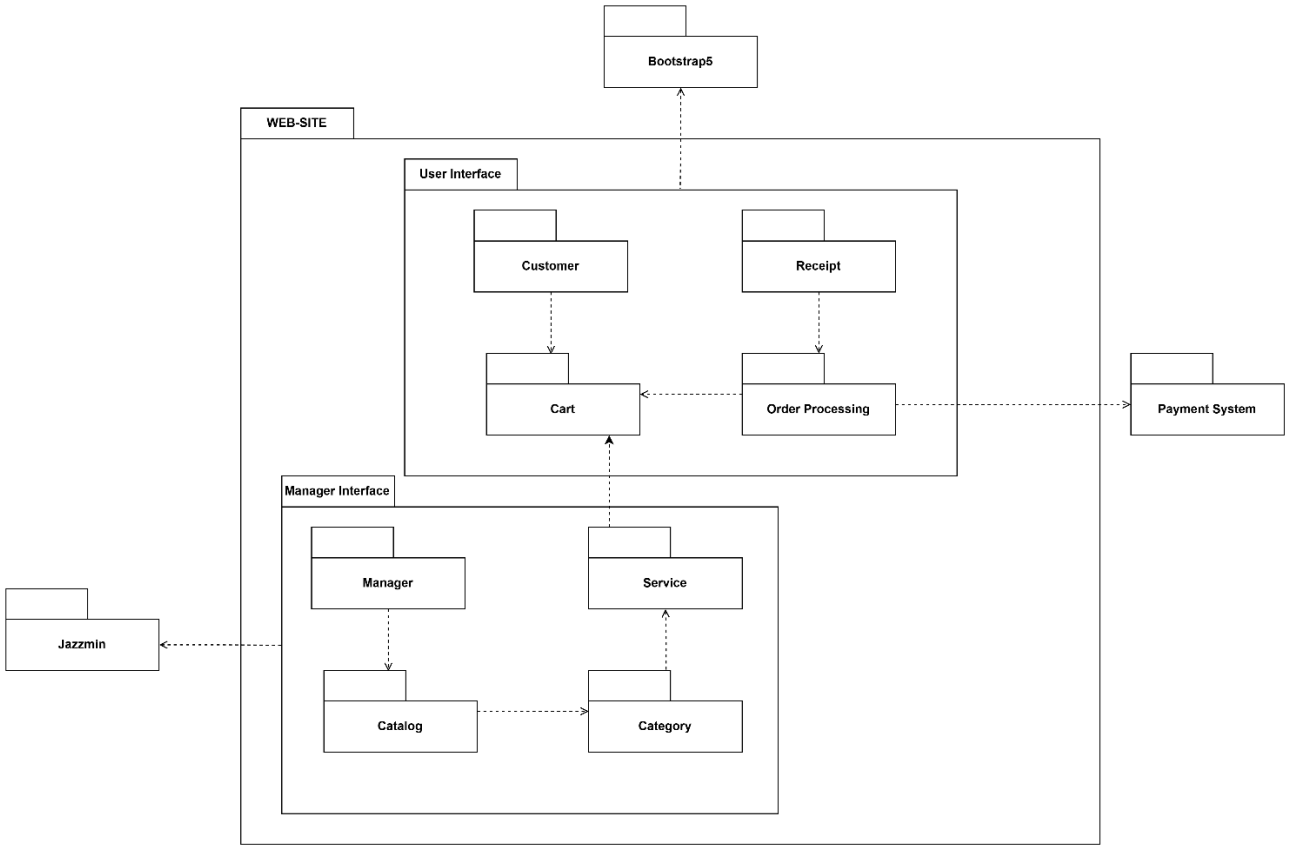


Рис 7. Діаграма пакетів

Пакет User Interface відповідає за взаємодію з клієнтами системи та містить компоненти Customer, Cart, Order Processing та Receipt. Ці компоненти забезпечують функціональність реєстрації користувачів, управління кошиком, обробки замовлень та генерації квитанцій. Пакет User Interface також взаємодіє з зовнішнім компонентом Payment System, що забезпечує функціональність оплати замовлень.

Пакет Manager Interface відповідає за адміністративну частину системи та містить компоненти Manager, Catalog, Service та Category. Ці компоненти

забезпечують функціональність управління каталогом послуг, категоріями та окремими послугами.

Для реалізації інтерфейсу користувача використовується фреймворк Bootstrap5, а для адміністративної панелі – фреймворк Jazzmin, що є розширенням для адміністративної панелі Django.

Взаємозв'язки між компонентами відображено пунктирними стрілками, що вказують на залежності між ними. Наприклад, компонент Cart залежить від компонента Customer, а компонент Order Processing залежить від компонента Cart.

2.4 Діаграма компонентів

Діаграми компонентів UML використовуються для моделювання великих систем на менші підсистеми, якими можна легко керувати. Діаграми компонентів UML використовуються для представлення різних компонентів системи. При моделюванні великих об'єктно-орієнтованих систем необхідно розбивати систему на керовані підсистеми. [23]

Діаграма компонентів для веб-додатку зображена на рис. 8, та надалі відповідно до підсистем описано її застосування.

Підсистема дистрибуції послуг. Виступає ядром системи, відповідаючи за централізоване управління потоком даних між усіма модулями. Ця підсистема інтегрує механізм обробки запитів, який забезпечує зв'язок між клієнтськими запитами та бізнес-логікою. Тут же функціонує модуль аутентифікації, що контролює доступ до системних ресурсів.

Підсистема представлення послуг. Включає каталог послуг, який структурує всі доступні пропозиції за категоріями та параметрами. Це забезпечує відображення детальної інформації про кожну позицію, тоді як кошик виконує функцію тимчасового сховища для вибраних клієнтом елементів перед оформленням замовлення.

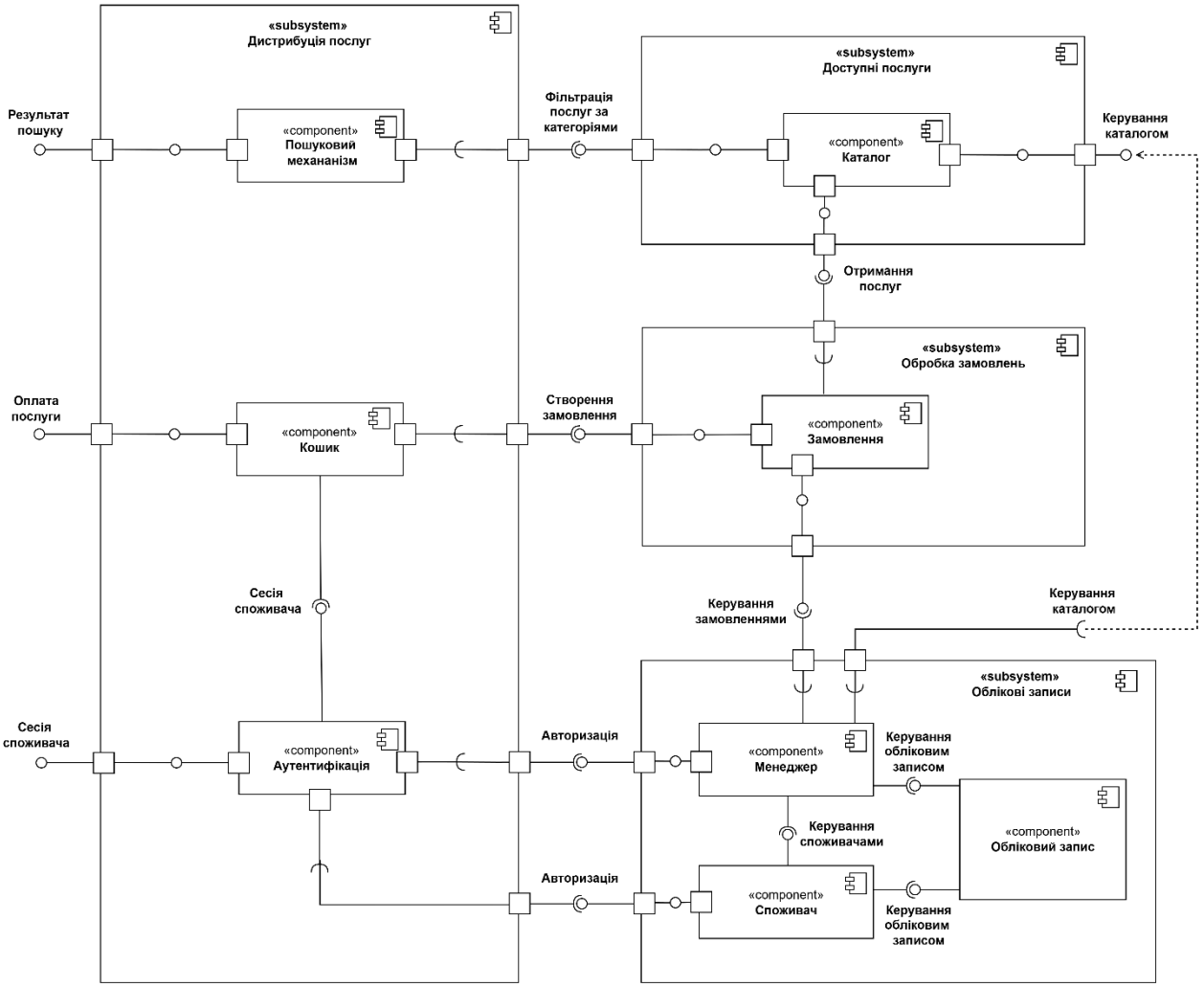


Рис 8. Діаграма компонентів

Підсистема обробки замовлень. Містить модуль, що відповідає за фіксацію та обробку клієнтських запитів. Особливістю є контроль статусів виконання та узгодження деталей замовлення.

Підсистема облікових записів. Ця підсистема забезпечує персоналізацію роботи з системою. Вона включає компонент «Обліковий запис», який зберігає дані користувачів, їх історію замовлень та уподобання. Ця інформація використовується для адаптації інтерфейсу, персоналізації пропозицій та прискорення процесу повторного замовлення.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Система управління базами даних є ключовим компонентом будь-якого сучасного програмного забезпечення, що забезпечує ефективне зберігання, організацію та доступ до даних.

Система управління базами даних (СУБД) – це комплекс програмних і лінгвістичних засобів, що призначені для створення, зберігання, управління та використання баз даних. Він забезпечує ефективне та надійне зберігання даних, а також надає користувачам зручний інтерфейс для роботи з ним. [15]

PostgreSQL — це об'єктно-реляційна система управління базами даних, відома своєю надійністю, потужністю та гнучкістю. PostgreSQL пропонує широкий набір функцій для забезпечення надійності даних, включаючи ACID-відповідність (атомарність, узгодженість, ізоляція і довговічність транзакцій), точки відновлення (point-in-time recovery), реплікацію та автоматичне відновлення. [4]

PostgreSQL має потужну систему індексації, що підтримує різні типи індексів. Це дозволяє оптимізувати продуктивність запитів для різних сценаріїв використання. Крім того, PostgreSQL має вбудовану підтримку для паралельного виконання запитів, що дозволяє ефективно використовувати багатоядерні процесори.

Для розробки веб-застосунку вибір PostgreSQL як СУБД є обґрунтованим з кількох причин:

1. Надійність та цілісність даних. PostgreSQL забезпечує повну підтримку ACID (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність даних навіть у випадку збоїв системи. Це критично важливо для веб-додатку, який обробляє замовлення та платежі, де втрата даних неприпустима.

2. Підтримка складних зв'язків між даними. Веб-додаток має складну структуру даних з багатьма взаємозв'язками між сутностями (послуги, категорії, замовлення, користувачі тощо). PostgreSQL забезпечує підтримку зовнішніх ключів та обмежень цілісності, що дозволяє ефективно моделювати ці взаємозв'язки.

3. Масштабованість. З ростом бізнесу обсяг даних та кількість одночасних користувачів веб-додатку будуть збільшуватися. PostgreSQL пропонує різні механізми масштабування, включаючи реплікацію та шардинг, що дозволяє системі рости разом з бізнесом.

4. Інтеграція з Django. Оскільки веб-додаток розробляється на фреймворку Django, важливо відзначити, що Django має відмінну підтримку PostgreSQL. Django ORM повністю підтримує всі особливості PostgreSQL, включаючи спеціальні типи даних та розширені функції.

Враховуючи всі ці фактори, PostgreSQL є оптимальним вибором для розробки веб-додатку магазину послуг з організації заходів на Django. Вона забезпечує необхідну надійність, функціональність та масштабованість, що дозволяє створити ефективну та стабільну систему, яка відповідає всім вимогам проекту.

3.2 Розробка інформаційної бази

Інформаційна база даного проекту розроблена з використанням Django ORM, що є потужним інструментом для роботи з базами даних у фреймворку Django. Django ORM дозволяє визначати структуру бази даних через моделі Python, що значно спрощує процес розробки та взаємодії з базою даних.

Django ORM пропонує потужний інструмент для керування зв'язками та підтримки цілісності бази даних, спрощуючи при цьому обслуговування коду. Зв'язки між моделями важливі для організації та управління даними в логічному вигляді. [10]

Основні переваги використання Django ORM:

1. Абстракція від конкретної СУБД - код працює однаково незалежно від того, яка система управління базами даних використовується (PostgreSQL, MySQL, SQLite тощо).

2. Автоматична генерація SQL-запитів - розробнику не потрібно писати SQL-запити вручну.

3. Міграції – зручний механізм для відстеження змін у структурі бази даних та їх застосування.

4. Валідація даних – вбудовані механізми перевірки даних перед збереженням у базу.

Структура інформаційної бази представлена на рис. 9, де відображено основні таблиці системи та зв'язки між ними. Центральними сутностями бази даних є користувачі, послуги, категорії, замовлення та відгуки.

Таблиця users_customuser є розширенням стандартної моделі користувача Django і містить додаткові поля для зберігання інформації про користувачів системи. Серед основних атрибутів цієї таблиці: ідентифікатор користувача, електронна пошта, пароль, ім'я, прізвище, телефон, адреса, місто, поштовий індекс, а також службові поля, такі як is_superuser, is_staff, is_active, last_login та date_joined. Ця таблиця забезпечує аутентифікацію та авторизацію користувачів у системі.

Таблиця services_category містить інформацію про категорії послуг, що пропонуються магазином. Кожна категорія має унікальний ідентифікатор, назву та опис. Категорії дозволяють структурувати послуги та полегшити їх пошук користувачами.

Таблиця services_service є однією з ключових у системі та містить інформацію про послуги, що пропонуються магазином. Кожна послуга має унікальний ідентифікатор, назву, опис, ціну, зображення, а також зовнішній ключ на категорію. Додатково таблиця містить поле is_active, що дозволяє тимчасово вимикати послуги без їх видалення з бази даних.

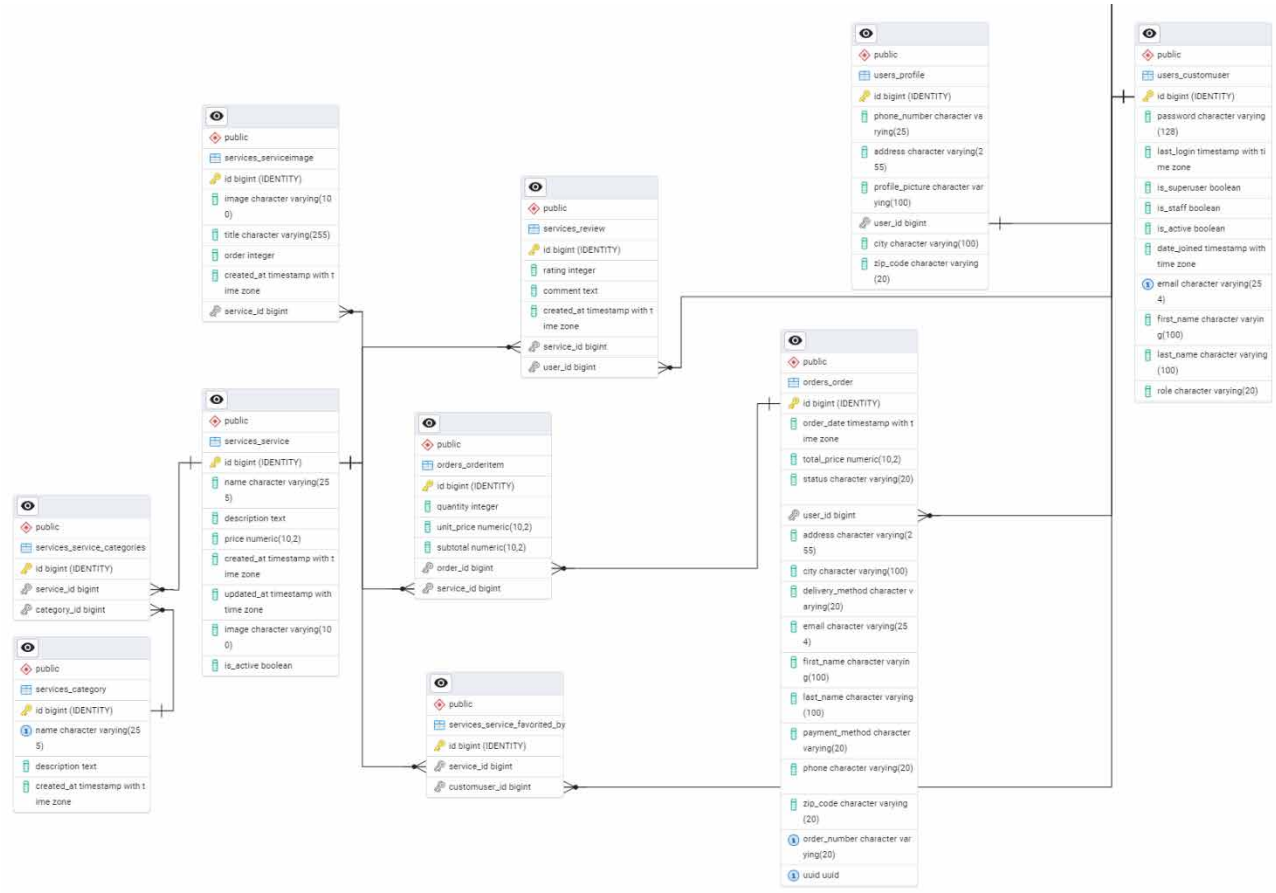


Рис 9. Фізична модель даних в PostgreSQL

Таблиця orders_order зберігає інформацію про замовлення користувачів. Кожне замовлення має унікальний ідентифікатор, дату створення, статус, загальну вартість, а також зовнішній ключ на користувача, який зробив замовлення. Статус замовлення може приймати різні значення, такі як "В очікуванні", "В процесі", "Виконане" або "Скасовано".

Таблиця orders_orderitem реалізує зв'язок "багато-до-багатьох" між замовленнями та послугами. Кожен запис у цій таблиці представляє конкретну послугу в конкретному замовленні та містить інформацію про кількість, ціну за одиницю та загальну вартість. Таблиця має зовнішні ключі на замовлення та послугу.

Таблиця services_review зберігає відгуки користувачів про послуги. Кожен відгук має унікальний ідентифікатор, рейтинг, коментар та дату створення.

Таблиця має зовнішні ключі на користувача, який залишив відгук, та на послугу, якої стосується відгук.

Таблиця services_servicefavorite_by реалізує функціональність послуг зі списку бажань, дозволяючи користувачам зберігати послуги, які їх зацікавили. Ця таблиця має зовнішні ключі на користувача та послугу.

Таблиця services_serviceimage дозволяє зберігати додаткові зображення для послуг, крім основного. Кожне зображення має унікальний ідентифікатор, шлях до файлу та зовнішній ключ на послугу.

Усі таблиці мають відповідні індекси для забезпечення швидкого доступу до даних, а також обмеження цілісності, що гарантують коректність даних. Зв'язки між таблицями реалізовані за допомогою зовнішніх ключів, що забезпечує цілісність даних на рівні бази даних.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка сучасного веб-додатку вимагає використання ефективних та надійних технологій, які забезпечать швидку розробку, зручність підтримки та високу продуктивність системи. У цьому розділі розглянуто основні технології та інструменти, які були обрані для реалізації проєкту.

Django web framework є високорівневим веб-фреймворком на мові Python, який заохочує швидку розробку та чистий, прагматичний дизайн. Він був створений досвідченими розробниками з метою спростити процес створення складних веб-додатків, забезпечуючи при цьому високий рівень безпеки та масштабованості.

Перевага Django перед іншими фреймворками полягає у величезному наборі готового функціоналу, завдяки якому можна швидко і просто створити систему реєстрації на сайті, додати форум, реалізувати систему пошуку або виконати інші дії на сайті. У Django використовується схема MVC. Ця схема дозволяє зручно розділити файли на одну з трьох категорій: HTML-шаблони,

файли моделей для роботи з базою даних і файли контролерів для зв’язку моделей та HTML-шаблонів між собою. [21]

Моделі Django (Models) використовуються для опису структури даних та взаємозв'язків між ними. Наприклад, у проєкті створено моделі для користувачів, послуг, категорій, замовлень та відгуків. Кожна модель представляє собою клас Python, який успадковується від базового класу models.Model та містить поля, що відповідають стовпцям у таблиці бази даних. Детальну реалізацію моделей описано в додатку А.

Представлення Django (Views) відповідають за обробку HTTP-запитів та генерацію відповідей. У проєкті вони є як функціональні представлення, так і класові представлення - Class-Based Views, які забезпечують більш структурований підхід до обробки запитів. Наприклад, для відображення списку послуг використовується класове представлення ListView, а для відображення деталей послуги - DetailView. У додатку Б розписано повну реалізацію основних представлень веб-додатку.

Шаблони Django (Templates) використовуються для генерації HTML-сторінок. Система шаблонів Django дозволяє відокремити логіку представлення від дизайну, що спрощує розробку та підтримку проєкту. Створено шаблони для всіх основних сторінок веб-додатку, таких як головна сторінка, сторінка каталогу послуг, сторінка деталей послуги, сторінка кошика, сторінка оформлення замовлення та інші.

Django Rest Framework (DRF) — це гнучкий і потужний фреймворк для створення RESTful API в проектах на Django. DRF дозволяє розробникам швидко створювати масштабовані API та підходить для різних сценаріїв завдяки своїм широким можливостям. Він вирізняється обробкою даних на основі моделей, потужними механізмами аутентифікації та зручним управлінням помилками. [2]

Представлення DRF (ViewSets) використовуються для обробки HTTP-запитів та генерації відповідей у форматі JSON. Наприклад, створено ViewSet для моделі Service, який забезпечує отримання списку послуг, отримання деталей

послуги, створення нової послуги, оновлення існуючої послуги та видалення послуги.

Bootstrap – це відкритий та безкоштовний HTML, CSS та JS фреймворк для швидкої верстки адаптивних дизайнів сайтів та WEB-додатків. Найчастіше його використовують для фронтенд розробки сайтів та інтерфейсів адмінпанелей. [20]

Система сітки Bootstrap дозволяє створювати адаптивні макети, які автоматично пристосовуються до розміру екрану пристрою. Це забезпечує зручність використання веб-додатку як на настільних комп'ютерах, так і на мобільних пристроях.

JavaScript є мовою програмування, яка використовується для створення інтерактивних веб-сторінок. jQuery полегшує пошук та маніпулювання елементами HTML-сторінки. Він надає зручні методи обробки різних подій на сторінці, таких як клацання миші, натискання клавіш, відправка форми та інші. jQuery спрощує надсилання AJAX-запитів на сервер та обробку отриманих даних. [22]

У проєкті ці технології використовуються для додавання інтерактивності до веб-сторінок. Зокрема, реалізовано такі функції, як динамічне оновлення кошика без перезавантаження сторінки, валідація форм на стороні клієнта та анімація елементів інтерфейсу.

PostgreSQL є потужною об'єктно-реляційною системою управління базами даних з відкритим кодом. Вона має понад 30 років активного розвитку та здобула репутацію надійної, стабільної та функціонально багатої СУБД.

Розроблений з акцентом на розширюваність та сумісність зі стандартом SQL, PostgreSQL підтримує як реляційні, так і деякі NoSQL-функції, такі як зберігання JSON-даних та ключ-значення. Її основне призначення - надавати безпечне та надійне рішення для зберігання даних, яке дозволяє користувачам ефективно керувати складними транзакціями та виконувати аналіз даних у режимі реального часу. [19]

Jazzmin є сучасною темою для адміністративної панелі Django, яка значно покращує її зовнішній вигляд та функціональність. Вона базується на AdminLTE, популярній темі для адміністративних панелей, та додає багато корисних функцій, таких як налаштовуване меню, віджети на панелі приладів, покращена навігація та інші. [3]

Jazzmin дозволяє легко налаштовувати вигляд панелі адміністратора без необхідності значних змін у коді. Серед основних можливостей: інтеграція Select2 для випадаючих списків, модальні вікна замість стандартних попапів, налаштовувані бокове та верхнє меню, а також вбудований UI-кастомайзер для зміни зовнішнього вигляду в реальному часі. Цей інструмент значно покращує користувацький досвід та робить адміністративну панель більш привабливою та функціональною. [3]

Pillow — це бібліотека для мови Python, яка дозволяє обробляти зображення. Вона є форком більш старої бібліотеки PIL (Python Imaging Library). Pillow підтримує широкий спектр форматів зображень, таких як JPEG, PNG, BMP, GIF і навіть незрозумілі TIFF. Вона дозволяє не просто дивитися на зображення, але й обертати їх, змінювати кольори і створювати мініатюри. [6]

Django Crispy Forms є додатком для Django, який дозволяє легко створювати красиві та функціональні форми.

Модуль включає налаштування вигляду форм за допомогою класів FormHelper та Layout, що дозволяє змінювати порядок полів, додавати HTML-елементи, встановлювати ідентифікатори, класи та атрибути без написання кастомних шаблонів. [1]

У проєкті Crispy Forms використовується для створення форм, які відповідають дизайну Bootstrap. Реалізовано форми для реєстрації та входу користувачів, форми для оформлення замовлення, форми для додавання відгуків та інші.

3.4 Алгоритмізація та програмування програмних модулів

Розробка веб-додатку для магазину послуг з організації заходів вимагає ретельного проектування та реалізації програмних модулів, які забезпечують основну функціональність системи. У цьому розділі розглянуто основні алгоритми та програмні рішення, які були використані при розробці ключових модулів системи.

Представлена блок-схема на рис. 10 відображає алгоритм оформлення замовлення у веб-додатку магазину послуг з організації заходів. Алгоритм починається з відкриття користувачем головної сторінки веб-додатку та завершується записом даних замовлення до бази даних.

На початку алгоритму користувач відкриває головну сторінку веб-додатку. Після цього система перевіряє, чи авторизований користувач. Якщо користувач авторизований, система відображає персоналізований інтерфейс з урахуванням його попередніх замовлень та вподобань. Якщо споживач не авторизований, система перенаправляє його на сторінку входу або реєстрації, де він може увійти до свого облікового запису або створити новий.

Після успішної авторизації користувач отримує доступ до функціоналу фільтрації та сортування послуг. Ця функція дозволяє користувачу швидко знайти потрібні послуги за різними критеріями, такими як категорія, ціна або рейтинг.

Наступним кроком користувач вибирає конкретну послугу з каталогу. На цьому етапі система відображає детальну інформацію про послугу, включаючи опис, ціну, доступність та відгуки інших користувачів.

Після вибору послуги користувач додає її до кошика. Система зберігає інформацію про вибрану послугу та її кількість у сесії користувача.

Коли споживач завершує вибір послуг, він переходить до етапу підтвердження замовлення. На цьому етапі система відображає список вибраних послуг, їх кількість та загальну вартість замовлення. Користувач має можливість перевірити правильність вибору та внести зміни за необхідності.

Якщо споживач підтверджує замовлення, система переходить до етапу оформлення замовлення. На цьому етапі користувач вводить або підтверджує інформацію для доставки, вибирає спосіб оплати та вводить додаткові коментарі до замовлення.

Після успішного оформлення замовлення система записує дані замовлення до бази даних. Це включає інформацію про користувача, вибрані послуги, їх кількість, загальну вартість, спосіб доставки та оплати, а також дату та час створення замовлення. Система генерує унікальний ідентифікатор замовлення для подальшого відстеження та надсилає підтвердження замовлення електронною поштою.

Для забезпечення безпеки та зручності користувачів реалізовано механізм аутентифікації та авторизації, який використовує декоратор login_required_with_message. Цей декоратор перевіряє, чи користувач авторизований, і якщо ні, перенаправляє його на сторінку входу з відповідним повідомленням.

Важливим аспектом системи є відправка підтверджень замовлень на електронну пошту користувачів. Для цього використовується функція send_mail з модуля django.core.mail, яка дозволяє відправляти HTML-повідомлення з використанням шаблонів. Шаблон повідомлення формується за допомогою функції render_to_string , яка генерує HTML-код на основі шаблону та контексту.

Для зручності користувачів реалізовано функцію повторного замовлення, яка дозволяє швидко створити нове замовлення на основі попереднього. Ця функція реалізована у функції reorder , яка отримує ідентифікатор попереднього замовлення, завантажує відповідні елементи замовлення з бази даних та додає їх до кошика користувача.

Програмна реалізація описаних в розділі методів та модулів описано в додатках А-Г.

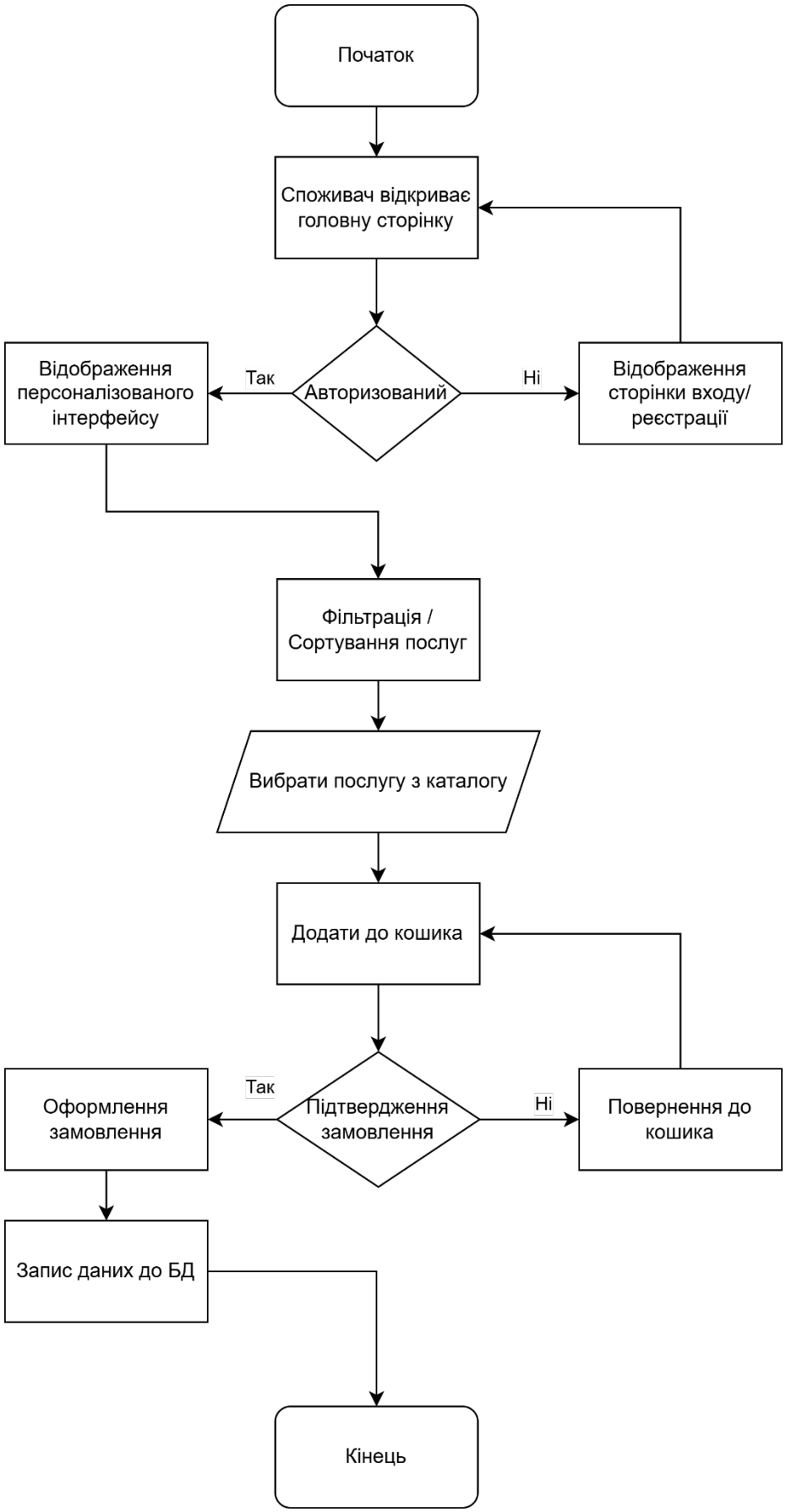


Рис 10. Блок-схема алгоритму оформлення замовлення

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення дозволяє виявити дефекти та переконатися у відповідності системи встановленим вимогам. Для веб-додатку було застосовано кілька методів тестування, зокрема: [24]

1. Функціональне тестування - перевірка відповідності функціональних можливостей системи вимогам, визначеним на етапі проектування. Цей метод дозволив переконатися, що всі функції системи працюють відповідно до вимог.

2. Інтерфейсне тестування – перевірка зручності використання інтерфейсу користувача, його інтуїтивності та відповідності принципам UX/UI дизайну. Особлива увага приділялася адаптивності інтерфейсу для різних пристроїв.

3. Тестування продуктивності – перевірка швидкодії системи при різних навантаженнях, що дозволило виявити та усунути вузькі місця в архітектурі додатку.

4. Тестування безпеки – перевірка захищеності системи від несанкціонованого доступу та інших загроз безпеки, особливо в контексті обробки персональних даних користувачів та платіжної інформації.

5. Тестування сумісності – перевірка коректної роботи додатку в різних браузерах та на різних пристроях, що забезпечило широку доступність системи для користувачів.

Для проведення тестування було залучено групу потенційних користувачів системи, які виконували роль тестувальників. Такий підхід дозволив отримати реальний зворотний зв'язок від цільової аудиторії та виявити проблеми, які могли бути пропущені розробниками.

Процес тестування веб-додатку користувачами розпочався з формування групи з 10 потенційних користувачів різного віку, статі та рівня комп'ютерної грамотності.

Процес тестування розпочинався з реєстрації та авторизації користувачів у системі. Вони заповнювали реєстраційну форму, та входили в систему. Цей етап дозволив перевірити коректність валідації введених даних та зручність процесу автентифікації. Відповідні форми зображені на рис. 11.

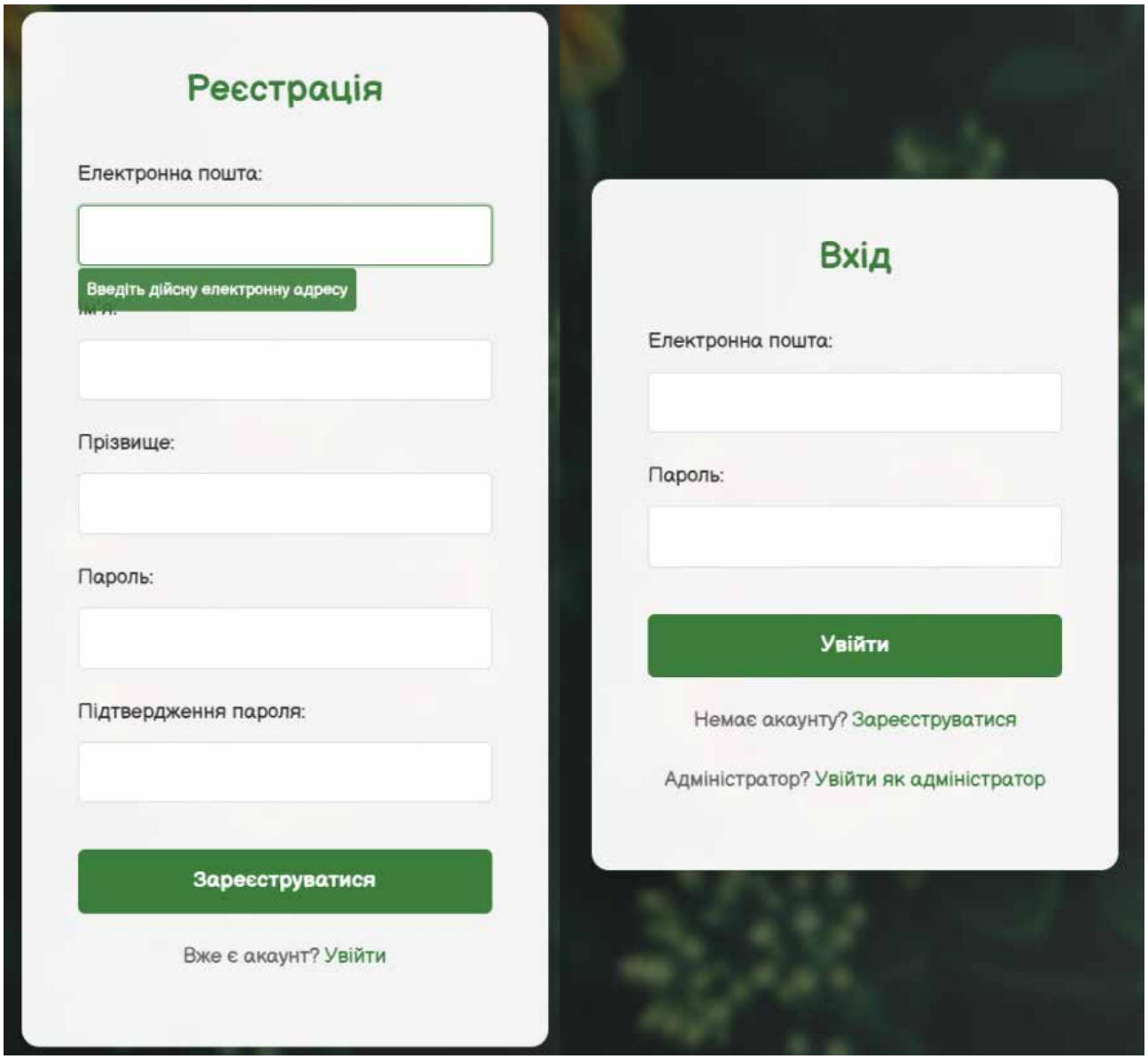


Рис 11. Форми реєстрації та авторизації

Каталог послуг тестувався достатньо делікатно, так як це є важливою складовою проєкту. Інтерфейс каталогу, представлений на рис. 12, надає користувачам зручні інструменти для фільтрації за категоріями та редагування відображення послуг на сторінці. Реалізовано можливість сортування послуг за ціною (як за зростанням, так і за спаданням). Кожна послуга в каталозі

представлена у вигляді інформативної картки з зображенням, назвою, ціною, коротким описом та назвою категорії, до якої вона належить. Такий формат представлення інформації є інтуїтивно зрозумілим та дозволяє швидко орієнтуватися в асортименті послуг.

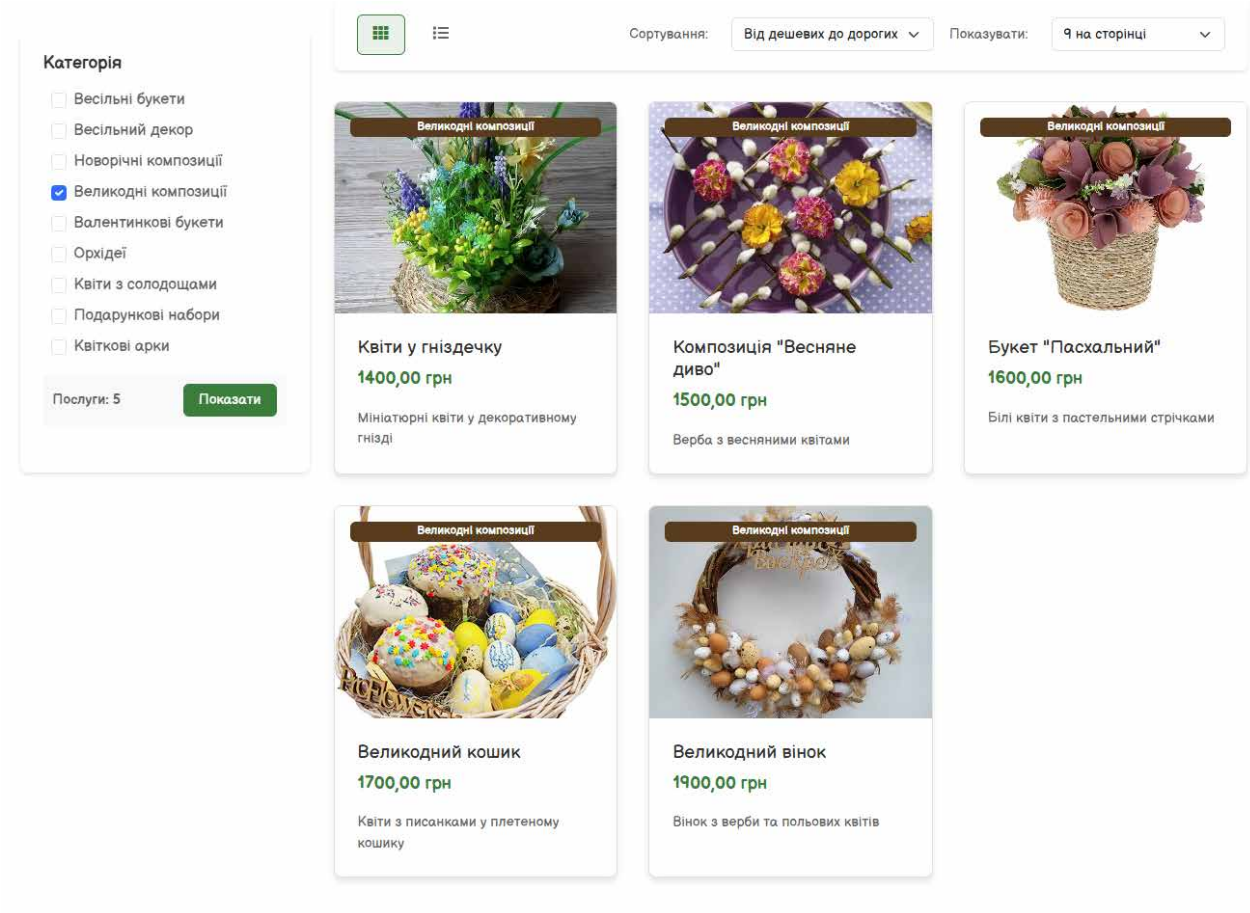


Рис 12. Каталог товарів та послуг з фільтрацією та сортуванням

При переході на сторінку детальної інформації про послугу, зображену на рис. 13, користувач бачить перед собою зображення послуги, рейтинг послуги, можна встановити кількість, додати її до кошику або до списку бажань. Також є можливість поділитися послугою, переглянути категорію, перейти за посиланням, щоб відобразити всі послуги для категорії, і поділитися послугою в соціальних мережах.

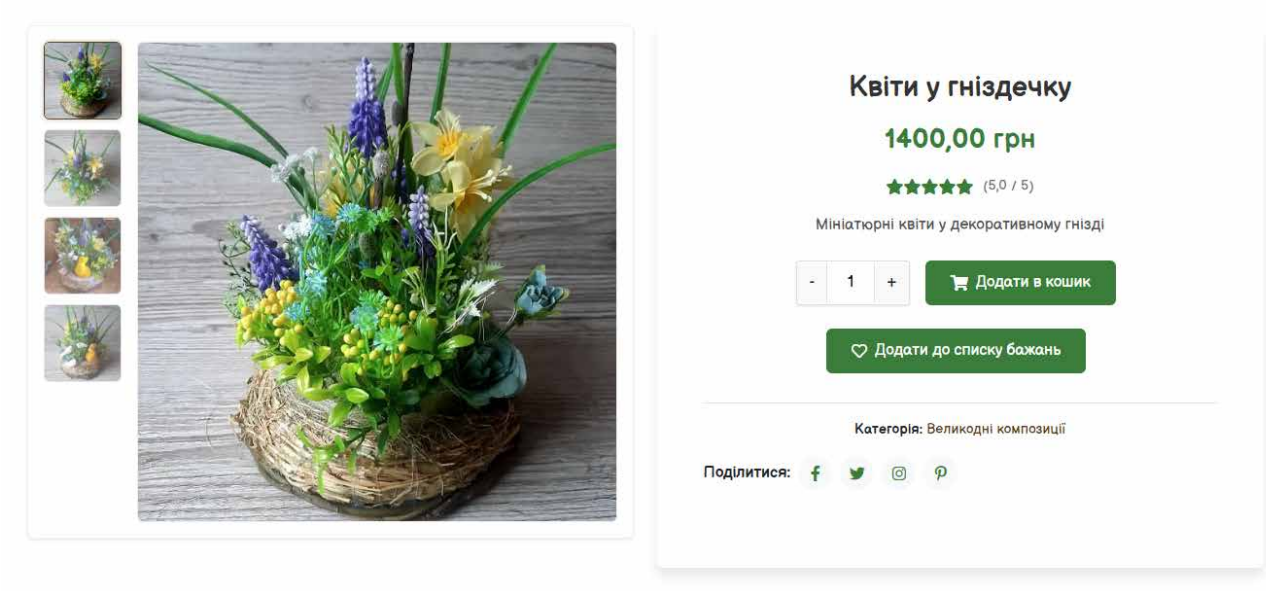


Рис 13. Детальна картка послуги

Трохи нижче на сторінці користувач бачить опис послуги (рис. 14) та відгуки. Вкладка опису містить детальну інформацію про послугу, включаючи її особливості, умови та додаткові опції. Інформація подається в структурованому вигляді, що сприяє кращому розумінню суті послуги та прийняттю обґрунтованого рішення щодо її замовлення.

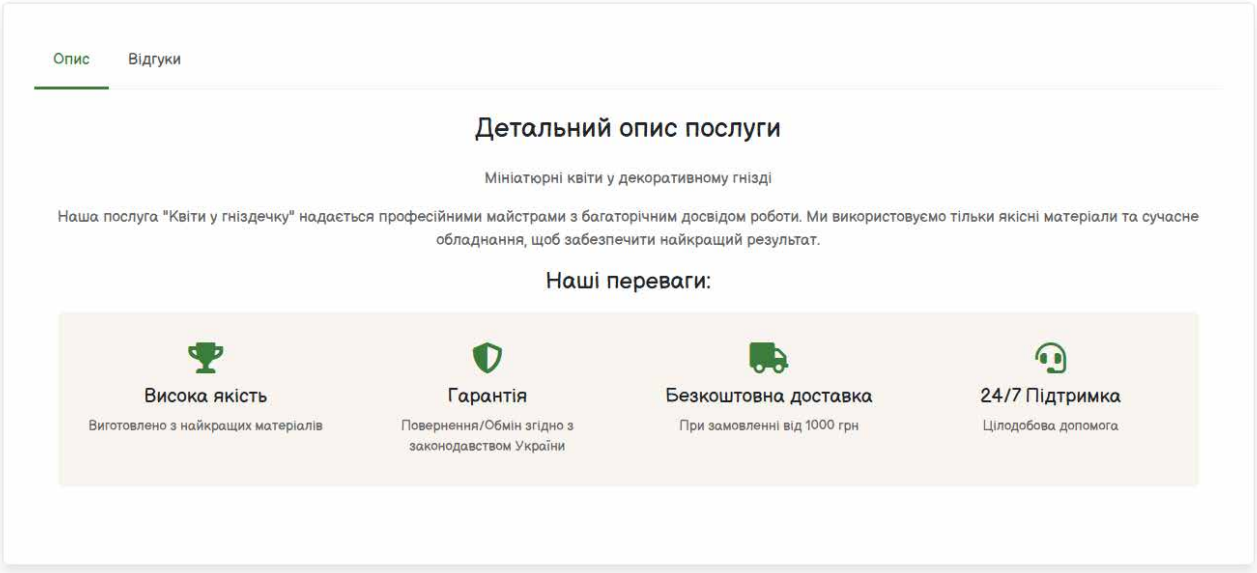


Рис 14. Вкладка опису послуги

Вкладка Відгуки (рис. 15) відображає думки інших клієнтів про послугу, їхніх оцінок та коментарів. Кожен відгук містить інформацію про автора, дату

публікації, оцінку за п'ятибальною шкалою та текстовий коментар. Наявність відгуків сприяє формуванню об'єктивного уявлення про якість послуги та рівень задоволеності клієнтів.

Зокрема процес залишення відгуку (рис. 16-17) достатньо тривіальний, що є user-friendly для користувача. Потрібно лише натиснути на кнопку залишення відгуку, обрати оцінку написати свої враження про послугу і за лічені секунди відгук вже буде відображено на сайті. Навіть якщо користувач випадково допустив помилку, чи хоче доповнити відгук завжди є можливість відредагувати його, або повністю видалити зі сторінки послуги.

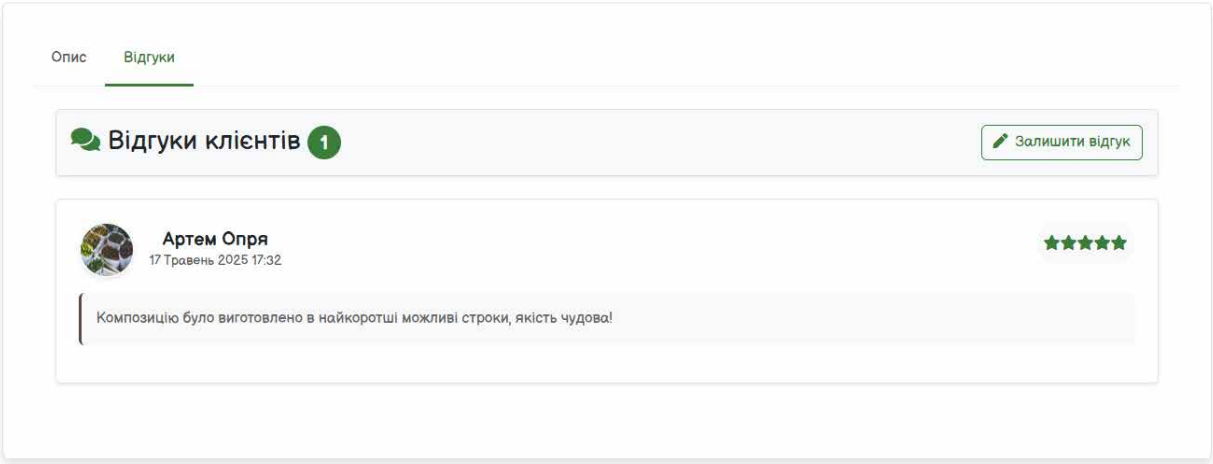


Рис 15. Вкладка відгуків клієнтів

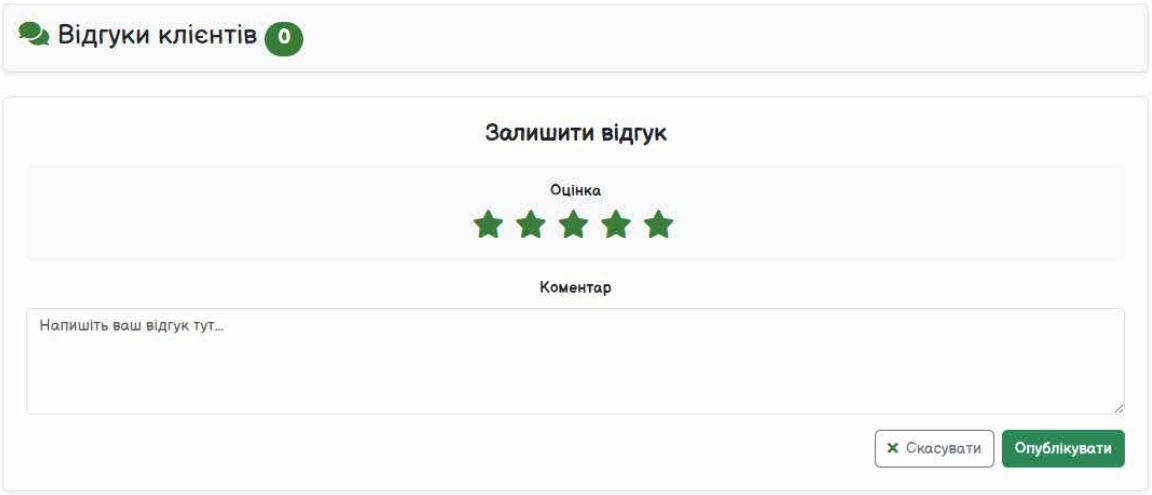


Рис 16. Форма додавання відгуку

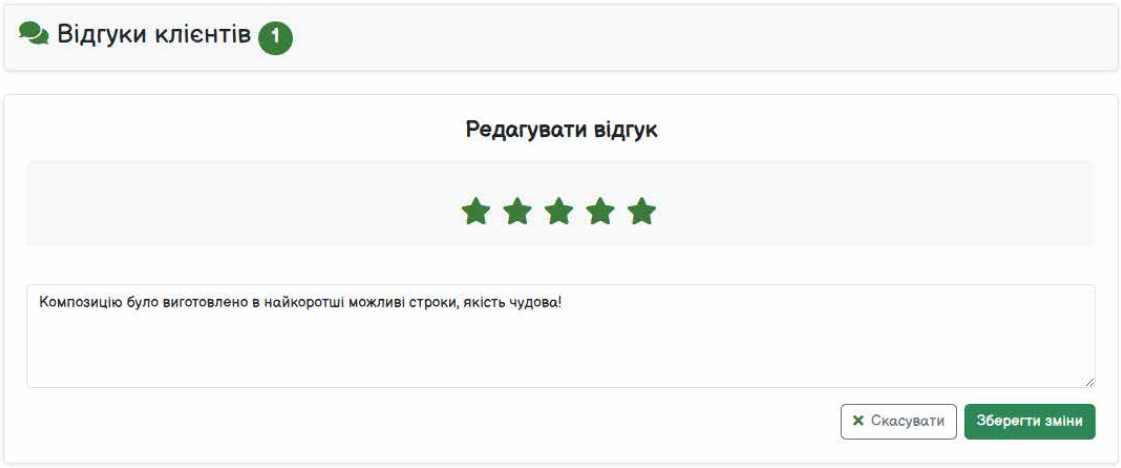


Рис 17. Форма редагування відгуку

Галерея зображень, представлена на рис. 18, розташована під описом та дозволяє користувачам переглянути фотографії, пов'язані з послугою. Реалізовано зручний інтерфейс з можливістю збільшення зображень та перегляду їх у повноекранному режимі. Наявність якісних зображень допомагає краще зрозуміти, що саме пропонується в рамках послуги, та оцінити її візуальну привабливість.

У нижній частині сторінки розташовано блок "Пов'язані послуги", зображений на рис. 18, який містить картки інших послуг з тієї ж категорії або схожих категорій. Цей елемент інтерфейсу реалізує функцію рекомендаційної системи, що дозволяє користувачам ознайомитися з альтернативними варіантами та, можливо, знайти більш підходящу послугу. Такий підхід до рекомендацій є корисним та допомагає розширити уявлення про асортимент послуг магазину.

Тестування показало, що інтерфейс системи є інтуїтивно зрозумілим, а навігація між різними розділами сайту є логічною та зручною. Швидкість завантаження сторінок є оптимальною для комфортного користування системою, хоча на мобільних пристроях з повільним інтернет-з'єднанням спостерігалася деяка затримка при завантаженні галереї зображень. Надана інформація є достатньою для прийняття рішення щодо замовлення послуги.

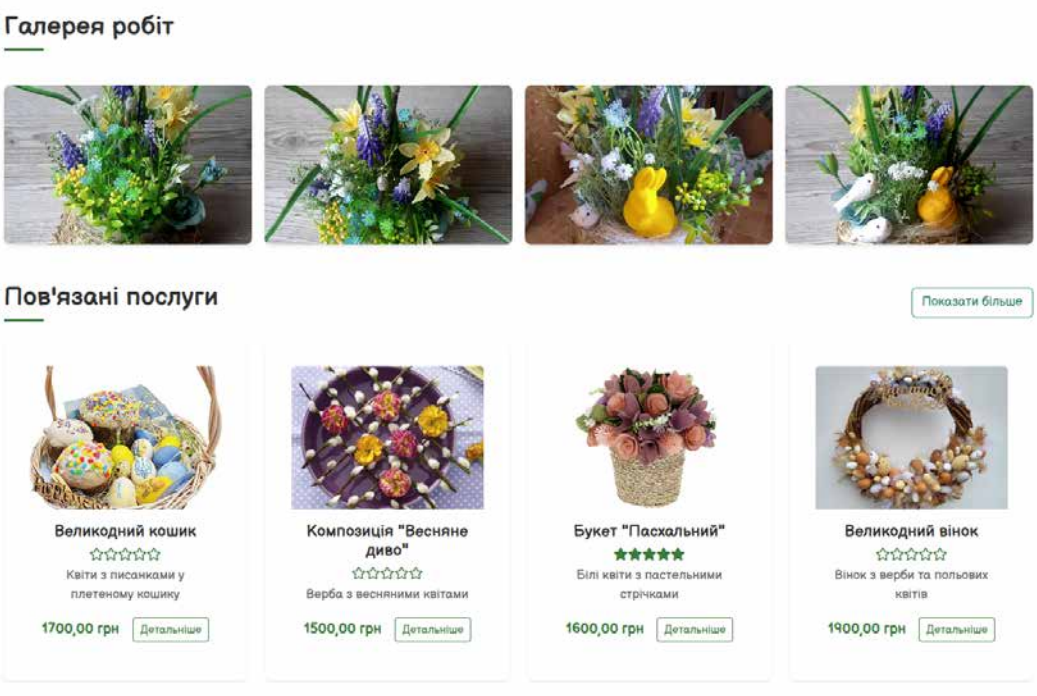


Рис 18. Галерея робіт та пов’язані послуги

Після ретельного вибору послуг користувач може додати їх до кошика, змінювати кількість та видаляли послуги з кошика. На сторінці (рис. 19) розраховується підсумок усього списку послуг, можна перейти до оформлення замовлення або очистити кошик однією кнопкою. Цей етап дозволив перевірити коректність роботи кошика та розрахунку загальної вартості замовлення.

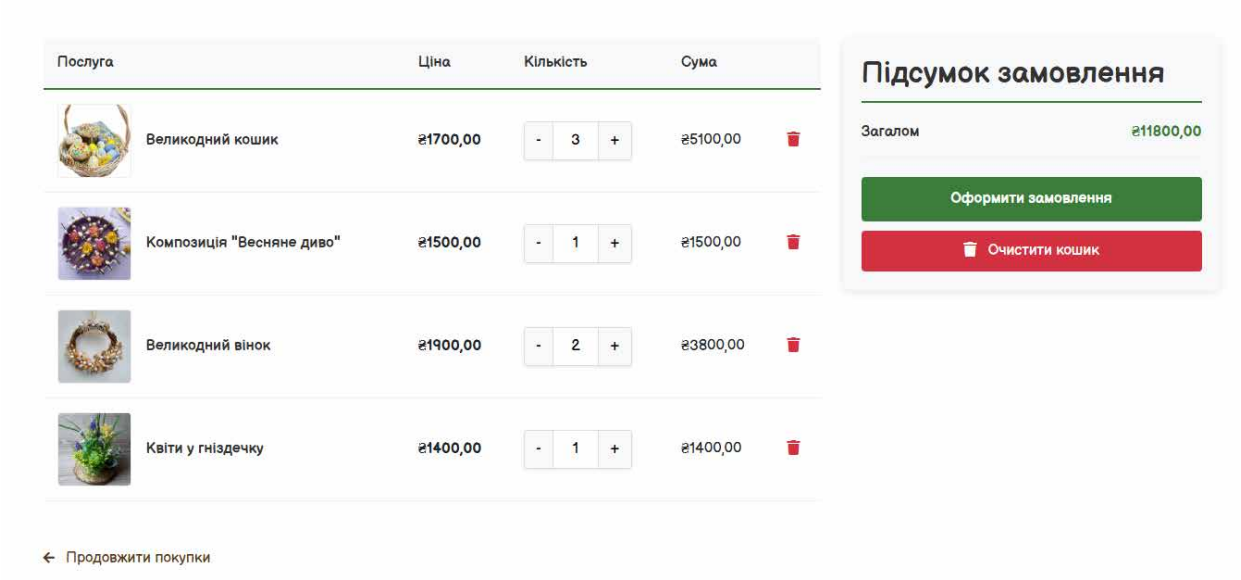


Рис 19. Сторінка кошику користувача

Процес оформлення замовлення включає заповнення форми з контактними даними, вибір способу доставки та оплати (рис. 20). На цьому етапі оцінювалася зручність форми оформлення замовлення, коректність валідації введених даних та інформативність повідомлень про помилки. Реалізовано можливість збереження контактних даних для майбутніх замовлень, що підвищує зручність користування системою.

Після оформлення замовлення та вибору способу оплати онлайн є можливість продовжити та здійснити оплату, використовуючи умовний шлюз Google Pay. Цей етап дозволив перевірити як в майбутньому при інтеграції системи буде відбуватись платіж. На рис. 21-23 відображено процес від початку оплати замовлення користувачем до завершення обробки системою.

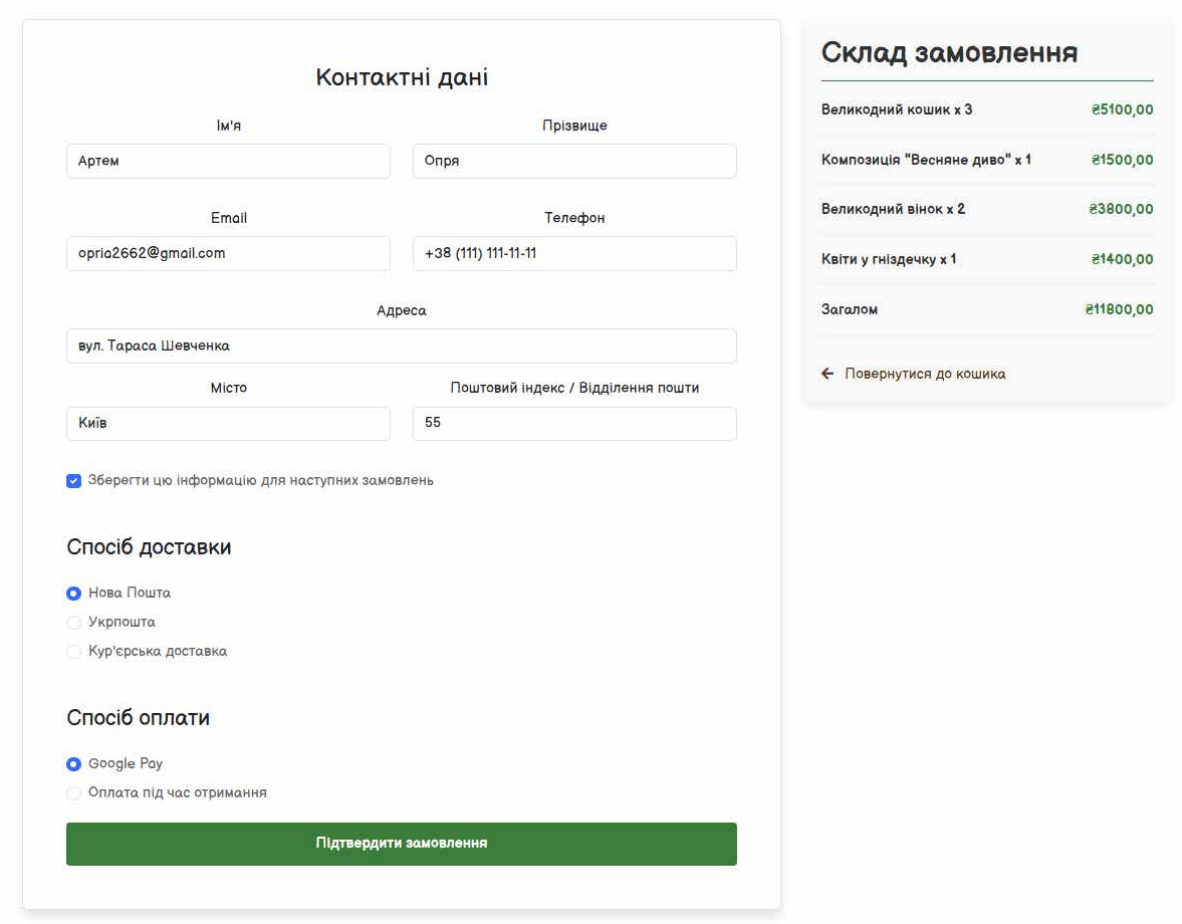


Рис 20. Сторінка оформлення замовлення

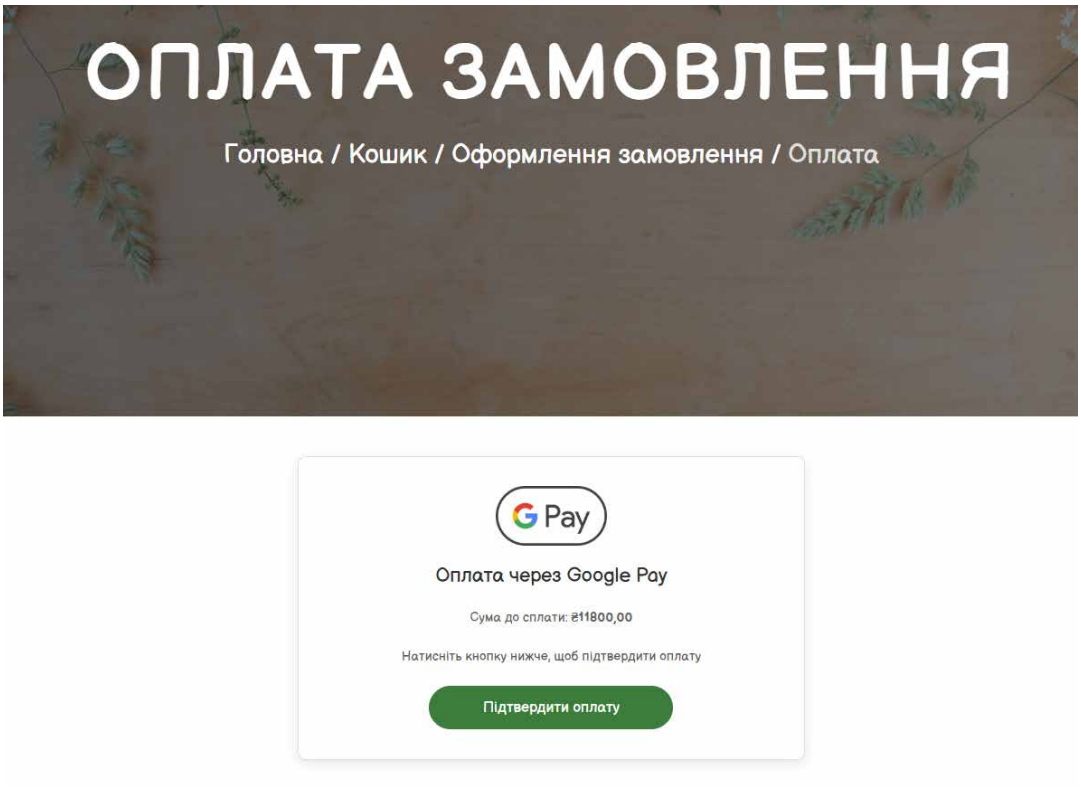


Рис 21. Сторінка оплати замовлення

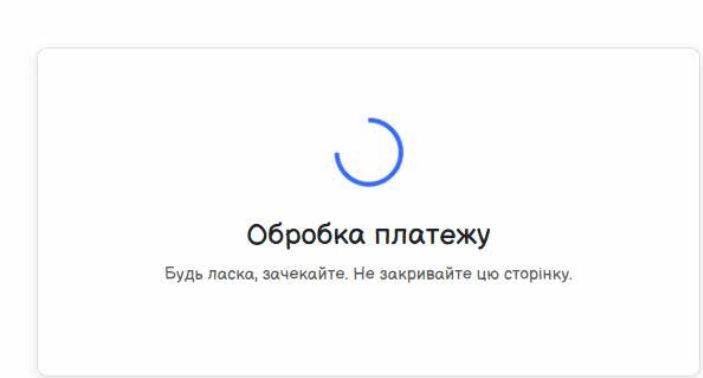


Рис 22. Вікно обробки платежу

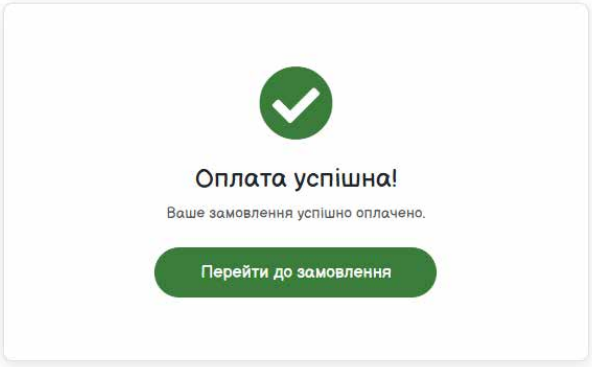


Рис 23. Вікно підтвердження оплати

Після успішної оплати користувачі попадають на сторінку оформленого замовлення (рис. 24), де продемонстровані деталі замовлення, інформація про клієнта, та склад замовлення, також можна повторити замовлення або повернутись до історії замовлень.

Сторінка історії замовлень (рис. 25) оформлена у табличному вигляді, що відображає чітку структуру з номером, датою, статусом та сумою замовлення, біля кожного з замовлень є кнопка деталей, яка перенаправляє на більш детальну інформацію про замовлення.

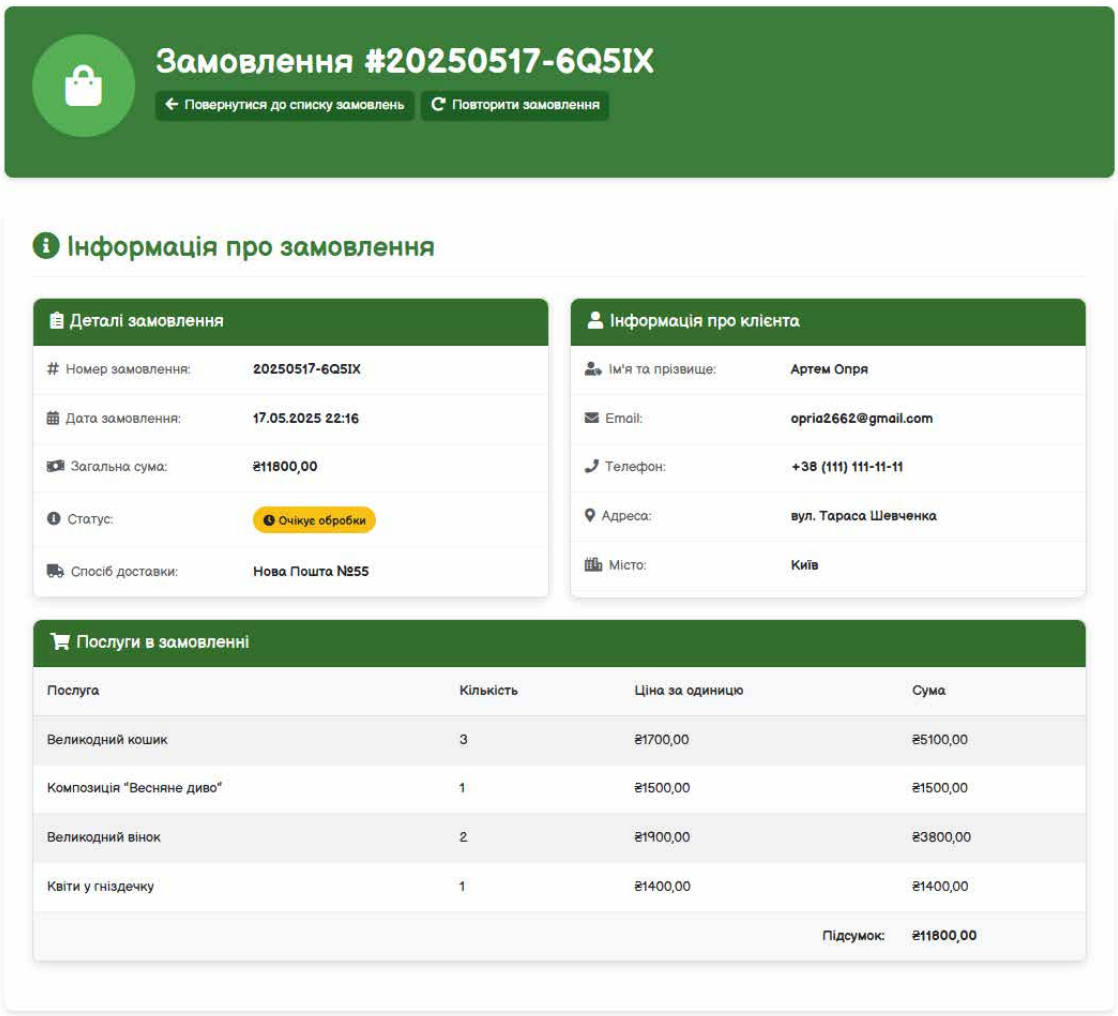


Рис 24. Сторінка оформленого замовлення

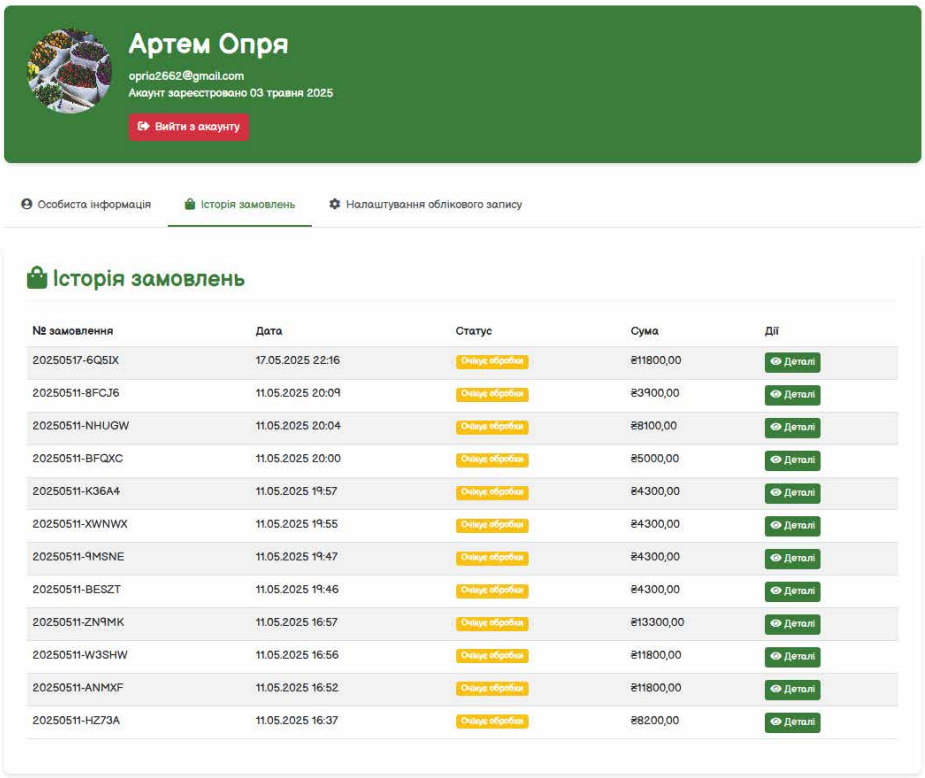


Рис 25. Сторінка історії замовлень

Далі на сторінці особистого кабінету користувач має можливість корегувати особисту інформацію, вибрати аватар для профілю та зберегти зміни (рис. 26).

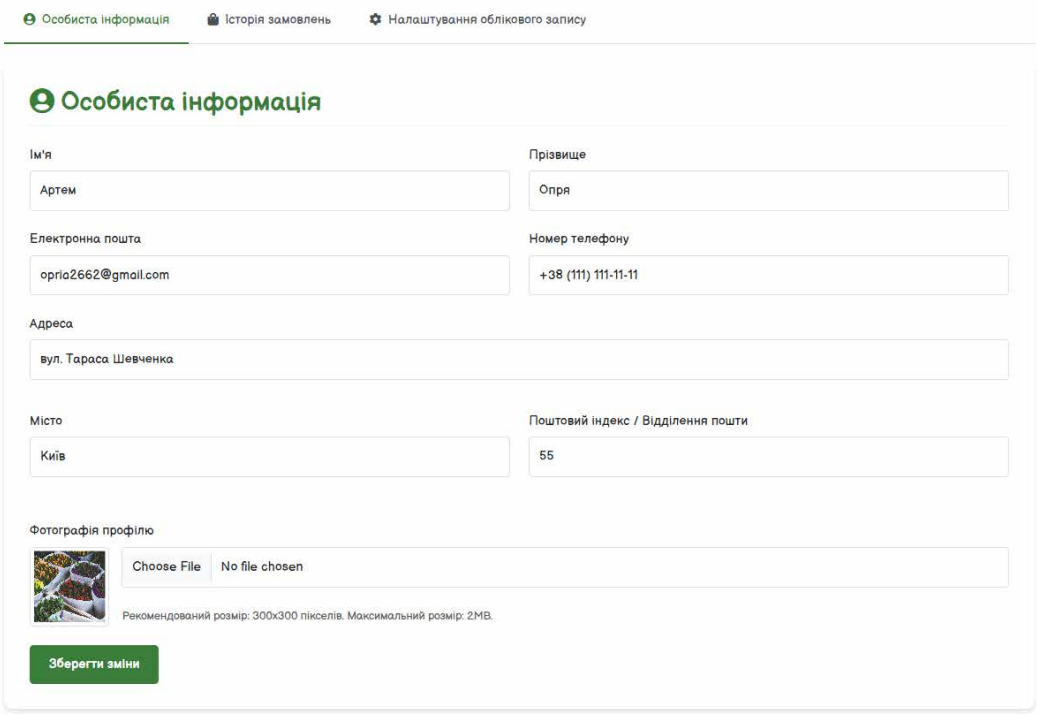


Рис 26. Сторінка особистої інформації користувача

Вкладка «Налаштування облікового запису» (рис. 27) дозволяє змінити пароль, у випадку якщо є підозра про витік старого пароля. На цьому етапі важливо перевірити коректність валідації полів про зміну пароля, та відповідно чи працює цей функціонал. Також, якщо споживач захоче, то може повністю видалити профіль з усіма особистими даними про себе.

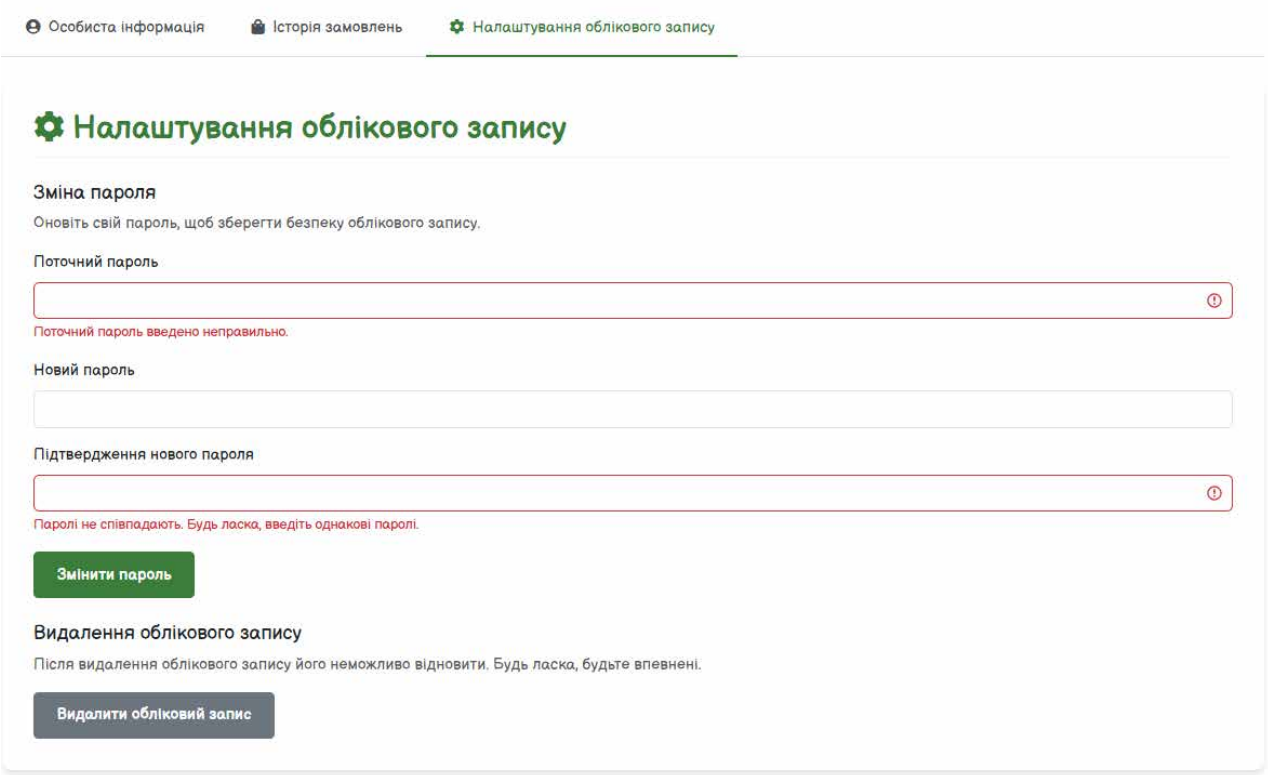


Рис 27. Сторінка налаштувань облікового запису

Сторінка списку бажань (рис. 28) дозволяє зберігати послуги, які їх зацікавили, для подальшого розгляду або замовлення. Картки послуг розташовані в ряд, тому користувач може легко переглядати всі збережені послуги. Кожна картка має однаковий дизайн, це забезпечує візуальний стиль інтерфейсу.

Сторінка пошуку (рис. 29) забезпечує зручний доступ користувачів до потрібних їм послуг. На представленому зображенні видно результати пошуку за певним запитом. Споживач може перейти за посиланням на сторінку послуги і переглянути детальну інформацію про неї.

Зокрема сторінки мають мінімалістичний дизайн, що дозволяє користувачу зосередитися на важливій інформації без відволікання на зайві елементи. Кольорова схема сторінки відповідає загальному стилю сайту, з використанням зеленого кольору для навігаційної панелі, що асоціюється з природою та квітами, що відповідає тематиці сайту.

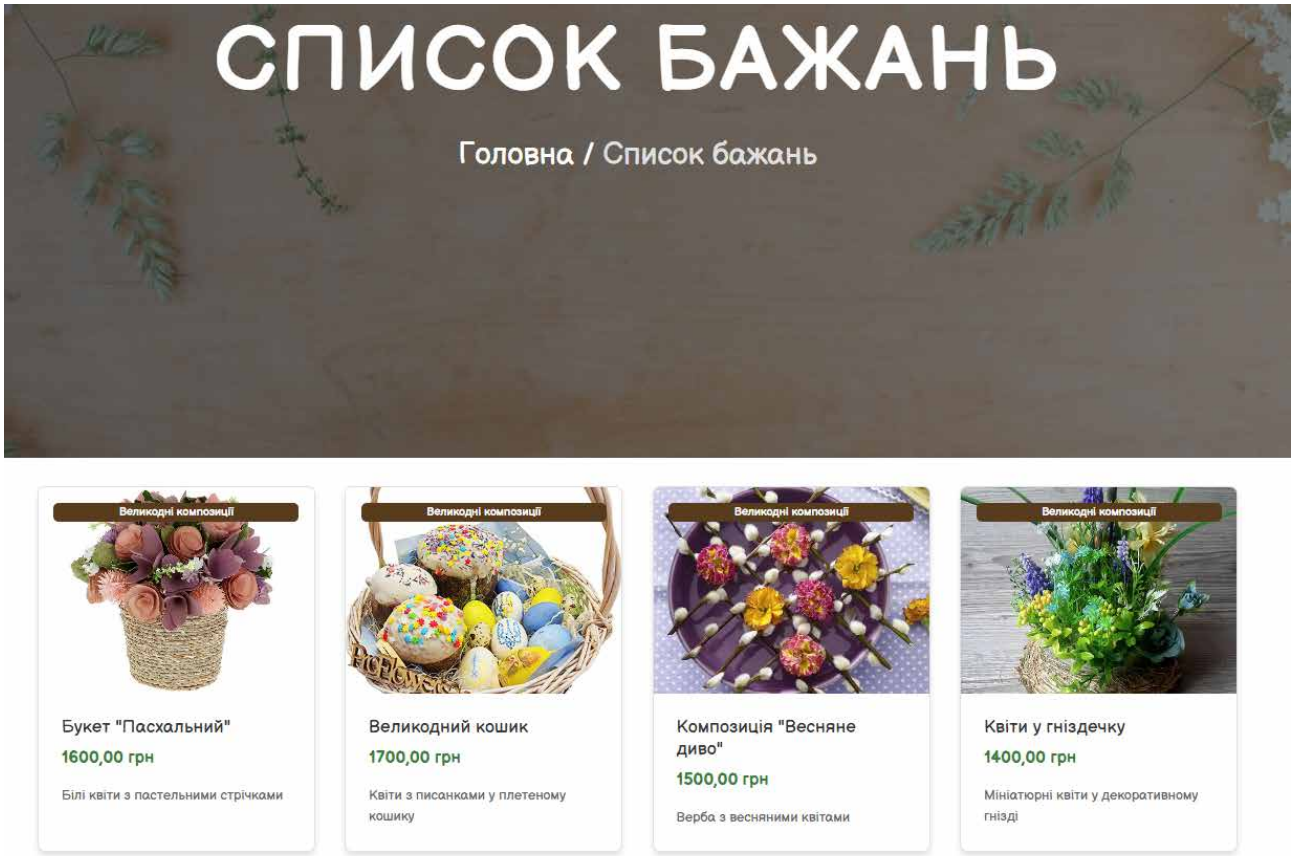


Рис 28. Сторінка списку бажань

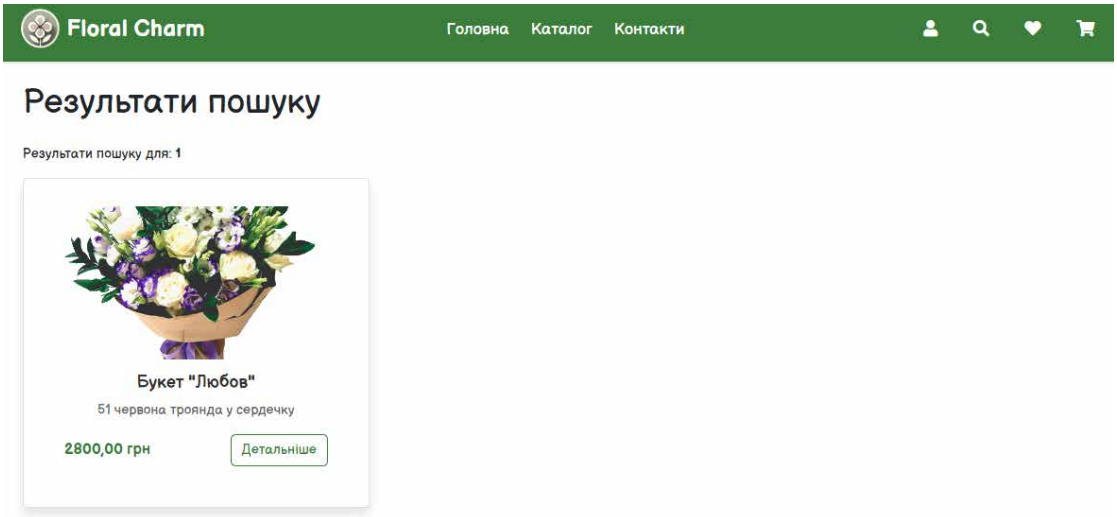


Рис 29. Сторінка результатів пошуку

Наступна сторінка з контактною інформацією (рис. 30) відображає дані про компанію, яка надає відповідні послуги, їх адресу, телефон, електронну пошту, а також години роботи. На сторінці є мапа з міткою точної локації розташування компанії, тому звичайному користувачу буде легко побудувати маршрут від свого місцезнаходження до вказаної локації.

Форма зворотного зв’язку має на меті дати можливість споживачу надіслати повідомлення на пошту компанії, завдяки чому сервіс буде вдосконалюватися та рости відповідно до наданої інформації.

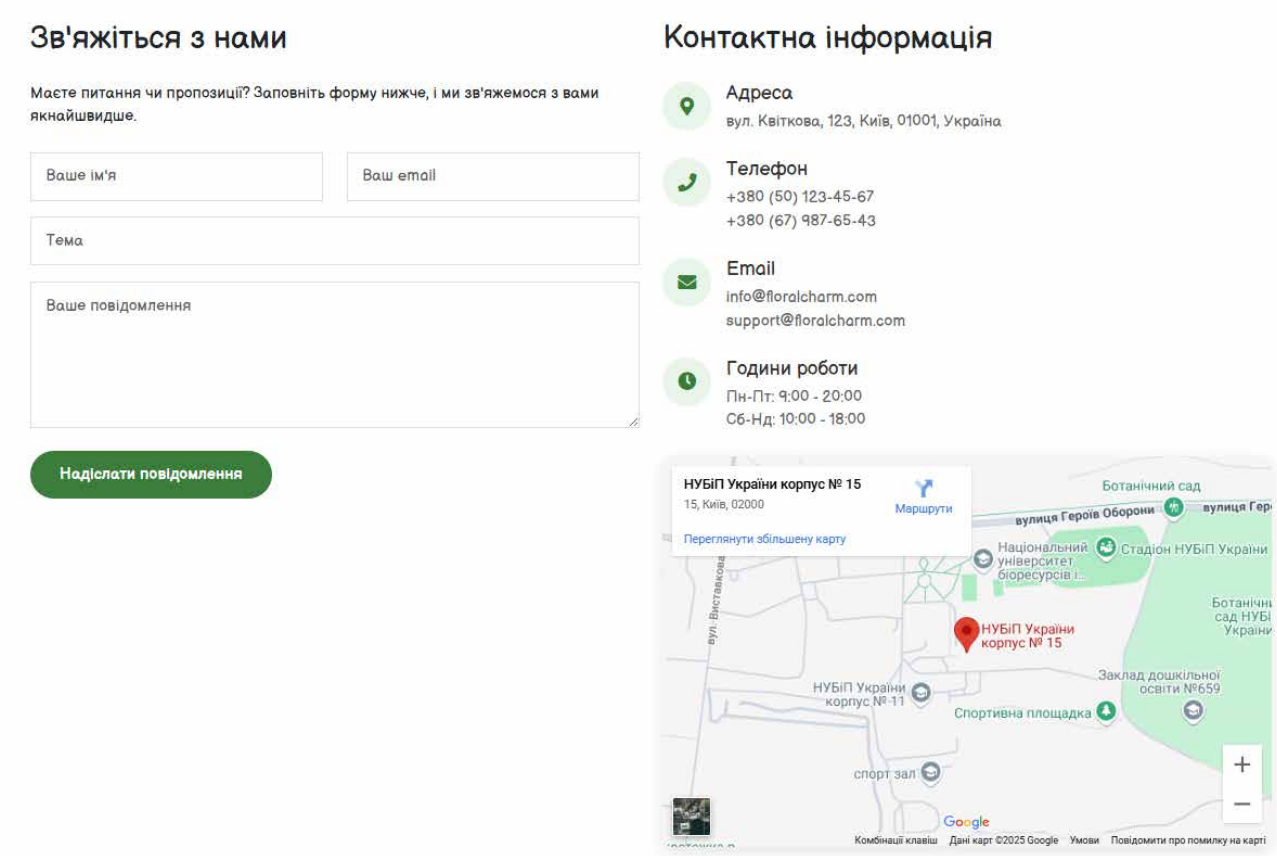


Рис 30. Сторінка контакти

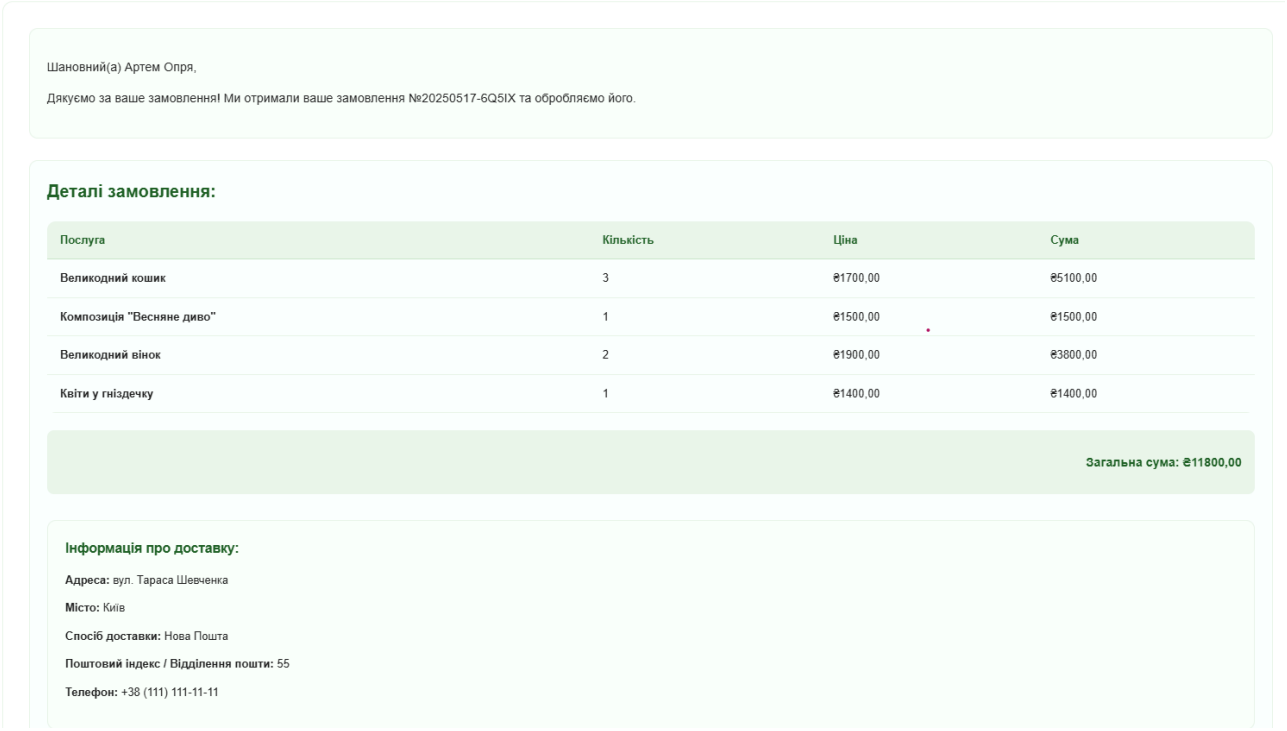
Зокрема на сайті підключено *SMTP* сервіс, що дозволяє надсилати підтвердження отримання замовлення на електронну пошту та надає клієнту всю необхідну інформацію щодо його деталей (рис. 31).

Основна частина повідомлення містить детальну інформацію про замовлення у вигляді таблиці. Вона включає назви замовлених послуг, їх кількість, ціну за одиницю та загальну суму за все замовлення. Такий формат

представлення інформації є зручним для сприйняття та дозволяє клієнту швидко перевірити правильність даних.

Після таблиці з деталями замовлення розміщується блок з інформацією про доставку. Тут міститься адресу доставки, місто, спосіб доставки, поштовий індекс або відділення пошти та контактний телефон. Так клієнт переконується, що його замовлення буде доставлено за правильною адресою та у зручний для нього спосіб.

Дизайн повідомлення виконано у стилі, що відповідає загальному дизайну веб-додатку, з використанням корпоративних кольорів та шрифтів.



Шановний(а) Артем Опря,

Дякуємо за ваше замовлення! Ми отримали ваше замовлення №20250517-6Q5IX та обробляємо його.

Деталі замовлення:

Послуга	Кількість	Ціна	Сума
Великодний кошик	3	₴1700,00	₴5100,00
Композиція "Весняне диво"	1	₴1500,00	₴1500,00
Великодний вінок	2	₴1900,00	₴3800,00
Квіти у гніздечку	1	₴1400,00	₴1400,00

Загальна сума: ₴11800,00

Інформація про доставку:

Адреса: вул. Тараса Шевченка

Місто: Київ

Спосіб доставки: Нова Пошта

Поштовий індекс / Відділення пошти: 55

Телефон: +38 (111) 111-11-11

Рис 31. Повідомлення про оформлене замовлення на пошті користувача

4.2 Вимоги до апаратного та програмного забезпечення

У цьому розділі розглядаються технічні аспекти функціонування веб-додатку магазину послуг з організації заходів. Метою розділу є визначення та обґрунтування вимог до апаратного та програмного забезпечення, необхідних для ефективної роботи системи. Розділ охоплює аналіз архітектури розгортання системи, специфікацію серверної інфраструктури та вимоги до клієнтських

пристроїв. Особлива увага приділяється забезпеченню оптимальної продуктивності, масштабованості та безпеки веб-додатку в умовах різного навантаження та на різних платформах. Як один з способів описати архітектуру веб-додатку, це розглянути діаграму розгортання і зрозуміти, як все побудовано.

Діаграма розгортання (Deployment Diagram) відображає фізичне розташування артефактів системи на обчислювальних вузлах.

Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. [18]

Діаграма розгортання використовує наступні основні елементи:

- **Вузли (Nodes)** – фізичні обчислювальні ресурси, такі як сервери, робочі станції, мобільні пристрої
- **Артефакти (Artifacts)** – фізичні елементи, що створюються та розгортаються в процесі розробки, такі як виконувані файли, бібліотеки, бази даних
- **Зв'язки (Connections)** – комунікаційні шляхи між вузлами, що відображають протоколи взаємодії.

Для веб-додатку було розроблено діаграму розгортання (рис. 32), що відображає всі ключові компоненти системи та їх взаємодію.

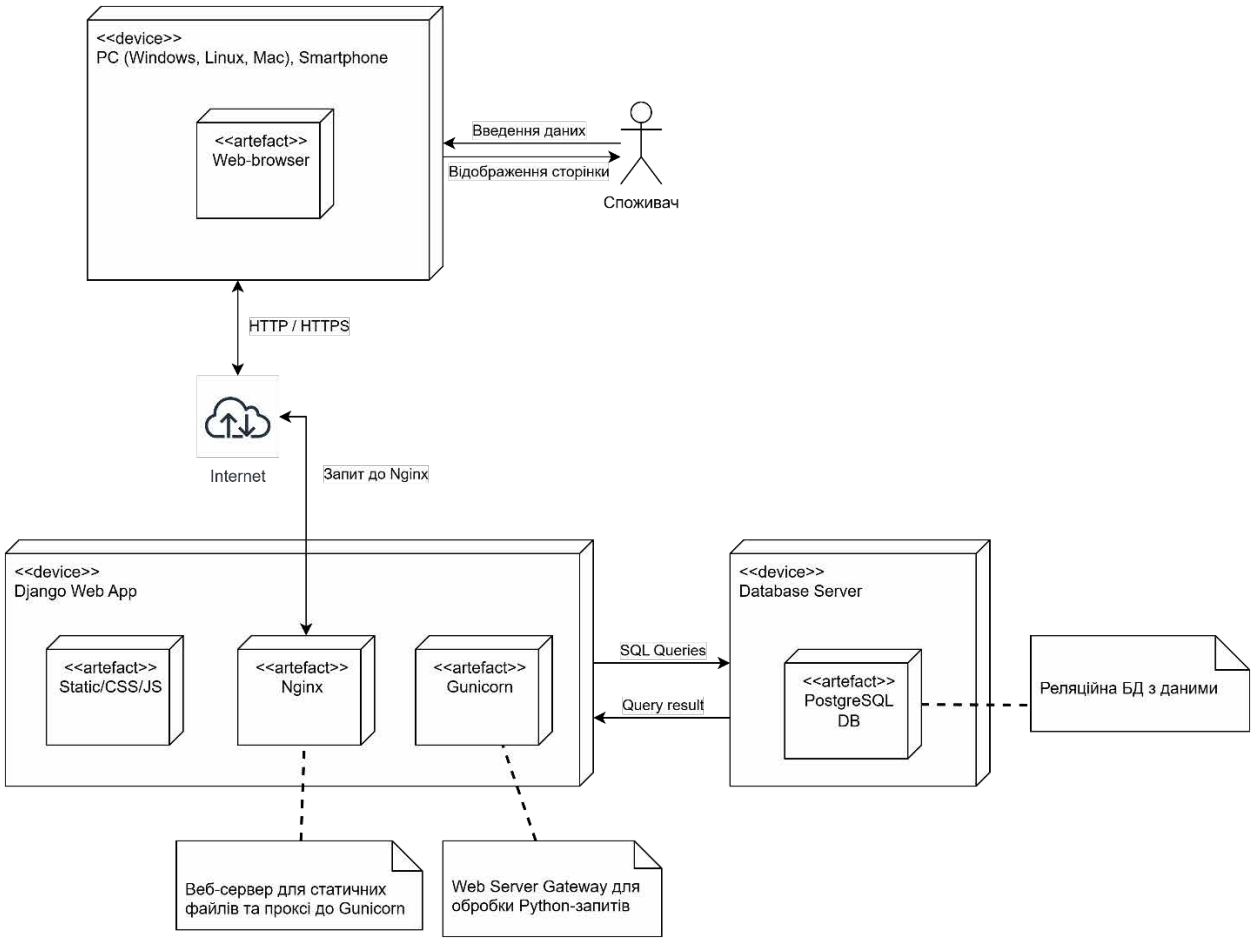


Рис 32. Діаграма розгортання

На діаграмі розгортання представлено наступні основні компоненти:

1. Клієнтська частина:

- Вузол "PC (Windows, Linux, Mac), Smartphone" – представляє пристрої користувачів, з яких здійснюється доступ до веб-додатку.
- Артефакт "Web-browser" – веб-браузер, через який користувач взаємодіє з системою.
- Взаємодія з користувачем відбувається через введення даних та відображення сторінок.

2. Серверна частина:

- Вузол "Django Web App" – основний сервер додатку, що містить бізнес-логіку та відповідає за обробку запитів користувачів.

- Артефакт "Static/CSS/JS" – статичні файли веб-додатку, такі як стилі, скрипти та зображення.
- Артефакт "Nginx" – веб-сервер, що виконує роль проксі та обслуговує статичні файли.
- Артефакт "Gunicorn" – WSGI-сервер для запуску Python-додатків.

3. База даних:

- Вузол "Database Server" – сервер бази даних.
- Артефакт "PostgreSQL DB" – система управління базами даних PostgreSQL, що зберігає всі дані додатку.

4. Комунікаційні зв'язки:

- HTTP/HTTPS – протокол взаємодії між клієнтом та сервером.
- SQL Queries – запити до бази даних.
- Query result – результати запитів до бази даних.

Така архітектура забезпечує високу продуктивність, масштабованість та безпеку веб-додатку. Використання Nginx як проксі-сервера дозволяє ефективно обслуговувати статичні файли та балансувати навантаження. Gunicorn забезпечує стабільну роботу Python-додатку, а PostgreSQL надає надійне зберігання даних з підтримкою транзакцій та складних запитів.

Веб-додаток розроблено з урахуванням принципів адаптивного дизайну, що гарно відображає інтерфейс на пристроях з різною роздільною здатністю екрану. Для мобільних пристроїв оптимізовано завантаження зображень та інших ресурсів, що дозволяє зменшити споживання трафіку та прискорити роботу додатку.

Рекомендується використовувати сучасні веб-браузери з підтримкою протоколу HTTPS та актуальними оновленнями безпеки. Також необхідно регулярно оновлювати операційну систему та веб-браузер до останніх версій.

Для того, щоб створити комфортні умови для роботи користувачів з веб-додатком потрібно дотримуватися певних вимог до клієнтських пристроїв.

Нижче в таблиці 4.1 наведено рекомендовані характеристики для персональних комп'ютерів та мобільних пристроїв.

Таблиця 4.1

Вимоги до апаратного забезпечення пристроїв

Характеристика	**Персональний комп'ютер**	**Мобільний пристрій**
Процесор	Intel Core i3 / AMD Ryzen 3 або потужніший	Qualcomm Snapdragon 665 / MediaTek Helio G80 або потужніший
Оперативна пам'ять	Мінімум 4 ГБ	Мінімум 2 ГБ
Дисковий простір	Мінімум 1 ГБ вільного місця	Мінімум 500 МБ вільного місця
Операційна система	Windows 10/11, macOS 10.15+, Linux з графічним інтерфейсом	Android 8.0+, iOS 12.0+
Веб-браузер	Google Chrome 90+, Mozilla Firefox 88+, Safari 14+, Edge 90+	Google Chrome для Android 90+, Safari для iOS 14+
Роздільна здатність екрану	Мінімум 1366x768	Мінімум 320x568
Інтернет-з'єднання	Мінімум 1 Мбіт/с	Мінімум 1 Мбіт/с

4.3 Склад інсталяційного пакету

Перед початком розгортання проєкту необхідно переконатися у наявності всіх необхідних компонентів програмного забезпечення. Для успішного розгортання веб-додатку потрібно встановити Python версії 3.8 або вище, що є основою для роботи фреймворку Django. Також необхідно встановити систему управління базами даних PostgreSQL версії 12 або вище, яка забезпечує надійне зберігання даних з підтримкою складних запитів та транзакцій.

Першим кроком розгортання проєкту є отримання його вихідного коду. Для цього необхідно клонувати репозиторій проєкту з системи контролю версій.

Після успішного клонування репозиторію потрібно перейти до директорії проєкту. Надалі для ізоляції залежностей проєкту важливо створити віртуальне

середовище Python. Після активації віртуального середовища командний рядок відображатиме назву віртуального середовища, що свідчить про його успішну активацію.

Для зручності управління залежностями проєкту рекомендується використовувати віртуальне середовище Python, що дозволяє ізолювати залежності проєкту від інших проєктів на тій самій системі. Це забезпечує стабільність роботи додатку та запобігає конфліктам між різними версіями бібліотек.

Проєкт містить файл requirements.txt, який перелічує всі необхідні залежності. Виконання команди з взаємодією з ним дозволить інсталювати всі необхідні бібліотеки Python, включаючи Django, драйвер для роботи з PostgreSQL, бібліотеки для роботи з зображеннями, форми та інші компоненти, необхідні для функціонування веб-додатку.

Для роботи веб-додатку необхідно створити базу даних PostgreSQL. Спочатку необхідно встановити сам PostgreSQL, якщо він ще не встановлений в системі. Після встановлення потрібно створити базу даних та користувача для проєкту.

Для безпечного зберігання чутливих даних, таких як паролі та ключі API, проєкт використовує змінні середовища. Для цього потрібно створити файл .env у кореневій директорії проєкту та додати до нього відповідні змінні. Детальніше налаштування відображені в таблиці 4.2.

Щоб мати доступ до адміністративної панелі веб-додатку потрібно створити суперкористувача. у командному рядку будуть відповідні інструкції, щоб ввести ім'я користувача, електронну пошту та пароль для суперкористувача.

Для правильного відображення стилів, скриптів та зображень у веб-додатку необхідно зібрати статичні файли. Надалі всі вони з різних додатків скопіюються у єдину директорію, що спрощує їх обслуговування веб-сервером.

Для тестування веб-додатку в середовищі розробки потрібно запустити вбудований сервер Django. Після запуску сервера веб-додаток буде доступний за

адресою http://127.0.0.1:8000/. Адміністративна панель доступна за адресою http://127.0.0.1:8000/admin/.

Таблиця 4.2

Налаштування конфігураційного файлу

Параметр	**Значення**
SECRET_KEY	*your_secret_key*
DEBUG	*True*
ALLOWED_HOSTS	*localhost, 127.0.0.1*
DATABASE_URL	*postgres://service_store_user:your_password@localhost:5432/service_store*
EMAIL_HOST	*smtp.gmail.com*
EMAIL_PORT	*587*
EMAIL_HOST_USER	*your_email@gmail.com*
EMAIL_HOST_PASSWORD	*your_email_password*
EMAIL_USE_TLS	*True*

Для забезпечення безпеки системи інсталяційний пакет включає ряд заходів:

1. **Використання змінних середовища** - чутливі дані (паролі, ключі API) зберігаються у змінних середовища, а не в коді.

2. **Налаштування HTTPS** - інструкції з налаштування SSL-сертифікатів для забезпечення шифрованого з'єднання.

3. **Налаштування брандмауера** - рекомендації з налаштування брандмауера для обмеження доступу до сервера.

4. **Регулярні оновлення** - інструкції з оновлення залежностей для усунення вразливостей.

5. **Обмеження прав доступу** - рекомендації з налаштування прав доступу до файлів та каталогів.

ВИСНОВКИ

У процесі написання дипломної роботи було розроблено веб-додаток для магазину послуг з організації заходів, який дозволяє автоматизувати основні бізнес-процеси, пов’язані з пошуком, замовленням і адмініструванням послуг, таких як оформлення весіль, свят, продаж букетів та інших декоративних елементів. Створення такої системи дало змогу краще зрозуміти особливості організації предметної області, вимоги до її цифрової трансформації та практичні аспекти веб-розробки на сучасних технологіях.

Проведений аналіз предметної області дозволив визначити основні вимоги до функціональності системи та особливості її реалізації. Зокрема вивчено наявні програмні рішення, змодельовано інформаційні потоки та бізнес-процеси системи за допомогою UML-діаграм. Це дало змогу точно сформулювати функціональні та нефункціональні вимоги до системи, що в подальшому лягло в основу її проєктування. Частина роботи було моделювання структури даних, побудові ER-діаграми, діаграм класів, кооперацій, компонентів, а також послідовності взаємодії між об'єктами системи. Такий підхід дозволив не лише структурувати майбутній код, але й забезпечити цілісність та логічну узгодженість усіх модулів.

Було досліджено існуючі рішення, що дало можливість виявити їхні переваги та недоліки, а також визначити унікальні характеристики розроблюваного веб-додатку.

Проєктування та реалізація системи виконувалися із застосуванням сучасного технологічного стеку, що включає Django як бекенд-фреймворк, PostgreSQL як основну СУБД, Jinja для шаблонізації, HTML/CSS та JavaScript для фронтенду, а також Bootstrap для побудови адаптивного інтерфейсу. Такий вибір дозволив реалізувати ефективну, масштабовану й зручну у використанні систему з розмежуванням ролей користувачів, що має як клієнтську частину, так і адміністративну панель.

Алгоритмізація основних процесів, зокрема оформлення замовлення, дозволила оптимізувати взаємодію користувача з системою та мінімізувати можливість помилок. Розроблені алгоритми забезпечують логічну послідовність дій та враховують різні сценарії взаємодії.

Тестування системи проводилося на різних етапах розробки, що дозволило своєчасно виявляти та усувати помилки. Було проведено функціональне тестування, тестування інтерфейсу користувача, тестування продуктивності та безпеки, що підтвердило відповідність системи встановленим вимогам.

Практична цінність роботи полягає у створенні готового до впровадження веб-додатку, який може бути використаний реальними компаніями, що надають послуги з організації заходів. Система дозволяє автоматизувати процеси взаємодії з клієнтами, підвищити ефективність роботи персоналу та збільшити обсяги продажів.

Подальший розвиток системи може включати інтеграцію з платіжними системами, розширення функціональності мобільної версії, впровадження елементів штучного інтелекту для персоналізації рекомендацій та аналізу поведінки користувачів, а також розширення географії надання послуг.

Отже, результати цієї дипломної роботи продемонстрували доцільність створення такого інструменту для бізнесу в сфері організації заходів. Робота показала, що грамотне проєктування та впровадження інформаційної системи здатне значно покращити комунікацію з клієнтами, спростити обробку замовлень та підвищити якість обслуговування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bancroft M. Django-crispy-forms: Clean Horizontal Multiple Choice Fields. Medium. URL: https://medium.com/@mattbancroft03/django-crispy-forms-clean-horizontal-multiple-choice-fields-564734738287 (дата звернення: 02.05.2025).

2. Django Rest Framework: Основні аспекти розробки RESTful API - JS Communities. JS Communities. URL: https://javascript.org.ua/django-rest-framework-osnovni-aspekti-rozrobki-restful-api/ (дата звернення: 19.04.2025).

3. Jazzmin. URL: https://django-jazzmin.readthedocs.io/ (дата звернення: 20.05.2025).

4. База даних PostgreSQL: інформація та короткий гайд. URL: https://itproger.com/ua/news/baza-dannih-postgresql-informatsiya-i-kratkiy-gayd (дата звернення: 05.04.2025).

5. Бази даних. Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів | BestProg. BestProg | Програмування: теорія та практика. URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/ (дата звернення: 11.04.2025).

6. Вступ до обробки зображень з бібліотекою Pillow. URL: https://javarush.com/ua/quests/lectures/ua.javarush.python.core.lecture.level30.lecture00 (дата звернення: 21.05.2025).

7. Діаграми класів – UA5.org. Матеріали з інформаційних технологій. URL: https://ua5.org/oop/392-diagrami-klasiv.html (дата звернення: 22.05.2025).

8. Діаграми пакетів, компонентів і розміщення. URL: https://ua.kursoviks.com.ua/metodychni_vkazivky/article_post/1701-laboratorna-robota-9-na-temu-diagrami-paketiv-komponentiv-i-rozmishchennya-z-kursu-upravlinnya-vimogami-v-it-proyektakh (дата звернення: 27.03.2025).

9. Моделювання даних (Data Modelling). Maxym Zosym. URL: https://www.maxzosim.com/data-modelling/ (дата звернення: 16.04.2025).

10. Моделі Зв'язків Django ORM: Ефективні Шляхи Керування Зв'язками Бази Даних. JS Communities. URL: https://javascript.org.ua/django-orm-modeli-zvyazkiv-efektivni-shlyahi-keruvannya-zvyazkami-bazi-danih/#Що_таке_Django_ORM (дата звернення: 25.05.2025).

11. Нормалізація та Нормальні Форми. Codefinity: Courses with certificates | Online Learning Platform. URL: https://codefinity.com/ua/courses/v2/5ac24d9d-4a16-45b3-8856-07dec028c5e9/1ae84587-37c3-434a-a228-904a7b0951b5/f7797d90-b825-4886-b55b-0646376b0249 (дата звернення: 17.03.2025).

12. Нормальні форми бази даних. URL: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture02 (дата звернення: 15.04.2025).

13. Опря А.О. Програмне забезпечення з розробки веб-додатку для магазину послуг з організації заходів. Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2025», м. Київ, 24 квітня. 2025 р. НУБІП України. Київ, 2025.

14. Основи UML. Елементи UML. Агрегація. URL: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-elements.html (дата звернення: 18.05.2025).

15. Система управління базами даних: сучасні тенденції у розробці. FoxmindEd. URL: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ (дата звернення: 13.05.2025).

16. Уніфікована мова моделювання (Unified Modeling Language - UML). Maxym Zosym. URL: https://www.maxzosim.com/unifikovana-mova-modeluvannia/ (дата звернення: 16.05.2025).

17. Учасники проектів Вікімедіа. Діаграма класів – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_класів (дата звернення: 02.04.2025).

18. Учасники проектів Вікімедіа. Діаграма розгортання – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Діаграма_розгортання (дата звернення: 01.03.2025).

19. Хостинг-провайдер TheHost. URL: https://thehost.ua/ua/wiki/administration/database/postgresql-install (дата звернення: 03.04.2025).

20. Що таке Bootstrap? – IT Master - електроніка та програмування. URL: https://itmaster.biz.ua/programming/web-prohramuvannia/bootstrap.html (дата звернення: 05.05.2025).

21. Що таке Django? Розробка веб сайтів на Джанго. URL: https://itproger.com/ua/course/django (дата звернення: 09.04.2025).

22. Що таке jQuery?. URL: https://xn----7sbbaqhlkm9ah9aiq.net/ua/info/faq/shcho-take-jquery.html (дата звернення: 29.03.2025).

23. Що таке діаграма компонентів UML в OOAD. Guru99. URL: https://www.guru99.com/uk/component-diagram-uml-example.html (дата звернення: 11.05.2025).

24. Які існують види тестування. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/ (дата звернення: 13.04.2025).

Додаток А

Моделі Django ORM

```
class Order(models.Model):
    order_number = models.CharField(max_length=20, unique=True,
verbose_name='Номер замовлення', blank=True)
    uuid = models.UUIDField(default=uuid.uuid4, editable=False, unique=True,
verbose_name='UUID замовлення')
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, verbose_name='Користувач')
    order_date = models.DateTimeField(auto_now_add=True, verbose_name='Дата
замовлення')
    total_price = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Загальна сума')
    status = models.CharField(max_length=20, choices=[
        ('pending', 'В очікуванні'),
        ('in_progress', 'В процесі'),
        ('completed', 'Виконано'),
        ('cancelled', 'Скасовано'),
    ], default='pending', verbose_name='Статус')

    first_name = models.CharField(max_length=100, blank=True,
verbose_name='Ім\'я')
    last_name = models.CharField(max_length=100, blank=True,
verbose_name='Прізвище')
    email = models.EmailField(blank=True, verbose_name='Електронна пошта')
    phone = models.CharField(max_length=20, blank=True,
verbose_name='Телефон')
    address = models.CharField(max_length=255, blank=True,
verbose_name='Адреса')
    city = models.CharField(max_length=100, blank=True, verbose_name='Місто')
    zip_code = models.CharField(max_length=20, blank=True,
verbose_name='Поштовий індекс')

    delivery_method = models.CharField(max_length=20, choices=[
        ('nova_poshta', 'Нова Пошта'),
        ('ukr_poshta', 'Укрпошта'),
        ('courier', 'Кур\'єрська доставка'),
    ], blank=True, verbose_name='Спосіб доставки')
    payment_method = models.CharField(max_length=20, choices=[
        ('googlepay', 'Google Pay'),
        ('cash_on_delivery', 'Оплата під час отримання'),
```

```
    ], blank=True, verbose_name='Спосіб оплати')

    class Meta:
        verbose_name = 'Замовлення'
        verbose_name_plural = 'Замовлення'
        ordering = ['-order_date']

    def __str__(self):
        return f"Order {self.order_number}"

    def save(self, *args, **kwargs):
        if not self.order_number:
            self.order_number = self.generate_order_number()
        super().save(*args, **kwargs)

    def generate_order_number(self):
        date_str = timezone.now().strftime('%Y%m%d')
        random_str = ''.join(random.choices(string.ascii_uppercase + string.digits, k=5))
        return f"{date_str}-{random_str}"

class OrderItem(models.Model):
    order = models.ForeignKey(Order, related_name='items',
on_delete=models.CASCADE, verbose_name='Замовлення')
    service = models.ForeignKey(Service, on_delete=models.CASCADE,
verbose_name='Послуга')
    quantity = models.PositiveIntegerField(default=1, verbose_name='Кількість')
    unit_price = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Ціна за одиницю')
    subtotal = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Підсумок')

    class Meta:
        verbose_name = 'Послуга в замовленні'
        verbose_name_plural = 'Склад замовлення'

    def save(self, *args, **kwargs):
        self.subtotal = self.quantity * self.unit_price
        super().save(*args, **kwargs)

    def __str__(self):
        return f"{self.quantity}x {self.service.name} in Order
{self.order.order_number}"
```

```
class Category(models.Model):
    name = models.CharField(max_length=255, unique=True, verbose_name='Назва')
    description = models.TextField(blank=True, null=True, verbose_name='Опис')
    created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Створено')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'Категорія'
        verbose_name_plural = 'Категорії'

class Service(models.Model):
    name = models.CharField(max_length=255, verbose_name='Назва')
    description = models.TextField(verbose_name='Опис')
    price = models.DecimalField(max_digits=10, decimal_places=2,
verbose_name='Ціна')
    categories = models.ManyToManyField(Category, related_name='services',
blank=True, verbose_name='Категорії')
    image = models.ImageField(upload_to=service_image_path, null=True,
blank=True, verbose_name='Зображення')
    is_active = models.BooleanField(default=True, verbose_name='Активна')
    created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Створено')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Оновлено')
    favorited_by = models.ManyToManyField(settings.AUTH_USER_MODEL,
related_name='favorite_services', blank=True, verbose_name='У списку бажань')

    class Meta:
        verbose_name = 'Послуга'
        verbose_name_plural = 'Послуги'

    class Admin:
        fieldsets = [
            ('Інформація про послугу', {'fields': ['name', 'description', 'price', 'image',
'is_active']}),
            ('Загальне', {'fields': ['categories']}),
            ('Деталі відгуків', {'fields': ['favorited_by']}),
            ('Дати', {'fields': ['created_at', 'updated_at']}),
        ]

    def __str__(self):
```

```
        return self.name

    def get_average_rating(self):
        reviews = self.reviews.all()
        if reviews.exists():
            total_rating = sum(review.rating for review in reviews)
            return total_rating / reviews.count()
        return 0

    def get_gallery_images(self):
        """Повертає всі зображення галереї для цієї послуги"""
        return self.gallery_images.all()

class Review(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, verbose_name='Користувач')
    service = models.ForeignKey(Service, related_name='reviews',
on_delete=models.CASCADE, verbose_name='Послуга')
    rating = models.IntegerField(validators=[MinValueValidator(1),
MaxValueValidator(5)], verbose_name='Рейтинг')
    comment = models.TextField(verbose_name='Коментар')
    created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Створено')

    class Meta:
        unique_together = ('user', 'service')
        verbose_name = 'Відгук'
        verbose_name_plural = 'Відгуки'

    def __str__(self):
        return f"Review by {self.user.username} for {self.service.name}"

class ServiceImage(models.Model):
    service = models.ForeignKey(Service, related_name='gallery_images',
on_delete=models.CASCADE, verbose_name='Послуга')
    image = models.ImageField(upload_to=service_gallery_path,
verbose_name='Зображення')
    title = models.CharField(max_length=255, blank=True, verbose_name='Назва')
    order = models.PositiveIntegerField(default=0, verbose_name='Порядок')
    created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Створено')

    class Meta:
```

```
        ordering = ['order']
        verbose_name = 'Зображення послуги'
        verbose_name_plural = 'Зображення послуги'

    def __str__(self):
        return f"Зображення для {self.service.name} #{self.order}"

class CustomUserManager(BaseUserManager):
    """
    Custom user manager where email is the unique identifier
    for authentication instead of username.
    """

    def create_user(self, email, password=None, **extra_fields):
        """Create and save a user with the given email and password."""
        if not email:
            raise ValueError(_('The Email must be set'))
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, email, password=None, **extra_fields):
        """Create and save a SuperUser with the given email and password."""
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_active', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError(_('Superuser must have is_staff=True.'))
        if extra_fields.get('is_superuser') is not True:
            raise ValueError(_('Superuser must have is_superuser=True.'))
        return self.create_user(email, password, **extra_fields)

class CustomUser(AbstractUser):
    ADMIN = 'admin'
    MANAGER = 'manager'
    USER = 'user'
    ROLE_CHOICES = [
        (ADMIN, 'Адміністратор'),
        (MANAGER, 'Менеджер'),
        (USER, 'Користувач'),
```

```
    ]

    class Meta:
        verbose_name = 'Користувач'
        verbose_name_plural = 'Користувачі'

    username = None
    email = models.EmailField(_('email address'), unique=True)
    first_name = models.CharField(_('first name'), max_length=100)
    last_name = models.CharField(_('last name'), max_length=100)
      role = models.CharField(_('role'), max_length=20, choices=ROLE_CHOICES, default=USER)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name']

    objects = CustomUserManager()

    def __str__(self):
        return self.email

class Profile(models.Model):
        user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, verbose_name='Користувач')
            phone_number = models.CharField(max_length=25, blank=True, verbose_name='Номер телефону')
                address = models.CharField(max_length=255, blank=True, verbose_name='Адреса')
    city = models.CharField(max_length=100, blank=True, verbose_name='Місто')
                zip_code = models.CharField(max_length=20, blank=True, verbose_name='Поштовий індекс')
                 profile_picture = models.ImageField(upload_to='profile_pics', default='profile_pics/default.jpg', blank=True, verbose_name='Фото профілю')

    class Meta:
        verbose_name = 'Профіль'
        verbose_name_plural = 'Профілі'

    def __str__(self):
        return f"{self.user.email}'s Profile"
```

Додаток Б

Валідація форм Django

```
class CustomUserCreationForm(UserCreationForm):
    email = forms.EmailField(
        label='Електронна пошта',
        error_messages={
            'required': 'Це поле обов\'язкове.',
            'invalid': 'Введіть коректну електронну адресу.'
        }
    )
    first_name = forms.CharField(
        label="Ім'я",
        error_messages={'required': 'Це поле обов\'язкове.'}
    )
    last_name = forms.CharField(
        label='Прізвище',
        error_messages={'required': 'Це поле обов\'язкове.'}
    )
    password1 = forms.CharField(
        label='Пароль',
        widget=forms.PasswordInput(),
        error_messages={
            'required': 'Це поле обов\'язкове.',
            '    password_too_short': 'Пароль занадто короткий. Він повинен містити принаймні 8 символів.',
            'password_too_common': 'Цей пароль занадто поширений.',
            'password_entirely_numeric': 'Пароль не може складатися тільки з цифр.',
            'password_too_similar': 'Пароль занадто схожий на ваше прізвище.'
        }
    )
    password2 = forms.CharField(
        label='Підтвердження пароля',
        widget=forms.PasswordInput(),
        error_messages={
            'required': 'Це поле обов\'язкове.',
        }
    )

    def clean_password2(self):
        password1 = self.cleaned_data.get("password1")
        password2 = self.cleaned_data.get("password2")

        if password1 and password2 and password1 != password2:
            raise ValidationError(_('Паролі не співпадають.'))

        return password2
```

```
    class Meta:
        model = User
        fields = ('email', 'first_name', 'last_name', 'password1', 'password2')

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['email'].widget.attrs.update({'class': 'form-control'})
        self.fields['first_name'].widget.attrs.update({'class': 'form-control'})
        self.fields['last_name'].widget.attrs.update({'class': 'form-control'})
        self.fields['password1'].widget.attrs.update({'class': 'form-control'})
        self.fields['password2'].widget.attrs.update({'class': 'form-control'})

class CustomAuthenticationForm(AuthenticationForm):
    username = forms.EmailField(
        label='Електронна пошта',
        widget=forms.EmailInput(attrs={'class': 'form-control'}),
        error_messages={
            'required': 'Це поле обов\'язкове.',
            'invalid': 'Введіть коректну електронну адресу.'
        }
    )
    password = forms.CharField(
        label='Пароль',
        widget=forms.PasswordInput(attrs={'class': 'form-control'}),
        error_messages={'required': 'Це поле обов\'язкове.'}
    )

    error_messages = {
        'invalid_login': 'Елекронна пошта або пароль введено невірно',
        'inactive': 'Цей обліковий запис неактивний.'
    }

class ProfileForm(forms.ModelForm):
    first_name = forms.CharField(max_length=100, required=False)
    last_name = forms.CharField(max_length=100, required=False)

    class Meta:
        model = Profile
        fields = ['phone_number', 'address', 'city', 'zip_code', 'profile_picture']

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if self.instance and self.instance.user_id:
            self.fields['first_name'].initial = self.instance.user.first_name
            self.fields['last_name'].initial = self.instance.user.last_name

        self.fields['phone_number'].widget.attrs.update({'type': 'tel'})
    def save(self, commit=True):
        profile = super().save(commit=False)
        if commit:
            if 'first_name' in self.cleaned_data and 'last_name' in self.cleaned_data:
```

```
            user = profile.user
            user.first_name = self.cleaned_data['first_name']
            user.last_name = self.cleaned_data['last_name']
            user.save()
        profile.save()
    return profile

class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = ['rating', 'comment']
        widgets = {
            'rating': forms.Select(choices=[(i, i) for i in range(1, 6)]),
            'comment': forms.Textarea(attrs={'rows': 4, 'placeholder': 'Напишіть ваш відгук тут...'}),
        }
        labels = {
            'rating': 'Оцінка',
            'comment': 'Коментар',
        }
```

Додаток В

Основна логіка обробки модулів додатку

```
def order_detail(request, order_id):
    order = get_object_or_404(Order, id=order_id, user=request.user)
    return render(request, 'orders/order_detail.html', {'order': order})

@login_required_with_message(message="Для оплати замовлення необхідно увійти в акаунт.")
def payment(request):
    order_id = request.session.get('order_id')
    if not order_id:
        messages.error(request, 'Замовлення не знайдено')
        return redirect('cart')

    try:
        order = Order.objects.get(id=order_id, user=request.user)

        if order.payment_method != 'googlepay':
            messages.error(request, 'Неправильний метод оплати')
            return redirect('order_detail', order_id=order.id)

    except Order.DoesNotExist:
        raise Http404('Замовлення не знайдено')

def cart(request):
    cart_items = []
    subtotal = 0
    total = 0

    if 'cart' in request.session:
        cart_data = request.session['cart']
        for item_id, quantity in cart_data.items():
            try:
                service = Service.objects.get(id=item_id)
                item_total = service.price * quantity
                cart_items.append({
                    'id': item_id,
                    'service': service,
                    'quantity': quantity,
                    'total_price': item_total
                })
                subtotal += item_total
            except Service.DoesNotExist:
                pass

        total = subtotal

    context = {
```

```
        'cart_items': cart_items,
        'subtotal': subtotal,
        'total': total,
        'form_errors': form_errors if 'form_errors' in locals() else {}
    }

    return render(request, 'orders/cart.html', context)

@login_required_with_message(message="Для оформлення замовлення необхідно увійти в акаунт.")
def checkout(request):
    cart_items = []
    subtotal = 0
    total = 0

    if 'cart' in request.session:
        cart_data = request.session['cart']
        for item_id, quantity in cart_data.items():
            try:
                service = Service.objects.get(id=item_id)
                item_total = service.price * quantity
                cart_items.append({
                    'id': item_id,
                    'service': service,
                    'quantity': quantity,
                    'total_price': item_total
                })
                subtotal += item_total
            except Service.DoesNotExist:
                pass

        total = subtotal

    if not cart_items:
        messages.warning(request, 'Ваш кошик порожній. Додайте послуги перед оформленням замовлення.')
        return redirect('cart')

    user_profile = None
    phone = ''
    address = ''
    city = ''
    zip_code = ''

    try:
        user_profile = request.user.profile
        phone = user_profile.phone_number
        address = user_profile.address
        city = user_profile.city
        zip_code = user_profile.zip_code
    except:
```

```
        pass

    if request.method == 'POST':
        form_valid = True
        form_errors = {}

        first_name = request.POST.get('first_name', '').strip()
        last_name = request.POST.get('last_name', '').strip()
        email = request.POST.get('email', '').strip()
        phone = request.POST.get('phone', '').strip()
        address = request.POST.get('address', '').strip()
        city = request.POST.get('city', '').strip()
        zip_code = request.POST.get('zip', '').strip()
        delivery_method = request.POST.get('delivery_method', '')
        payment_method = request.POST.get('payment_method', '')
        save_info = 'save_info' in request.POST
        if form_valid and cart_items:
            order = Order(
                user=request.user,
                total_price=total,
                status='pending',
                first_name=first_name,
                last_name=last_name,
                email=email,
                phone=phone,
                address=address,
                city=city,
                zip_code=zip_code,
                delivery_method=delivery_method,
                payment_method=payment_method
            )
            order.save()

            for item in cart_items:
                order_item = OrderItem(
                    order=order,
                    service=item['service'],
                    quantity=item['quantity'],
                    unit_price=item['service'].price,
                    subtotal=item['total_price']
                )
                order_item.save()

            subject = 'Підтвердження замовлення'
            html_message = render_to_string('orders/email/order_confirmation.html', {
                'user': request.user,
                'cart_items': cart_items,
                'total': total,
                'first_name': first_name,
                'last_name': last_name,
                'order': order,
```

```
                'address': address,
                'city': city,
                'zip_code': zip_code,
                'phone': phone
            })
            plain_message = strip_tags(html_message)
            from_email = 'noreply@servicestore.com'
            to_email = email

            if payment_method == 'googlepay':
                request.session['order_id'] = order.id
                send_mail(subject, plain_message, from_email, [to_email], html_message=html_message)
                return redirect('payment')

            if 'cart' in request.session:
                del request.session['cart']
                request.session.modified = True

            send_mail(subject, plain_message, from_email, [to_email], html_message=html_message)

                messages.success(request, 'Ваше замовлення успішно оформлено! Підтвердження надіслано на вашу електронну пошту.')

            response = redirect('profile')
            response.set_cookie('active_tab', 'orders', max_age=30)
            return response
        else:
            for field, error in form_errors.items():
                messages.error(request, f'{error}')

    context = {
        'cart_items': cart_items,
        'subtotal': subtotal,
        'total': total,
        'user_phone': phone,
        'user_address': address,
        'user_city': city,
        'user_zip': zip_code,
        'form_errors': form_errors if 'form_errors' in locals() else {}
    }

    return render(request, 'orders/checkout.html', context)

@login_required_with_message(message="Для повторного замовлення необхідно увійти в акаунт.")
def reorder(request, order_id):
    order = get_object_or_404(Order, id=order_id, user=request.user)
    new_cart = {}
    for item in order.items.all():
        new_cart[str(item.service.id)] = item.quantity
```

```
    request.session['cart'] = new_cart
    request.session.modified = True

    return redirect('cart')


def service_detail(request, service_id):
    service = get_object_or_404(Service, id=service_id)
    reviews = service.reviews.all().order_by('-created_at')
    gallery_images = service.get_gallery_images()
    review_form = None
    user_review = None

    if request.user.is_authenticated:
        try:
            user_review = reviews.get(user=request.user)
        except Review.DoesNotExist:
            from .forms import ReviewForm
            review_form = ReviewForm()

    if request.method == 'POST' and request.user.is_authenticated:
        data = json.loads(request.body)
        action = data.get('action')

        if action == 'add_review':
            from .forms import ReviewForm
            from django.http import QueryDict

            # Створюємо QueryDict для правильної валідації форми
            post_data = QueryDict('', mutable=True)
            post_data.update({
                'rating': data.get('rating'),
                'comment': data.get('comment')
            })

            review_form = ReviewForm(post_data)
            if review_form.is_valid():
                review = Review()
                review.user = request.user
                review.service = service
                review.rating = data.get('rating')
                review.comment = data.get('comment')
                review.save()
                return JsonResponse({'status': 'success'})
            else:
                return JsonResponse({'status': 'error', 'errors': review_form.errors})

        elif action == 'edit_review':
            review_id = data.get('review_id')
            review = get_object_or_404(Review, id=review_id, user=request.user)
```

```
            review.rating = data.get('rating')
            review.comment = data.get('comment')
            review.save()
            return JsonResponse({'status': 'success'})

        elif action == 'delete_review':
            review_id = data.get('review_id')
            review = get_object_or_404(Review, id=review_id, user=request.user)
            review.delete()
            return JsonResponse({'status': 'success'})

    avg_rating = 0
    star_list = ['empty'] * 5

    if reviews.exists():
        total_rating = sum(review.rating for review in reviews)
        avg_rating = total_rating / reviews.count()

        full_stars = int(avg_rating)
        half_star = (avg_rating - full_stars) >= 0.25 and (avg_rating - full_stars) < 0.75
        empty_stars = 5 - full_stars - int(half_star)
        star_list = ['full'] * full_stars + ['half'] * int(half_star) + ['empty'] * empty_stars

    related_services = []
    if service.categories.exists():
        categories = service.categories.all()
                                                                related_services                =
Service.objects.filter(categories__in=categories).exclude(id=service.id).distinct()[:4]

    context = {
        'service': service,
        'reviews': reviews,
        'review_form': review_form,
        'user_review': user_review,
        'reviews_count': reviews.count(),
        'avg_rating': avg_rating,
        'star_list': star_list,
        'related_services': related_services,
        'gallery_images': gallery_images
    }

    return render(request, 'services/service_detail.html', context)

@login_required_with_message(message="Для додавання послуги до списку бажань необхідно
увійти в акаунт.")
def toggle_favorite(request, service_id):
    service = get_object_or_404(Service, id=service_id)
    user = request.user

    if service in user.favorite_services.all():
        user.favorite_services.remove(service)
```

```
            is_favorite = False
        else:
            user.favorite_services.add(service)
            is_favorite = True

        favorite_count = user.favorite_services.count()

        return JsonResponse({
            'status': 'success',
            'is_favorite': is_favorite,
            'favorite_count': favorite_count
        })

def check_auth_for_cart(request, service_id):
    """
    Перевіряє авторизацію користувача для додавання послуги до кошика.
    Якщо користувач не авторизований, перенаправляє на сторінку входу.
    """
    if not request.user.is_authenticated:
        login_url = reverse('login') + '?next=' + request.META.get('HTTP_REFERER', '/')
        return JsonResponse({
            'status': 'redirect',
            'redirect_url': login_url,
            'message': "Для додавання послуги до кошика необхідно увійти в акаунт."
        })

    return JsonResponse({
        'status': 'success',
        'is_authenticated': True
    })
```

Додаток Г

Логіка для панелі адміністратора

```
    class ServiceImageInline(admin.TabularInline):
    model = ServiceImage
    extra = 1
    fields = ('image', 'title', 'order', 'thumbnail')
    readonly_fields = ('thumbnail',)

    def thumbnail(self, obj):
        if obj.image and hasattr(obj.image, 'url'):
            return format_html('<img src="{}" width="50" height="50" style="object-fit: cover;" />',
obj.image.url)
        return "-"
    thumbnail.short_description = 'Перегляд'

class ServiceAdmin(admin.ModelAdmin):
    list_display = ('name', 'get_categories', 'price', 'view_gallery')
    list_filter = ('categories', 'price', 'is_active')
    search_fields = ('name', 'description')
    ordering = ('-price',)
    inlines = [ServiceImageInline]
    fieldsets = (
        ('Інформація про послугу', {
            'fields': ('name', 'description', 'price', 'categories', 'image', 'is_active')
        }),
    )

    def view_gallery(self, obj):
        count = obj.gallery_images.count()
        if count:
            return format_html('<a href="{}?service__id__exact={}">{} зображення</a>',
                               '/admin/services/serviceimage/', obj.id, count)
        return "Немає зображень"
    view_gallery.short_description = 'Галерея'

    def get_categories(self, obj):
        return ", ".join([category.name for category in obj.categories.all()])
    get_categories.short_description = 'Categories'

class ServiceImageAdmin(admin.ModelAdmin):
    list_display = ('thumbnail', 'service', 'title', 'order')
    list_filter = ('service',)
    search_fields = ('service__name', 'title')
    ordering = ('service', 'order')
    autocomplete_fields = ['service']

    fieldsets = (
```

```
        ('Зображення', {
            'fields': ('service', 'image', 'title', 'order')
        }),
    )

    def thumbnail(self, obj):
        if obj.image:
            return format_html('<img src="{}" width="50" height="50" style="object-fit: cover;" />',
obj.image.url)
        return "-"
    thumbnail.short_description = 'Мініатюра'

admin.site.register(Service, ServiceAdmin)
admin.site.register(Category)
admin.site.register(ServiceImage, ServiceImageAdmin)

class ProfileInline(admin.StackedInline):
    model = Profile
    can_delete = False

class CustomUserAdmin(UserAdmin):
    model = CustomUser
    inlines = (ProfileInline,)
    list_display = ('email', 'first_name', 'last_name', 'role', 'is_staff', 'is_active')
    list_filter = ('role', 'is_staff', 'is_active')
    fieldsets = (
        (_('Дані для входу'), {'fields': ('email', 'password')}),
        (_('Personal info'), {'fields': ('first_name', 'last_name')}),
        (_('Роль'), {'fields': ('role',)}),
        (_('Permissions'), {'fields': ('is_active', 'is_staff', 'is_superuser', 'user_permissions')}),
        (_('Important dates'), {'fields': ('last_login', 'date_joined')}),
    )
    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('email', 'first_name', 'last_name', 'role', 'password1', 'password2'),
        }),
    )
    search_fields = ('email', 'first_name', 'last_name')
    ordering = ('email',)

class AdminSiteForAdmins(AdminSite):
    site_header = _('Administrator Portal')
    site_title = _('Administrator Portal')
    index_title = _('User Management')

class AdminSiteForManagers(AdminSite):
    site_header = _('Manager Portal')
    site_title = _('Manager Portal')
    index_title = _('Service Management')
```

```
admin_site = AdminSiteForAdmins(name='admin_site')
manager_site = AdminSiteForManagers(name='manager_site')

class ReviewAdmin(admin.ModelAdmin):
    list_display = ('user', 'service', 'rating', 'created_at')
    list_filter = ('rating', 'created_at')
    search_fields = ('user__email', 'service__name', 'comment')
    readonly_fields = ('created_at',)
    fieldsets = (
        (_('Інформація про відгук'), {'fields': ('user', 'service')}),
        (_('Деталі відгуку'), {'fields': ('rating', 'comment')}),
        (_('Дата створення'), {'fields': ('created_at',)}),
    )

class OrderItemInline(admin.TabularInline):
    model = OrderItem
    extra = 1

class OrderAdmin(admin.ModelAdmin):
    list_display = ('order_number', 'uuid', 'user', 'order_date', 'total_price', 'status')
    list_filter = ('status', 'order_date')
    search_fields = ('order_number', 'uuid', 'user__email', 'first_name', 'last_name', 'email', 'phone')
    date_hierarchy = 'order_date'
    ordering = ('-order_date',)
    inlines = [OrderItemInline]

    fieldsets = (
        ('Інформація про замовлення', {
            'fields': ('order_number', 'uuid', 'user', 'total_price', 'first_name', 'last_name', 'email', 'phone')
        }),
        ('Інформація про доставку', {
            'fields': ('address', 'city', 'zip_code', 'delivery_method', 'payment_method')
        }),
        ('Інформація про статус', {
            'fields': ('status',)
        }),
    )

    readonly_fields = ('order_date', 'order_number', 'uuid')

class OrderItemAdmin(admin.ModelAdmin):
    list_display = ('order', 'service', 'quantity', 'unit_price', 'subtotal')
    list_filter = ('order__status',)
    search_fields = ('order__id', 'service__name')

admin.site.register(Order, OrderAdmin)
```