Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка мобільного застосунку «Книжковий магазин та бібліотека»

Виконав: студент 4 курсу
групи ЗПІ-21Б спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Яворський Б.М.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.
(прізвище та ініціали)
«06.06» 2025 р.

Рецензент:. к.т.н., доц. каф. ОТ Крупельницький Л.В.
(прізвище та ініціали)
«06.06» 2025 р.

Допущено до захисту
Зав. кафедри ПЗ
д.т.н., проф. О.Н. Романюк
(ініціали та прізвище)
«06» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Яворському Богдану Миколайовичу

1. Тема роботи – Розробка мобільного застосунку «Книжковий магазин та бібліотека»

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: операційна система – Android/IOS; середовище розробки – WebStorm, Xcode; мова програмування – JavaScript

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз розвитку технологій мобільних систем для книжкових магазинів та постановка задач дослідження; розробка моделей та алгоритмів програмного продукту; розробка структури та програмних модулів мобільної системи книжкового магазину; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В.П., к.т.н., доцент кафедри ПЗ	21.03.25	02.06.2025

7. Дата видачі завдання 21 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку технологій мобільних систем для книжкових магазинів та постановка задач дослідження	25.03.25-08.04.25	вик.
2	Розробка моделей та алгоритмів програмного продукту	09.04.25 - 20.04.25	вик.
3	Аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення	21.04.25 - 23.04.25	вик.
4	Розробка структури та програмних модулів мобільної системи книжкового магазину	24.04.25 - 20.05.25	вик.
5	Тестування застосунку	21.05.25 - 25.05.25	вик.
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25	вик.

Студент _____ **Яворський Б.М.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ **Майданюк В.П.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 55 сторінок формату А4, на яких є 33 рисунка, 3 таблиці, список використаних джерел містить 24 найменування.

Метою роботи є поєднання можливостей електронної комерції та цифрового читання в межах одного мобільного застосунку, що автоматизують та спрощують процеси вибору, оформлення та читання книг.

У бакалаврській кваліфікаційній роботі розроблено програмні засоби мобільного застосунку для онлайн-продажу та перегляду книжок. Застосунок призначений для користувачів мобільних пристроїв і надає можливість переглядати каталог книг, здійснювати покупки, а також зберігати обрані видання у персональній бібліотеці.

Отримані в бакалаврській роботі результати можуть бути використані для подальшого розвитку бізнес-мобільних застосунків щодо електронної комерції та цифрових продажів книжкової продукції.

Ключові слова: мобільний застосунок, книжковий магазин, бібліотека, електронна комерці

ABSTRACT

The bachelor's thesis consists of 55 A4 pages with 33 figures, 3 tables, and a bibliography of 24 references.

The goal is to combine e-commerce and digital reading capabilities within a single mobile application that automates and simplifies the process of choosing, ordering and reading books

In the bachelor's thesis, you developed software tools for a mobile application for online sales and viewing of books. The application is designed for users of mobile devices and allows them to browse the catalogue of books, make purchases, and save selected publications to their personal library.

The results obtained in the bachelor's thesis can be used for the further development of business mobile applications for e-commerce and digital sales of book products.

Keywords: mobile application, bookstore, bookshop, e-commerce.

ЗМІСТ

ВСТУП 4

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ МОБІЛЬНИХ СИСТЕМ ДЛЯ КНИЖКОВИХ МАГАЗИНІВ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 7

1.1 Аналіз сучасних технологій та тенденцій розвитку мобільних систем книжкових магазинів 7

1.2 Порівняльний аналіз аналогів 8

1.3 Аналіз методів розв’язання задачі 12

1.4 Постановка задач бакалаврської роботи 14

1.5 Висновки 15

2 РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ .. 17

2.1 Аналіз вхідних даних системи 17

2.2 Розробка інтерфейсу програмного додатку 18

2.3 Розробка моделі системи 20

2.4 Розробка алгоритмів роботи системи 22

2.5 Висновки 25

3. РОЗРОБКА СТРУКТУРИ ТА ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОЇ СИСТЕМИ КНИЖКОВОГО МАГАЗИНУ 26

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення 26

3.2 Розробка модуля авторизації та реєстрації користувачів 28

3.3 Розробка модуля управління кошиком та оформлення замовлень з інтегрованою підтримкою Apple Pay, Google Pay та банківських карток 31

3.4 Розробка особистого кабінета користувача з персональними даними..... 35

3.5 Розробка модуля для читання книг із підтримкою базових функцій перегляду..... 39
3.6 Висновки 42
4 ТЕСТУВАННЯ СИСТЕМИ 43
4.1 Тестування мобільного застосунку 43
4.2 Розробка інструкції користувача 54
4.3 Висновки 56
ВИСНОВКИ..... 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТОК А (обов’язковий). Технічне завдання на роботу 62
ДОДАТОК Б (обов’язковий). Протокол перевірки на наявність запозичень 66
ДОДАТОК В (довідниковий). Лістинг коду додатку 67
ДОДАТОК Г (обов’язковий). Ілюстративна частина 117

ВСТУП

Обґрунтування вибору теми дослідження. Мобільні технології стали невіддільною частиною повсякденного життя сучасної людини. Протягом останніх років електронна комерція швидко розвивається у всьому світі, забезпечуючи при цьому безпрецедентні засоби і можливості для розвитку світової торгівлі [1]. Одним із основних напрямів цього розвитку є перехід електронної комерції у мобільний формат, і книжкові онлайн-магазини активно долучаються до цієї тенденції. Тому розробка мобільного застосунку книжкового магазину стає актуальною та перспективною задачею.

Однією з головних проблем електронної комерції книг є забезпечення того, щоб користувачі з мобільними пристроями отримували швидкий і легкий доступ до широкого асортименту літератури. Крупні бренди створюють власні маркетплейси що створює конкуренцію з іншими eCommerce проектами та забирає в них частину ринку. [2]. Програмна система мобільного книжкового магазину, яка забезпечує зручний та інтуїтивно зрозумілий інтерфейс, відкриває нові горизонти у покращенні якості послуг, спрямованих на обслуговування клієнтів та максимізацію продажів. Окрім того, мобільні технології відкривають нові горизонти для персоналізації користувацького досвіду [3].

Актуальність розробки мобільної системи для книжкового магазину зумовлена зростаючим попитом на зручні цифрові сервіси для придбання та читання книг. Проблема ускладнюється тим, що попит на мобільний доступ до цифрової книжкової продукції стабільно зростає. Люди дедалі частіше шукають можливість купити й прочитати книгу в межах одного застосунку – швидко, без зайвих дій, на звичному їм пристрої. Таким чином, розробка мобільного застосунку для книжкового магазину є актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є поєднання можливостей електронної комерції та цифрового читання в межах одного мобільного застосунку, що автоматизують та спрощують процеси вибору, оформлення та читання книг, які дозволять зменшити час на пошук необхідних видань та підвищити точність рекомендацій та забезпечити зручний доступ до електронного контенту.

Основними задачами роботи є:

- провести аналіз аналогів і сформувати вимоги до мобільного застосунку;
- розробити швидкий алгоритм пошуку книг за назвою, автором, жанром та іншими критеріями;
- розробити функціонал авторизації та реєстрації користувачів з валідацією даних;
- розробити модуль формування кошика, який буде спроможний зберігати вибрані книги, відображати їх кількість та вартість;
- розробити модуль оформлення замовлення, який буде забезпечувати зручне введення даних для доставки та оплати;
- розробити модуль для можливості зберігання та читання електронних книг;
- провести тестування розробленого мобільного застосунку.

Об'єкт дослідження – процес розробки мобільного застосунку для книжкових магазинів та бібліотек.

Предмет дослідження – методи та засоби розробки мобільних застосунків для книжкових магазинів та бібліотек.

Методи дослідження. У процесі досліджень використовувались: теорія баз даних для організації зберігання інформації, принципи об'єктно-орієнтованого проектування для створення архітектури застосунку, технології розробки кросплатформних застосунків для створення єдиної кодової бази для різних мобільних платформ, методи функціонального тестування для перевірки працездатності застосунку

Новизна отриманих результатів.

1. Подальшого розвитку отримала система для книжкової торгівлі, у яку на відміну від існуючих, інтегровано модуль читання електронних книг з підтримкою базових функцій перегляду (гортання сторінок, масштабування), що виключає потребу в зовнішніх рідерах.

2. Подальшого розвитку отримала система обробки платежів, у якій на відміну існуючих, забезпечена підтримка інтеграції банківських карток, Apple Pay та Google Pay безпосередньо у мобільному застосунку, що підвищує зручність та безпеку здійснення оплат в процесі оформлення замовлення та не потребує переходу до сторонніх платіжних систем.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для автоматизації процесів купівлі та продажу книжкової продукції через мобільні пристрої.

Особистий внесок здобувача. Усі результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованій праці, опублікованій у співавторстві, автору належать такі результати: роль мобільних застосунків у популяризації читання: як цифрові технології можуть збільшити інтерес до книг [4].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи доповідались на «Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженері» (Вінниця, 2025).

Публікації. Основні результати досліджень було опубліковано на «Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженері» (Вінниця, 2025).

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ МОБІЛЬНИХ СИСТЕМ ДЛЯ КНИЖКОВИХ МАГАЗИНІВ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасних технологій та тенденцій розвитку мобільних систем книжкових магазинів

У сучасних умовах глобалізації та стрімкого розвитку цифрових технологій ринок книжкової торгівлі зазнає суттєвих трансформацій. Особливої актуальності набувають мобільні рішення, які забезпечують користувачам зручний доступ до літератури незалежно від місця та часу. За даними Statista, станом на 2023 рік понад 60% покупок у сфері електронної комерції здійснюється саме через мобільні пристрої [5].

У сучасному цифровому світі мобільні застосунки стали необхідним інструментом для розвитку бізнесу та полегшення нашого повсякденного життя. Як наслідок, конкуренція на цьому ринку надзвичайно велика. Якщо застосунок лишити малопомітним, він просто загубиться серед незліченних альтернатив [6]. Покупка книги тепер може здійснюватися за кілька хвилин – незалежно від місця перебування користувача, що суттєво підвищує зручність сервісу.

Мобільні системи для книжкових магазинів мають ряд переваг у порівнянні з традиційними форматами продажу:

- зручність – мобільні застосунки дозволяють користувачам здійснювати покупки в будь-який час і в будь-якому місці, використовуючи лише свій смартфон або планшет;
- персоналізація – сучасні мобільні системи можуть аналізувати дані про користувачів, такі як їхні попередні покупки та читацькі переваги, щоб пропонувати персоналізовані рекомендації та акції;
- інтерактивність – мобільні застосунки можуть включати інтерактивні елементи, такі як відгуки користувачів, рейтинги книг та можливість спілкування з іншими читачами.

Не менш важливим є розвиток інтеграцій мобільних систем із суміжними сервісами: платіжними шлюзами, службами доставки, соціальними мережами. Така інтеграція забезпечує безшовний користувацький досвід – від вибору книжки до її доставки на вказану адресу.

Отже, аналіз сучасних технологій і тенденцій розвитку мобільних систем у сфері книжкової торгівлі свідчить про їхню важливість і перспективність. Ефективна розробка таких систем дозволяє книжковим магазинам не тільки зберігати свою конкурентоспроможність на ринку, а й активно розширювати аудиторію шляхом залучення нових сегментів користувачів. Урахування сучасних технологічних трендів та своєчасне впровадження інновацій стають визначальними чинниками успішності у даній галузі.

1.2 Порівняльний аналіз аналогів

Ринок мобільних застосунків для книжкових магазинів постійно зростає, пропонуючи користувачам різноманітні функції та можливості. З кожним роком все більше людей використовують смартфони та планшети для пошуку та купівлі книг, що стимулює розвиток мобільних платформ для книжкової торгівлі. Аналіз цих застосунків дозволяє виявити ключові тенденції, переваги та недоліки, що є важливим для розробки власної конкурентоспроможної системи. До найбільш відомих мобільних застосунків належать:

- Yakaboo;
- Ridmi;
- PocketBook Reader.

Одним із прикладів є Yakaboo – це найбільша в Україні електронна бібліотека з більш ніж 20 000 електронних та аудіокнижок від відомих українських та світових авторів [7]. Мобільний застосунок Yakaboo надає зручний інтерфейс для пошуку та купівлі книг, можливість читати фрагменти книг перед покупкою, відстежувати новинки та акції, а також керувати своєю електронною бібліотекою. Yakaboo активно співпрацює з українськими

видавництвами, що забезпечує широкий вибір літератури українською мовою. Зображення мобільного застосунку зображено на рисунку 1.1.

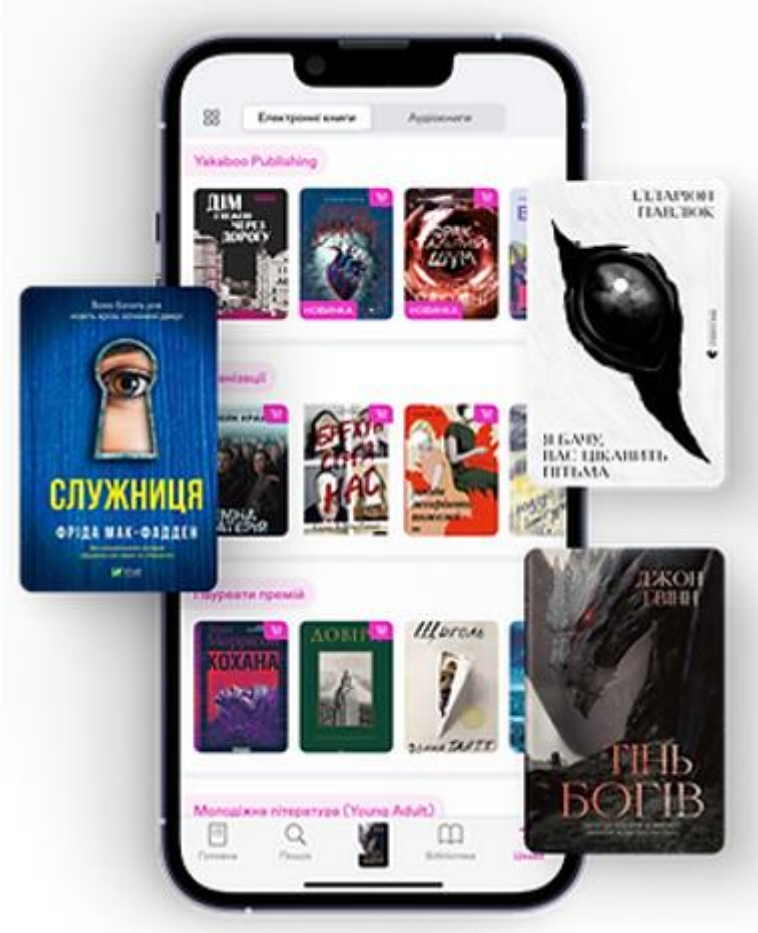


Рисунок 1.1 – Зображення мобільного застосунку «Yakaboo»

Ridmi – це не просто мобільний застосунок для читання електронних книг, а справжня соціальна платформа для книголюбів [8]. Він поєднує в собі зручну читалку з широкими можливостями соціальної взаємодії.

В Ridmi користувачі можуть створювати власні профілі, де вони діляться своїми читацькими вподобаннями, залишають відгуки на прочитані книги, оцінюють їх та беруть участь в обговореннях (рис. 1.2). Застосунок дозволяє додавати інших користувачів в друзі, слідкувати за оновленнями улюблених авторів та формувати власні спільноти за інтересами. Функція коментування книг дає можливість обмінюватися думками та враженнями, ділитися цитатами та аналізувати прочитане. Крім того, користувачі можуть створювати особисті

колекції книг за різними тематиками, що дуже зручно для систематизації власної бібліотеки.

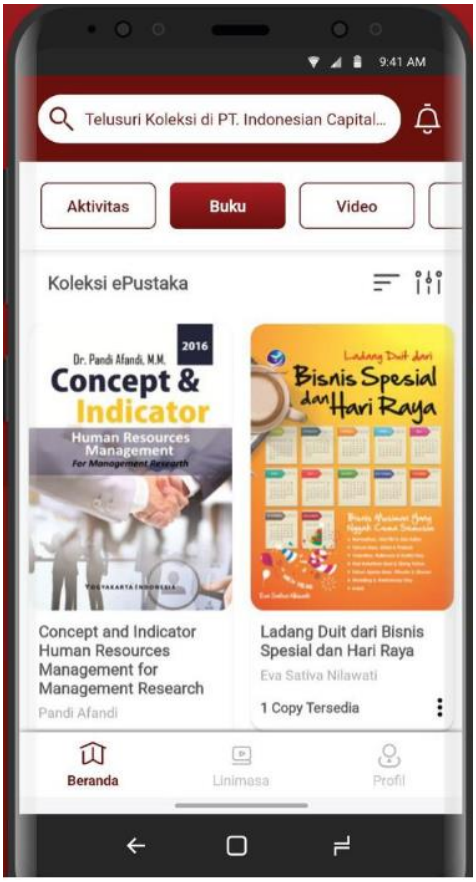


Рисунок 1.2 – Зображення мобільного застосунку «Ridmi»

PocketBook Reader – це багатофункціональний мобільний застосунок, розроблений компанією PocketBook, відомим виробником електронних рідеров [9]. Цей застосунок вирізняється своєю орієнтацією на максимально комфортний процес читання електронних книг, пропонуючи користувачам широкий спектр інструментів та налаштувань.

Однією з ключових особливостей PocketBook Reader є підтримка величезної кількості форматів електронних книг, включаючи популярні epub, fb2, mobi, а також більш рідкісні формати. Це дозволяє користувачам читати практично будь-яку електронну книгу без необхідності конвертації.

PocketBook Reader має інтегрований магазин електронних книг, що надає доступ до широкого асортименту літератури різних жанрів. Користувачі можуть

легко знаходити та купувати книги безпосередньо в застосунку. На рисунку 1.3 можна побачити мобільний застосунок PocketBook Reader.

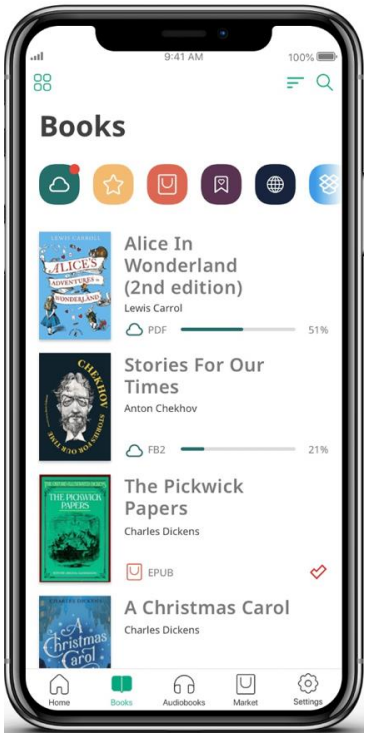


Рисунок 1.3 – Зображення мобільного застосунку «PocketBook Reader»

Розробка мобільної системи книжкового магазину вимагає ретельного аналізу конкурентів та визначення ключових факторів успіху. Порівняльна характеристика аналогів дозволяє виявити сильні та слабкі сторони кожного з них, а також визначити напрямки для розвитку власної системи.

Результати порівняння Yakaboo, Ridmi та PocketBook Reader зведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Функціональні можливості	Yakaboo	Ridmi	PocketBook	Власна розробка
Алгоритм підбору рекомендацій	1	0	1	1
Особистий кабінет	1	1	0	1
Відгуки та рейтинги	1	1	0	1

Продовження таблиці 1.1

Функціональні можливості	Yakaboo	Ridmi	PocketBook	Власна розробка
Бонусна система балів для купівлі книг	0	0	1	1
Фільтрація по жанрам книги	1	1	1	1
Сумарний коефіцієнт	4	3	3	5

Відповідно до таблиці порівняльних характеристик, розробка власного мобільного застосунку для книжкового магазину є доцільною та перспективною. аналіз існуючих рішень показав, що жоден з аналогів не поєднує в собі всі необхідні функції та переваги, які планується реалізувати у власному застосунку.

Отже, розробка власного мобільного застосунку для книжкового магазину відкриває широкі можливості для розвитку бізнесу, залучення нових клієнтів та підвищення лояльності існуючих. Завдяки унікальному поєднанню функцій та переваг, власний застосунок може стати ефективним інструментом для просування книг та розвитку культури читання.

1.3 Аналіз методів розв’язання задачі

Розробка мобільної системи для книжкового магазину потребує комплексного підходу, що охоплює не лише побудову інтерфейсу користувача, а й ефективну обробку даних, управління замовленнями та забезпечення безпеки інформації. Вибір відповідного технологічного стеку має базуватись на критеріях масштабованості, продуктивності, зручності супроводу, а також відповідності вимогам крос-платформності.

Одним з основних викликів є вибір системи управління базами даних (СУБД). Серед можливих варіантів розглядалися реляційні (PostgreSQL, MySQL) та документоорієнтовані (MongoDB, Couchbase) рішення. У даному випадку доцільним є використання MongoDB — NoSQL СУБД, що зберігає дані

у форматі BSON (розширення JSON). Вона забезпечує гнучку схему зберігання, що є перевагою при роботі з неоднорідною інформацією про книги, користувачів та транзакції [10].

MongoDB підтримує горизонтальне масштабування, має високу швидкодію при виконанні операцій читання та запису, а також дозволяє легко модифікувати структуру документів без потреби в складних міграціях [11]. Саме ці характеристики є критичними при розробці динамічного застосунку, де структура даних може змінюватися на етапі розвитку продукту.

Для реалізації серверної логіки обрано Nest.js — прогресивний фреймворк на базі Node.js і TypeScript. Порівняно з Express.js, Nest.js забезпечує модульну архітектуру, що значно покращує підтримуваність коду в довгостроковій перспективі. Додатковою перевагою є вбудована підтримка інжекції залежностей, middleware, а також WebSocket-комунікацій, що дозволяє реалізовувати функції реального часу — наприклад, сповіщення про статус замовлення чи нові акції. Використання TypeScript у середовищі Nest.js дає змогу виявляти помилки ще на етапі компіляції, що підвищує загальну стабільність системи.

У якості фронтенд-фреймворку використовується Angular, який забезпечує розробку на основі компонентно-орієнтованої архітектури. Angular має вбудовану систему маршрутизації, шаблонів, а також інструменти для двостороннього зв’язку між даними та інтерфейсом. Завдяки бібліотеці RxJS, Angular дозволяє зручно працювати з асинхронними потоками, що є особливо актуальним для мобільних застосунків, де часто використовуються запити до API, події користувача, динамічне завантаження контенту тощо.

Альтернативою Angular могли б виступати React Native або Flutter. Однак, Angular було обрано з огляду на повну інтеграцію з існуючим бекендом, TypeScript-основу та гнучкість в інтеграції з іншими інструментами веб-розробки.

Для забезпечення крос-платформності мобільного застосунку доцільно використовувати фреймворк Capacitor, який дозволяє розгортати Angular-

додаток як нативний застосунок для iOS та Android [12]. На відміну від Cordova, Capacitor забезпечує кращу інтеграцію з сучасними API платформ, зокрема доступ до камери, геолокації, push-повідомлень тощо. Завдяки Capacitor можлива єдина кодова база з доступом до нативних функцій, що зменшує витрати на розробку та полегшує оновлення застосунку.

Проведений аналіз дозволив визначити найбільш доцільну архітектурну модель для розробки мобільної системи книжкового магазину. Вибір стеку технологій, що включає MongoDB, Nest.js, Angular і Capacitor, є обґрунтованим з огляду на вимоги до масштабованості, крос-платформності, гнучкості у зміні структури даних, а також високої продуктивності та безпеки. Такий підхід дозволяє реалізувати сучасний, стабільний та доступний продукт для кінцевих користувачів.

1.4 Постановка задач бакалаврської роботи

Метою даної роботи є розробка мобільного застосунку для книжкового магазину, який забезпечує зручну взаємодію користувача з каталогом книг, можливість оформлення замовлень, перегляд персональної інформації та інтеграцію з системами обліку. На основі попереднього аналізу технологій та порівняння аналогів, було сформульовано перелік функціональних та технічних завдань, що мають бути реалізовані у межах проєкту.

До основних задач роботи належать:

1. Реалізація функціоналу авторизації та реєстрації користувачів з валідацією даних. Необхідно розробити надійну систему автентифікації, яка забезпечить безпечний доступ користувачів до мобільного застосунку.

2. Розробка функціоналу для перегляду каталогу книг з функціями пошуку та фільтрації. Слід створити зручний інтерфейс для перегляду каталогу книг, який дозволить користувачам швидко знаходити необхідні видання.

3. Розробка функціоналу управління кошиком та оформлення замовлень. Важливо забезпечити можливість додавання книг до кошика, редагування його

вмісту та оформлення замовлень. Також необхідно розробити систему обліку замовлень, яка дозволить користувачам відстежувати статус їхніх замовлень.

4. Розробка особистого кабінету користувача з персональними даними. Кожен користувач повинен мати доступ до свого особистого кабінету, де він зможе переглядати та редагувати свої персональні дані (адреса доставки, історія замовлень, персональні рекомендації тощо).

5. Реалізація модуля для читання книг у застосунку. Необхідно забезпечити можливість відкривати електронні книги безпосередньо у мобільному застосунку. Система має зберігати прогрес читання для кожного користувача.

6. Здійснення тестування програмного забезпечення згідно поставлених задач. Після реалізації основних функціональних можливостей необхідно провести комплексне тестування мобільного застосунку для виявлення та виправлення можливих помилок та недоліків. Тестування повинно включати перевірку коректності роботи всіх функцій, зручність використання інтерфейсу та безпеку системи.

Таким чином, сформульовані задачі визначають основні напрямки реалізації програмного продукту відповідно до його функціонального призначення, технічних вимог і очікувань кінцевих користувачів.

1.5 Висновки

Метою першого розділу було обґрунтування доцільності розробки мобільного застосунку для книжкового магазину та формулювання основних технічних і функціональних вимог до системи.

У процесі аналізу сучасного стану ринку мобільних книжкових сервісів було встановлено, що користувачі очікують від таких застосунків не лише широкий вибір літератури, а й високий рівень персоналізації, інтеграцію з платіжними системами, підтримку соціальної взаємодії та простий у використанні інтерфейс.

Порівняльний аналіз аналогів, зокрема Yakaboo, Ridmi та PocketBook Reader, дозволив виявити наявні функціональні обмеження існуючих рішень та

сформувати перелік необхідних можливостей, які доцільно реалізувати у власному продукті.

Було проаналізовано методи реалізації мобільної системи, зокрема технічний стек, що включає MongoDB, Nest.js, Angular та Capacitor. Зазначене поєднання технологій забезпечує крос-платформну доступність, масштабованість і гнучкість у розробці застосунку.

На основі проведеного аналізу було сформульовано задачі, що включають створення системи автентифікації, реалізацію функціоналу перегляду каталогу, управління кошиком, оформлення замовлень, особистого кабінету користувача та модулів тестування.

Таким чином, результати першого розділу підтверджують доцільність створення власного мобільного застосунку, який відповідатиме сучасним вимогам до електронної комерції у сфері книжкової торгівлі та забезпечить користувачам зручні, безпечні та ефективні засоби взаємодії з сервісом.

2 РОЗРОБКА МОДЕЛЕЙ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

У процесі функціонування мобільного застосунку книжкового магазину система отримує значний обсяг вхідної інформації, яка є основою для взаємодії користувача з інтерфейсом, базою даних та логікою обробки замовлень. Вхідні дані – це не лише заповнення форм, а й усі дії користувача, що ініціюють запити до сервера та формують інформаційні потоки в системі. Вони опрацьовуються в рамках клієнтсько-серверної архітектури із використанням Angular на фронтенді, NestJS на бекенді та MongoDB як основної бази даних. Така технологічна зв'язка є типовою для сучасних високонавантажених додатків.

На клієнтському рівні користувач взаємодіє з формами реєстрації та входу, персональними налаштуваннями, пошуковим інтерфейсом та каталогом книг. Усі ці дії генерують вхідні дані, які надсилаються до серверу для перевірки та подальшої обробки. Важливо, що дані користувача включають як стандартну реєстраційну інформацію, так і додаткові параметри: історію переглядів, обрані книги, адреси доставки, а також уподобання щодо візуального оформлення інтерфейсу. Усі ці параметри зберігаються або в локальному сховищі пристрою, або синхронізуються з сервером через REST API [13].

Інформація про книги охоплює ключові атрибути: назву, автора, опис, жанр, рік видання, ціну, рейтинг та статус наявності. Ці відомості, а також супровідні метадані, формують основу для пошуку, фільтрації та виводу детальних сторінок книги в застосунку.

Коли користувач додає товар до кошика і переходить до оформлення замовлення, формується набір вхідних даних, що охоплює перелік обраних позицій, кількість одиниць, обраний спосіб доставки та платіжну інформацію. Ці дані надходять до серверу, де зберігаються в базі даних і обробляються модулем керування замовленнями. Особливу увагу слід приділити статусу замовлення: усі зміни його стану (наприклад, «у обробці», «доставляється», «доставлено»,

«скасовано») відбуваються виключно на серверній частині системи, відповідно до внутрішньої логіки обробки. Користувач отримує лише результат – оновлене значення статусу.

Окрему групу вхідних даних становлять взаємодії користувача, які не мають прямого відношення до транзакцій: пошукові запити, фільтрація в каталозі, відгуки тощо. Хоча такі дані не обов’язково зберігаються в базі, вони можуть використовуватися для аналітики та побудови персоналізованих рекомендацій, що є типовою практикою в сучасних комерційних застосунках.

Таким чином, аналіз вхідних даних засвідчив наявність декількох логічних потоків інформації: персональних, транзакційних, контентних та поведінкових. Їхнє правильне моделювання, обробка та збереження є ключовими передумовами для реалізації повноцінної функціональності мобільного застосунку, який буде відповідати вимогам сучасного користувача та забезпечить високу якість сервісу.

2.2 Розробка інтерфейсу програмного додатку

Інтерфейс користувача є критично важливою складовою будь-якого програмного забезпечення. Коли йдеться про мобільні застосунки – обмежений екранний простір вимагає максимальної лаконічності, інтуїтивності та доступності [14]. З огляду на це було обрано технологічну зв’язку Angular і Capacitor, що забезпечує кросплатформну підтримку, та використано компоненти Angular Material та Ionic UI для дотримання принципів Material Design.

На першому етапі було визначено ключові екрани застосунку: головна сторінка з каталогом книг, сторінка деталей книги, форма авторизації та реєстрації, особистий кабінет, а також пошук і фільтрація. Кожен екран розроблявся у відповідності до сучасних гайдлайнів Google Material Design, із використанням компонентів Angular Material та Ionic UI для створення єдиної

візуальної мови. На рисунку 2.1 представлено зовнішній вигляд головної сторінки застосунку, що містить перелік книг у вигляді карток.

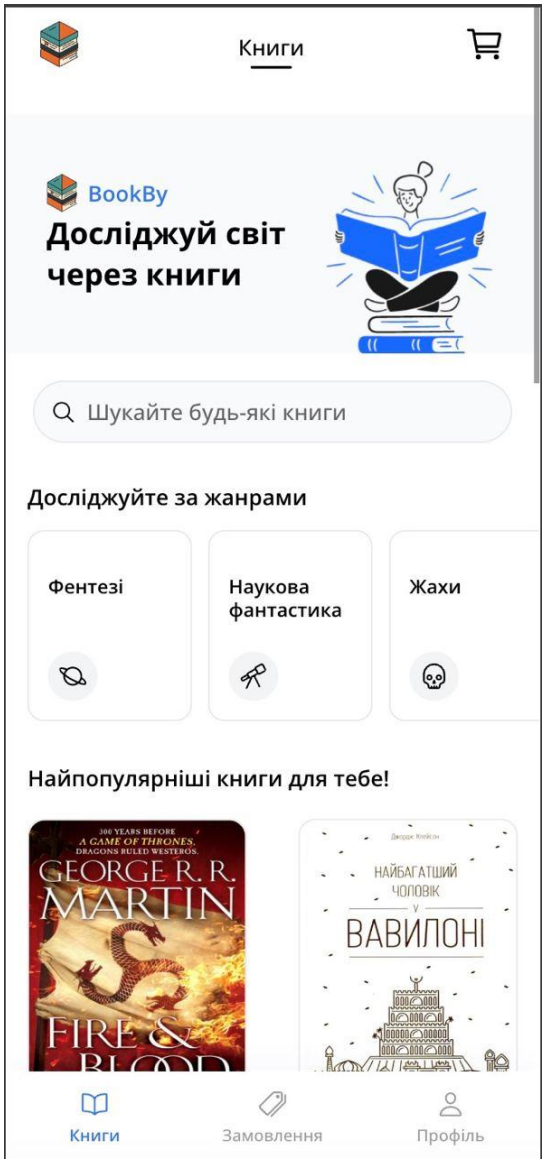


Рисунок 2.1 – Інтерфейс головної сторінки

Взаємодія між компонентами інтерфейсу забезпечується через модулі та сервісну архітектуру Angular. Наприклад, при відкритті сторінки книги її дані автоматично завантажуються через асинхронний сервіс, що отримує інформацію із серверної частини за допомогою HTTP-запитів. Завдяки цьому інтерфейс завжди працює з актуальними даними без потреби в ручному оновленні.

Авторизація та реєстрація реалізовані у вигляді окремих форм, з підтримкою валідації введених даних у реальному часі. Наприклад, при введенні некоректного email або слабкого пароля користувач одразу бачить відповідне повідомлення про помилку. Це підвищує якість взаємодії та зменшує кількість технічних помилок з боку клієнта.

Загалом, інтерфейс програмного додатку розроблявся із урахуванням потреб кінцевого користувача, забезпечуючи не лише функціональність, а й комфортну взаємодію, візуальну привабливість і відповідність сучасним стандартам мобільного UX-дизайну. Результатом є інтуїтивний і гнучкий інтерфейс, який легко масштабувати, модифікувати та підтримувати в межах розвитку застосунку.

2.3 Розробка моделі системи

Розробка моделі системи мобільного застосунку книжкового магазину передбачає побудову UML-діаграми, що описує основні процеси взаємодії користувача із застосунком (рис. 2.3). Відповідно до розробленої моделі, користувач розпочинає роботу із застосунком шляхом автентифікації, що забезпечує безпечний доступ до персоналізованих функцій системи. Після входу користувач отримує доступ до каталогу книг, де може здійснювати пошук за назвою, автором або жанром.

При виборі конкретної книги відображається детальна інформація, включаючи опис, рейтинг та відгуки інших користувачів. Якщо користувач вирішує зберегти книгу для подальшого перегляду, вона додається до списку вподобань. При бажанні переглянути вже додані книги він може перейти до відповідного розділу застосунку.

У разі оформлення замовлення система зберігає дані про вибрані книги та створює запис із деталями замовлення. Після підтвердження користувач отримує інформацію про статус обробки. Якщо замовлення сформоване, система дозволяє користувачу переглянути його стан у спеціальному розділі.

Під час взаємодії із застосунком система обробляє запити користувачів, звертаючись до серверної частини, реалізованої на NestJS, яка забезпечує отримання актуальних даних з бази. Завдяки використанню Capacitor, застосунок може отримувати доступ до деяких можливостей пристрою, таких як збереження даних у локальному сховищі для автономного режиму роботи.

Після завершення сесії користувач може вийти із застосунку, що закриває поточний сеанс та очищує тимчасові дані. Розробка діаграми діяльності була виконана на веб-сайті draw.io [15].

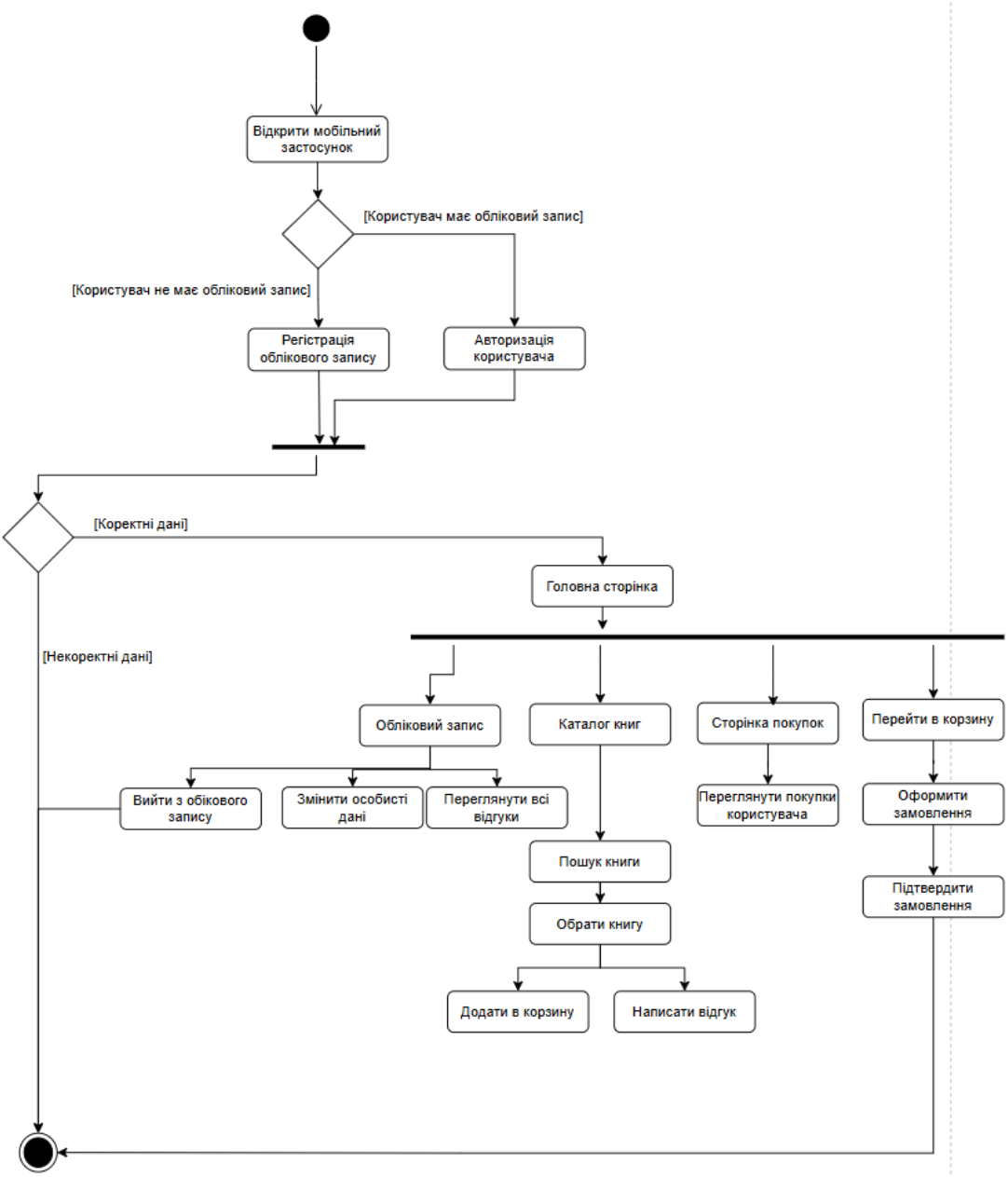


Рисунок 2.3 – Діаграма діяльності системи

Таким чином, побудована модель описує типові сценарії використання мобільного застосунку, що дозволяє не лише проєктувати інтерфейсні рішення, а й формалізувати бізнес-логіку системи у вигляді послідовних етапів взаємодії з цифровим сервісом. Це створює передумови для структурованої реалізації програмного забезпечення, мінімізує неузгодженості між клієнтською та серверною логікою та забезпечує масштабованість системи.

2.4 Розробка алгоритмів роботи системи

Для розробки мобільного застосунку книжкового магазину необхідно визначити основні алгоритми роботи системи, які забезпечать коректну взаємодію користувачів із функціоналом. Загальний алгоритм роботи системи охоплює процеси авторизації, взаємодії з каталогом та оформлення замовлення (рис. 2.4).

На початку взаємодії із застосунком відбувається перевірка наявності активного сеансу. У разі його відсутності система пропонує пройти процес реєстрації, що передбачає введення базових персональних даних, таких як електронна пошта, ім’я та пароль. Після надсилання форми дані передаються до серверної частини, де перевіряються на унікальність та коректність. У випадку помилки користувач отримує повідомлення з поясненням причини.

Успішна автентифікація дозволяє перейти до головної сторінки, де доступний каталог книг. Користувач має змогу здійснювати пошук за ключовими словами або застосовувати фільтри. Усі запити надсилаються до API, після чого результати миттєво відображаються на екрані. При натисканні на картку книги відкривається детальна інформація з описом, рейтингом, ціною та іншими метаданими. У разі зацікавлення користувач може зберегти книгу або додати її до кошика.

Взаємодія між клієнтською та серверною частинами відбувається за допомогою HTTP-запитів, організованих через Angular-сервіси. Завдяки використанню Capacitor, застосунок має можливість локально кешувати дані,

забезпечуючи часткову автономність у разі нестабільного підключення до мережі.

Окрім стандартної взаємодії з сервером, застосунок також використовує механізми обробки помилок, що дозволяють інформувати користувача про збої на сервері. Для цього реалізовано систему сповіщень, яка надає відповідні повідомлення в інтерфейсі. Усі дані, що надходять від користувача, проходять попередню валідацію на клієнтському рівні, що зменшує навантаження на сервер. Для підвищення продуктивності застосовано lazy loading модулів, що дозволяє завантажувати лише необхідні компоненти інтерфейсу в момент їх виклику. Це значно скорочує час відгуку та покращує загальну ефективність роботи мобільного застосунку.

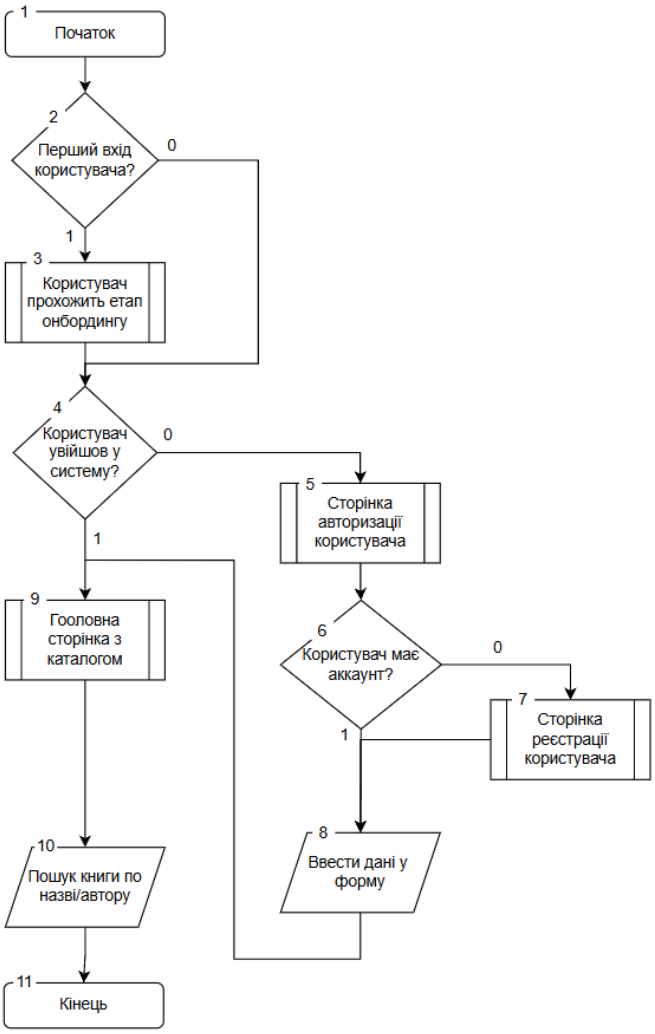


Рисунок 2.4– Алгоритм взаємодії з головною сторінкою застосунку.

Оформлення замовлення передбачає додавання книг у кошик, підтвердження вибору та внесення необхідної контактної інформації (рис 2.5). Після підтвердження даних замовлення зберігається в базі даних, а користувач отримує статус про його обробку. Подальший перегляд стану замовлення здійснюється у відповідному розділі, де система відображає актуальну інформацію про статус виконання.

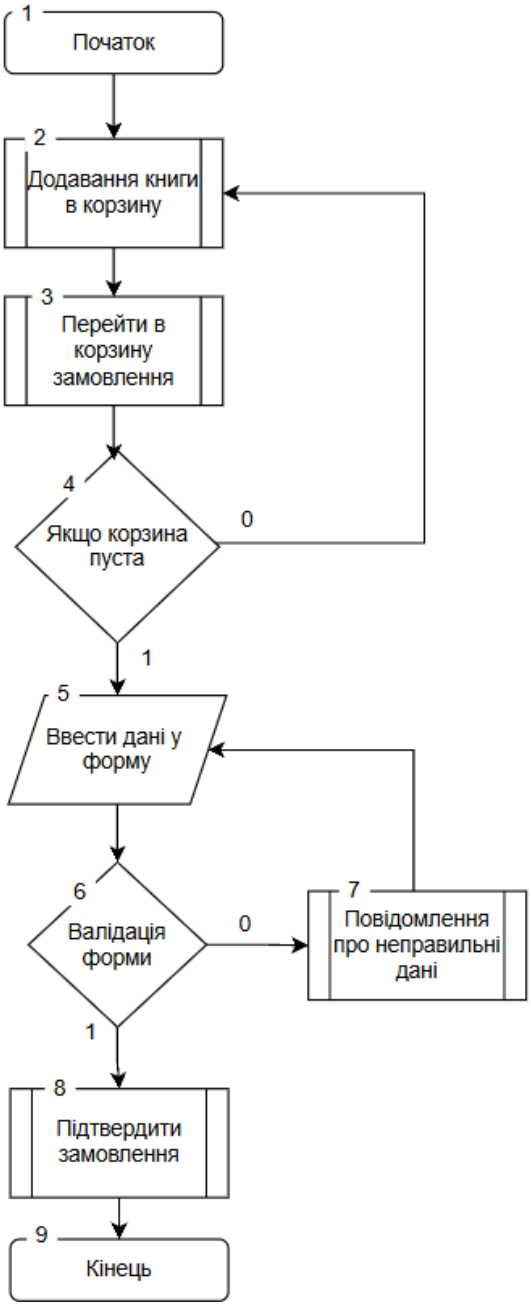


Рисунок 2.5 – Алгоритм оформлення замовлення

Таким чином, реалізація розглянутих алгоритмів забезпечує безперебійну роботу мобільного застосунку книжкового магазину, дозволяючи користувачам проходити реєстрацію, взаємодіяти з каталогом книг та здійснювати замовлення. Застосування Angular для клієнтської частини, NestJS для серверної логіки та Capacitor для інтеграції з мобільними платформами гарантує високу продуктивність і зручність використання системи.

2.5 Висновки

У даному розділі розглянуто основні алгоритми роботи мобільного застосунку книжкового магазину, що забезпечують коректну взаємодію користувачів із системою. Детально описано механізм авторизації та реєстрації, а також доступ до каталогу книг і можливості пошуку, фільтрації та збереження вподобаного. Реалізовано функціонал персоналізації, який дозволяє зберігати налаштування користувача, що підвищує зручність використання сервісу. Побудовані UML-діаграми представили логіку взаємодії між основними компонентами системи та формалізувати бізнес-процеси.

3. РОЗРОБКА СТРУКТУРИ ТА ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОЇ СИСТЕМИ КНИЖКОВОГО МАГАЗИНУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При створенні програмних модулів мобільної системи книжкового магазину критично важливим є вибір відповідних технологій. Вибрані засоби повинні забезпечувати ефективність, масштабованість, простоту підтримки, а також гарантувати інтуїтивну взаємодію користувача із системою. Це має вирішальне значення для досягнення комерційного успіху, тривалого функціонування застосунку та зручності його підтримки.

Для розробки фронтенд-частини мобільного застосунку був здійснений аналіз кількох сучасних JavaScript-фреймворків: Angular, React та Vue.js. Було складено табличку 1, що порівнює головні відмінності фрейморвків.

Таблиця 3.1 – Порівняння основних характеристик фреймворків

Критерій порівняння	**Angular**	**React**	**Vue.js**
Модульність	Висока	Середня	Середня
Типізація (TypeScript)	Вбудована	Доступна	Доступна
Підтримка великих проєктів	Оптимальна	Підходить	Менш підходить
Підтримка спільноти	Висока	Висока	Середня

В результаті аналізу було обрано Angular, оскаржуючи його переваги, серед яких модульна архітектура, компонентний підхід, широкі можливості

типізації через TypeScript, а також висока продуктивність [16]. Angular забезпечує можливість ефективної організації складних інтерфейсів, зокрема каталогів продукції, системи замовлення та управління кошиком покупок, що критично важливо для якісного користувацького досвіду. Завдяки можливостям цього фреймворку користувачі легко знаходять книги, отримують повну інформацію про них та здійснюють покупки без зайвих складнощів.

Щоб забезпечити крос-платформність мобільного застосунку та максимально ефективно використовувати наявні ресурси розробки, було обрано Capacitor – рішення, що дозволяє запускати веб-додатки в якості нативних застосунків на платформах iOS та Android. Порівняно з альтернативами, такими як React Native та Flutter, Capacitor має вагомі переваги: можливість повторного використання написаного коду, легкість інтеграції з веб-фреймворками (у тому числі Angular), а також зручний доступ до нативних функцій пристроїв (камера, геолокація, локальне сховище). Завдяки цьому можна забезпечити такі функції, як оплата без виходу із застосунку чи збереження персоналізованих налаштувань користувача, що підвищує конкурентоспроможність мобільного рішення.

Для серверної частини мобільного застосунку було здійснено порівняльний аналіз технологій Node.js, Django (Python) та Spring Boot (Java). За результатами вибору перевагу отримав Nest.js – сучасний серверний фреймворк, заснований на Node.js та TypeScript. Він вирізняється чіткою модульною структурою, високою продуктивністю, зручністю тестування та масштабування. Використання Nest.js дозволяє створити надійний API для обробки запитів мобільних клієнтів, організації ефективного управління даними книжкового магазину та реалізації складної бізнес-логіки, включаючи оплату замовлень, управління акаунтами користувачів та збереження даних у безпечному вигляді. Важливим аспектом також є простота взаємодії з різними базами даних, що спрощує роботу з інформацією про книги, замовлення та клієнтів.

Отже, вибір технологій Angular для фронтенду, Capacitor для забезпечення крос-платформності та Nest.js для серверної частини є повністю обґрунтованим. Цей комплекс технологій максимально відповідає специфічним бізнес-вимогам

до мобільного книжкового магазину, забезпечує ефективність розробки, високу продуктивність кінцевого продукту, можливість подальшого розвитку і підтримки застосунку, а також значні переваги з точки зору зручності для кінцевого користувача.

3.2 Розробка модуля авторизації та реєстрації користувачів

Модуль авторизації та реєстрації користувачів є ключовим компонентом мобільної системи книжкового магазину. Його основне завдання – забезпечити безпечний доступ користувачів до функціоналу застосунку, а також коректну обробку їхніх облікових даних.

Модуль реалізований у вигляді окремого компонента в Angular, що взаємодіє з бекендом, побудованим на NestJS. Для збереження даних використовується база даних, а аутентифікація реалізована через JWT (JSON Web Token).

Основні функції модуля:

- реєстрація користувача, що включає введення даних та їх перевірка на відповідність заданим вимогам;
- авторизація що перевіряє облікові дані користувача та отримує токен JWT;
- валідація даних – використання вбудованих механізмів валідації Angular для перевірки введених даних.

Розглянемо форму реєстрації користувача. Інтерфейс включає форму з трьома полями – ім’я, email та пароль – і включає дві кнопки, зареєструватися та повернутися до входу. Для зручності користувача передбачена можливість показу/приховування пароля, що підвищує безпеку введення даних у публічному просторі. Всі поля мають вбудовану індикативну валідацію – при неправильному введенні даних користувач отримує зрозуміле текстове повідомлення про помилку. Інтерфейс реєстрації зображений на рисунку 3.1

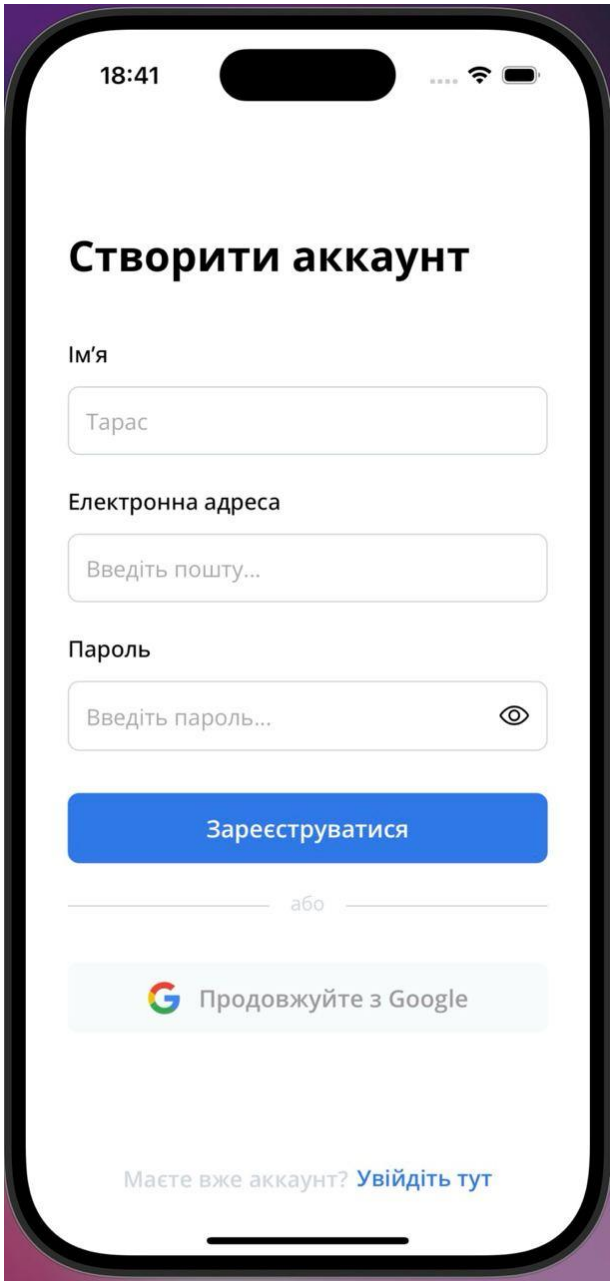


Рисунок 3.1 – Інтерфейс реєстрації

Також існує форма входу користувача, при наявності вже існуючого облікового запису. Цей інтерфейс також включає валідацію, і містить два поля – email та пароль. Алгоритм входу в обліковий запис користувача зображено на рисунку 3.2.

Рисунок 3.2 – Алгоритм входу в обліковий запис користувача

На бекенді валідація реалізована за допомогою class-validator у NestJS. Введені користувачем дані проходять додаткову перевірку перед їх збереженням у базі. Для захисту переданих даних використовується bcrypt, який хешує паролі перед збереженням. Це забезпечує безпеку в разі компрометації бази даних. Зображення фрагменту коду що відповідає створенню користувача зображений на рисунку 3.3

```
async signUp(signupData: SignupDto): Promise<{ message: string }> {
  const { email, password, name } = signupData;
  const emailInUse : (Document<unknown, {}, User> & User & Req...  = await this.userModel.findOne({ email });

  if (emailInUse) {
    throw new BadRequestException('Email already in use');
  }

  const hashedPassword : string  = await bcrypt.hash(password, 10);

  await this.userModel.create({ name, email, password: hashedPassword });

  return { message: 'success' };
}
```

Рисунок 3.3 – Логіка створення користувача на серверній стороні

Реалізований модуль авторизації та реєстрації дозволяє користувачам безпечно створювати облікові записи та входити в систему. Вбудована валідація значно зменшує навантаження на сервер, а зручні підказки покращують взаємодію користувачів із застосунком

3.3 Розробка модуля управління кошиком та оформлення замовлень з інтегрованою підтримкою Apple Pay, Google Pay та банківських карток

Модуль управління кошиком і оформлення замовлень є важливим компонентом мобільної системи книжкового магазину. Він забезпечує збереження вибраних користувачем товарів, їхню модифікацію та подальше оформлення замовлення через інтеграцію з API «Нова Пошта» для вибору міста та відділення доставки.

Модуль складається з двох основних частин:

– кошик – локальне збереження вибраних товарів та управління їхньою кількістю;

– оформлення замовлення – вибір способу доставки, введення контактних даних та інтеграція з API "Нова Пошта" для пошуку міст і відділень.

При розробці кошика, було приділено увагу простоті та швидкості доступу до основних дій, таких як: зміна кількості, видалення товару. Кожна книжка представлена у вигляді окремої картки з назвою, обкладинкою, кількістю екземплярів та ціною. Кнопки керування мають виразну іконографіку, що зменшує когнітивне навантаження на користувача. На рисунку 3.4 можна побачити інтерфейс кошика. Оновлення кошику відбувається в реально часі, без перезавантаження застосунку. На головній сторінці та на сторінці деталі про книги, у верхньому-правому кутку знаходиться іконка корзинки з кількістю обраних товарів, при натисненні на цю іконку, користувач попадає на сторінку оформлення замовлення, де може переглянути весь каталог, який був обраний та оформити замовлення.

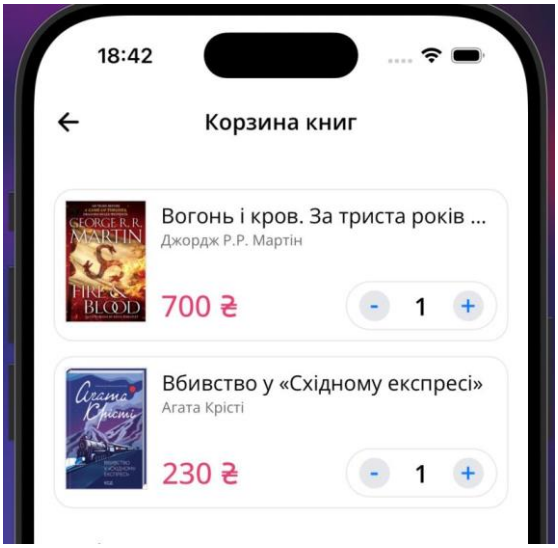


Рисунок 3.4 – Інтерфейс кошика застосунку

Для збереження книг та даних використовується LocalStorage, що дозволяє зберігати вміст кошика навіть після закриття застосунку, зберігаючи цілісність покупки та підвищуючи лояльність користувача.

Форма оформлення замовлення структурована так, щоби забезпечити всі необхідні для доставки й оплати дані (див. рисунок. 3.5).

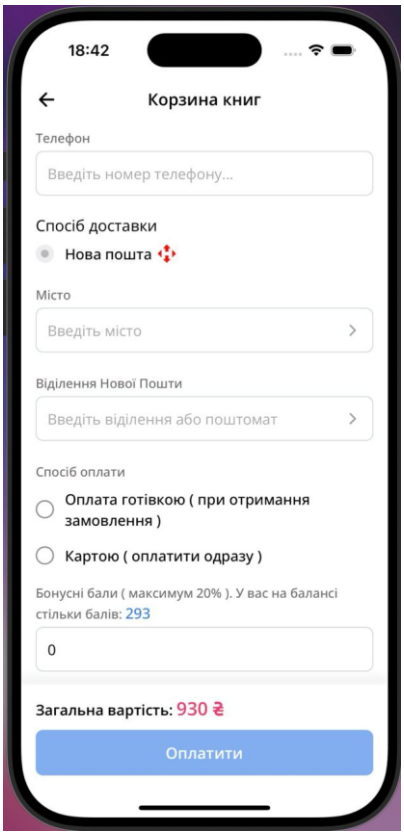


Рисунок 3.5 – Інтерфейс оформлення замовлення

Інтерфейс поділяє оформлення на три логічні блока:

- контактна інформація;
- вибір доставки та відділення;
- спосіб оплати та бонуси.

Вибір міста та відділення реалізовано через autocomplete. Користувач починає вводити назву, система автоматично підтягує список населених пунктів з API «Нова Пошта» (див. рисунок 3.6). Аналогічно реалізовано вибір відділення після підтвердження міста. Список результатів відображається у вигляді випадаючого списку зі скролом, що зручно для мобільних пристроїв.

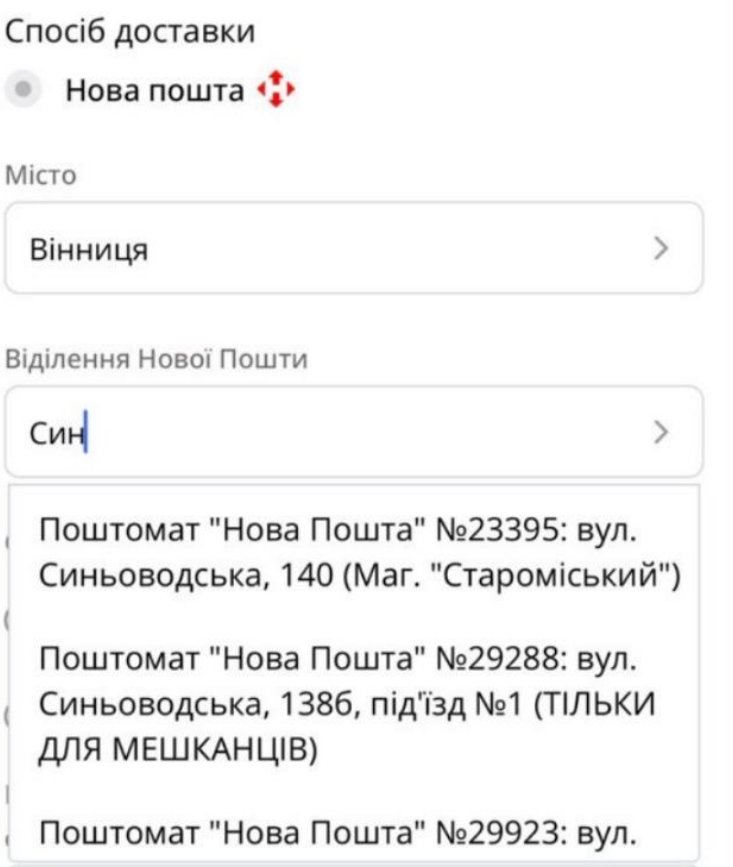


Рисунок 3.6 – Autocomplete після введення відділення

Також реалізовано бонусну систему у вигляді простого поля з автоматичним розрахунком максимально можливої суми списання, що зображено на рисунку 3.7.

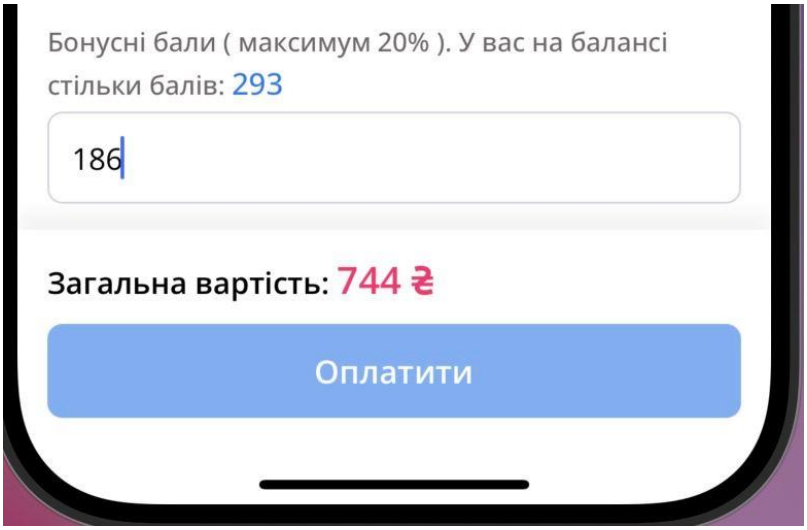


Рисунок 3.7 – Інтерфейс бонусної системи

Користувач може ввести значення бонусних балів, які в нього є на рахунку, але при тому, не більше ніж 20% від вартості замовлення. Розрахунок здійснюється у реальному часі на основі вартості замовлення. Після оформлення замовлення користувач получає кешбек у вигляді 3,5% від вартості замовлення. Також користувач може побачити баланс своїх бонуснів балів в особистому кабінеті.

Після заповнення форми та підтвердження покупки, замовлення передається на сервер через HTTP POST-запит.

На бекенді система валідує отримані дані, у тому числі перевіряє цілісність замовлення, актуальність ціни та відповідність бонусних балів. Така багаторівнева перевірка дозволяє запобігти спробам шахрайства або втручання у дані на клієнтській стороні.

Також реалізовано інтеграцію з платіжною системою «Fondy», яка забезпечує можливість здійснення онлайн-оплати безпосередньо через мобільний застосунок [18]. Така інтеграція дозволяє користувачам оплачувати замовлення банківськими картками Visa та Mastercard, а також через Apple Pay або Google Pay – відповідно до підтримки пристрою. Інтеграція Fondy була здійснена з урахуванням рекомендацій з офіційної документації платіжної системи Fondy, що дозволило дотриматись сучасних стандартів безпеки

(зокрема, 3D Secure 2.0) та забезпечити стабільну роботу модуля в межах мобільного середовища. Інтерфейс зображений на рисунку 3.9.

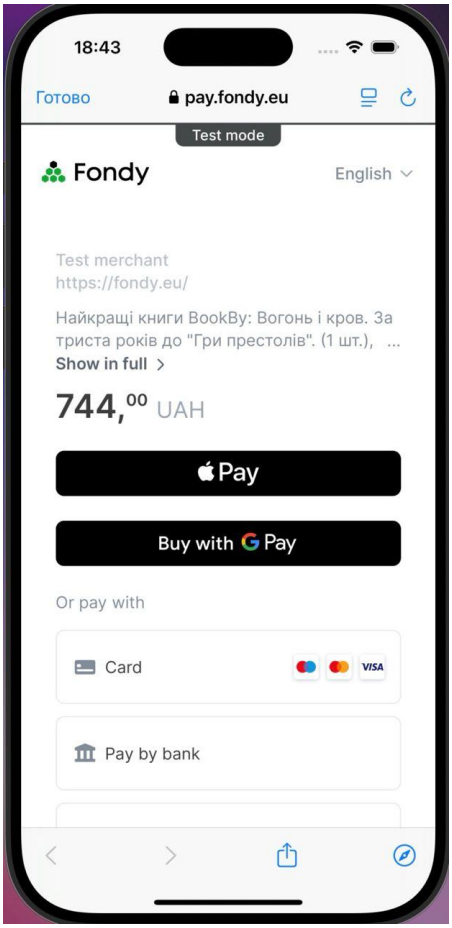


Рисунок 3.9 – Інтерфейс платіжної системи «Fondy»

Інтерфейс для модуля кошика та оформлення замовлень розроблений з акцентом на зручність, простоту і безпеку. Значну увагу приділено інтеграції з API «Нова Пошта» та платіжною системою «Fondy». Усе це в сукупності забезпечує комфортний процес покупки для користувачів та ефективну логістику для бізнесу.

3.4 Розробка особистого кабінета користувача з персональними даними.

Особистий кабінет – це елемент інтерфейсу сайту, який робить для користувача досвід більш персоналізованим [19]. Особистий кабінет є окремим

функціональним модулем мобільного застосунку, що надає користувачеві можливість переглядати та змінювати персональні дані, керувати улюбленими книгами, відстежувати залишені відгуки та виконувати вихід із системи. Його реалізовано за допомогою Angular та Capacitor з урахуванням адаптивного дизайну для мобільних пристроїв.

Після авторизації, користувач може попасти в особистий кабінет натиснувши «Профіль» в нижній частині навігації. Блок персональних даних містить ім’я, аватар користувача та поточну кількість накопичених бонусів та меню, для взаємодії з обліковим записом. Інтерфейс зображений на рисунку 3.10.

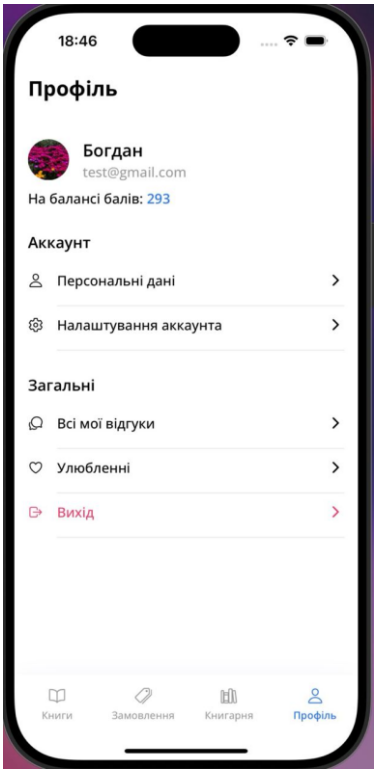


Рисунок 3.10 – Інтерфейс кабінету користувача

Кожне з цих полів інтерактивне, при натисканні «Персональні дані» відкривається форма редагування користувача. Можна змінити аватарку користувача, ім’я та пошту. Редагування реалізовано за допомогою форм Angular, що дає змогу виконувати вбудовану. Наприклад, у полі електронної пошти використано регулярні вирази для перевірки формату, а при збереженні даних система показує успішне або помилкове повідомлення, залежно від

відповіді API. Зміна аватарки реалізована через вибор файлів. Коли користувач обрав фотографію – файл конвертується у FormData та передається на сервер через HTTP-запит. Після успішного оновлення нове зображення миттєво відображається в інтерфейсі, без потреби перезавантаження сторінки. Форму редагування зображено на рисунку 3.11.

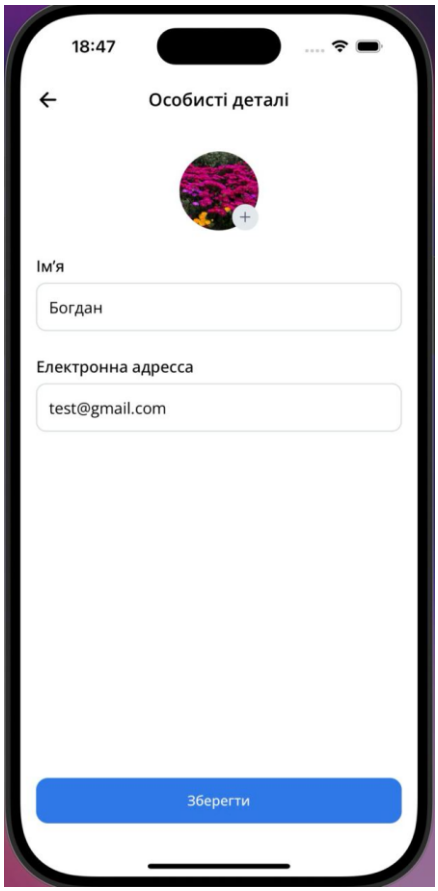


Рисунок 3.11 – Інтерфейс редагування облікового запису

Другим ключовим блоком є список улюблених книг. Він дозволяє користувачу зберігати обрані книги для зручного повернення до них у майбутньому. Усі книги відображаються у вигляді карток, кожна з яких містить обкладинку, назву, ім’я автора та кнопку перегляду детальної інформації. Щоб додати книгу до улюблених, йому потрібно зайти в деталі книги, і у нижньому кутку буде зображено іконка із серцем. При натисненні на нього, книга буде додана в улюблені. Інтерфейс сторінки «Улюблені» зображено на рисунку 3.12

Рисунок 3.12 – Інтерфейс сторінки «Улюблені»

Особистий кабінет також містить розділ відгуків користувача. Всі залишені коментарі згруповані в окрему секцію, де можна переглядати, редагувати або видаляти кожен з них. Редагування відбувається через модальне вікно з можливістю зміни тексту. Усі зміни автоматично відображаються на екрані без перезавантаження завдяки використанню асинхронного підходу до оновлення стану інтерфейсу. Це дозволяє користувачу бачити миттєвий результат своїх дій, що позитивно впливає на сприйняття застосунку. Інтерфейс відгуків зображено на рисунку 3.13

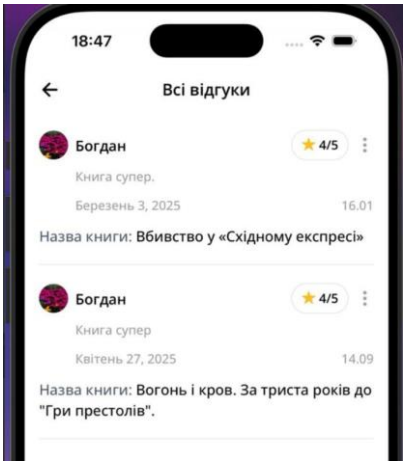


Рисунок 3.13 – Інтерфейс сторінки «Відгуки»

Особистий кабінет не обмежується відображенням персональної інформації. Це багатофункціональний простір взаємодії користувача з платформою, що об’єднує управління даними, улюбленими позиціями, контентом (відгуками), бонусною системою та безпекою акаунта. Ретельно продуманий інтерфейс дозволяє забезпечити високу продуктивність, інтуїтивність та персоналізацію, що відповідає очікуванням сучасного користувача.

3.5 Розробка модуля для читання книг із підтримкою базових функцій перегляду.

Однією з ключових функцій мобільного застосунку «Книжковий магазин та бібліотека» є можливість читання електронних книг прямо в застосунку. З огляду на сучасні тенденції цифрового споживання контенту, читання має бути зручним, швидким і доступним без сторонніх програм. Тому було реалізовано окремий модуль для перегляду книг у форматі PDF, з підтримкою збереження індивідуального прогресу читання для кожного користувача.

У кожного користувача є своя персональна бібліотека, де кожен може завантажити свою електронну книгу. Коли користувач додає книгу, він має вказати: назву, обкладинку для книги і саму книгу. Файл книги у форматі PDF зберігається на сервері, а клієнт отримує посилання для перегляду. Таким чином, у мобільному застосунку користувач бачить в каталозі лише метадані книги, що дозволяє не перевантажувати мережу при звичайному перегляді. Інтерфейс додавання книги зображено на рисунку 3.14. Для читання реалізовано вбудований переглядач PDF-документів. Його реалізовано за допомогою бібліотеки pdf.js, оптимізований під мобільні пристрої.

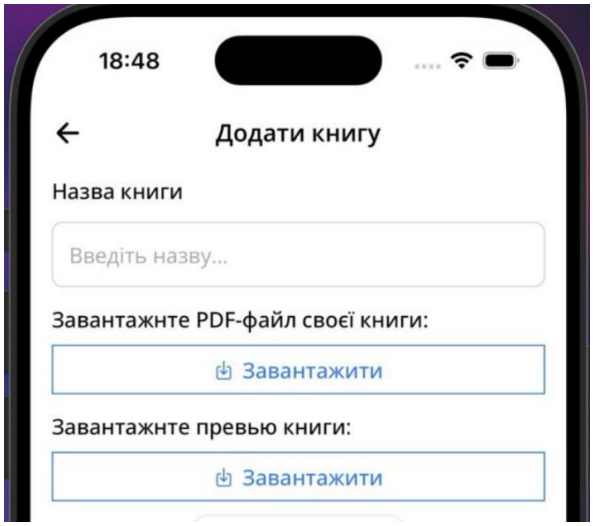


Рисунок 3.13 – Інтерфейс додавання книги

Книга відкривається на окремому екрані, де зосереджено увагу виключно на контенті. Елементи інтерфейсу мінімальні: навігація по сторінках, індикатор поточної сторінки та загальна кількість сторінок, кнопка виходу. Такий підхід забезпечує неперервність процесу читання, а відсутність зайвих елементів підвищує зосередженість користувача на тексті (див. рисунок. 3.14).

Рисунок 3.14– Інтерфейс відображення перегляду книги

Для кращого досвіду система зберігає прогрес читання кожної книги. Це реалізовано через збереження номера останньої прочитаної сторінки у базі даних. Після виходу із книги, система відправляє останню сторінку на якій зупинився користувач, і зберігає в базі даних цю сторінку.

При наступному відкритті книги система автоматично відкриває переглядач на останній збереженій сторінці. Такий підхід дозволяє продовжити читання з того місця, де користувач зупинився, навіть після виходу з застосунку або зміни пристрою. Прогрес зображений на рисунку 3.15.

Рисунок 3.15– Інтерфейс прогресу книги

Модуль читання книг став важливим доповненням функціоналу застосунку, що забезпечує повний цикл роботи з електронним контентом – від перегляду до завершеного читання. Такий підхід дозволяє створити зручне й комфортне середовище для сучасного читача, який очікує безперешкодного доступу до контенту будь-де і будь-коли.

3.6 Висновки

У третьому розділі обґрунтовано вибір технологій та інструментів, що використовувалися для розробки програмних модулів мобільної системи книжкового магазину. Виконавши аналіз вимог до програмного продукту, обрано стек технологій Angular, NestJS та Capacitor для забезпечення кросплатформенності та ефективності роботи застосунку.

Розглянуто розробку ключових модулів:

– модуль авторизації та реєстрації, який реалізує систему входу користувачів із валідацією даних на фронтенді;

– модуль управління кошиком, що використовує LocalStorage для збереження товарів;

– модуль оформлення замовлення, інтегрований з API "Нова Пошта" для вибору міста та відділення доставки;

– модуль для читання книг, з імплементацією бібліотеки pdf.js.

Розроблені модулі забезпечують безперебійну роботу системи, спрощуючи користувачам процес авторизації, управління кошиком і оформлення замовлень. Впроваджені технології та методи дозволяють гарантувати стабільність, масштабованість і зручність використання мобільного застосунку.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Тестування мобільного застосунку

Тестування програмного забезпечення є невід'ємною частиною розробки, оскільки дозволяє виявити помилки, дефекти та невідповідності ще до передачі продукту у використання. Його головною метою є гарантування надійності, стабільності та відповідності очікуванням замовника. Щоб почати тестування, попередньо формується тестове середовище, куди входить інсталяція тестової збірки додатку на робочі станції з Android або IOS, залежно від цільових платформ. Водночас готується відповідна документація – специфікації вимог і тестовий план, дозволяють оцінити відповідність функціоналу встановленим критеріям [20].

Процес розпочинається з перевірки функціональної коректності. На цьому етапі особливу увагу приділяють сценаріям введення та виведення даних, обробці невалідного вводу, а також дотриманню бізнес-логіки, закладеної в додатку. Усі основні механізми, починаючи з навігації інтерфейсом, завершуючи обробкою замовлення, проходять перевірку на відповідність заданим умовам.

Далі переходять до тестування на відмовостійкість. Тут моделюються різні виняткові ситуації: порушення з'єднання, збої при обробці запитів, передача некоректних даних або дія в умовах ізольованого середовища. Це дозволяє перевірити здатність системи не просто реагувати на помилки, а й правильно відновлювати роботу без втрати даних або порушення логіки.

Після перевірки функціональності та стабільності розпочинається продуктивне тестування. У його межах фіксуються час реакції програми на запити, швидкість виконання основних операцій і здатність застосунку працювати під навантаженням. Завдяки таким перевіркам можна об'єктивно оцінити загальну ефективність програмного забезпечення [21].

Завершальним етапом стає тестування сумісності. Воно охоплює перевірку роботи застосунку на різних пристроях, версіях IOS та програмного забезпечення, що встановлене на них. Такі перевірки дозволяють виявити

конфлікти між компонентами, адаптувати інтерфейс до різних екранів та уникнути потенційних обмежень у продуктивності або функціональності.

Під час початкового етапу тестування програмного забезпечення було перевірено реакцію застосунку на введення некоректних даних авторизації. Якщо користувач вводить неправильну комбінацію логіна та пароля або залишає одне з полів порожнім, система повинна виявити помилку та повідомити про неправильні облікові дані. У цьому випадку застосунок виводить повідомлення з поясненням, що дані введено некоректно, та пропонує повторити спробу. На рисунку 4.1 зображено інтерфейс авторизації з відповідним повідомленням про помилку.

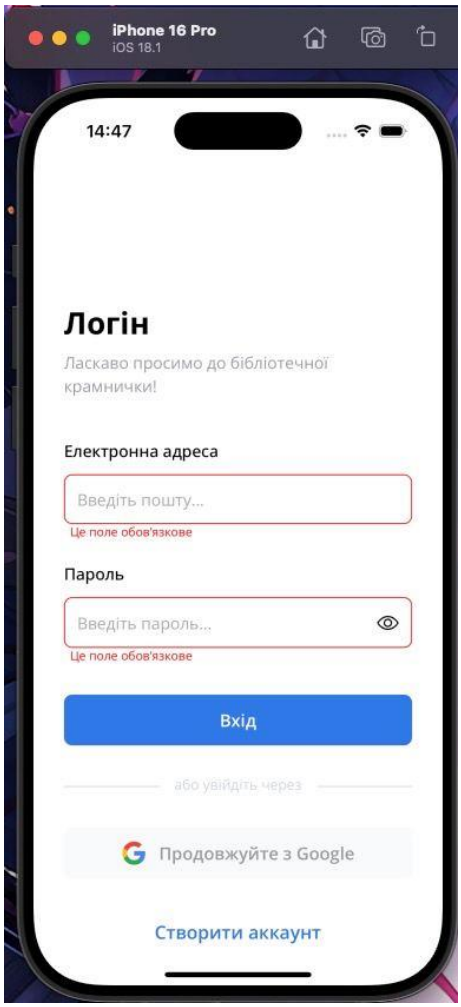


Рисунок 4.1 – Повідомлення про помилку під час авторизації

Після тестування сценарію з неправильними обліковими даними було проведено додаткове тестування, яке перевіряло реакцію застосунку на спробу

входу з обліковим записом, що не існує в системі. У цьому випадку користувач вводив правильний формат логіна та пароля, проте дані не відповідали жодному з наявних акаунтів. Система повинна розпізнати спробу доступу з неіснуючим акаунтом і повідомити користувача, що такий обліковий запис не зареєстровано. На рисунку 4.2, застосунок надає відповідне повідомлення.

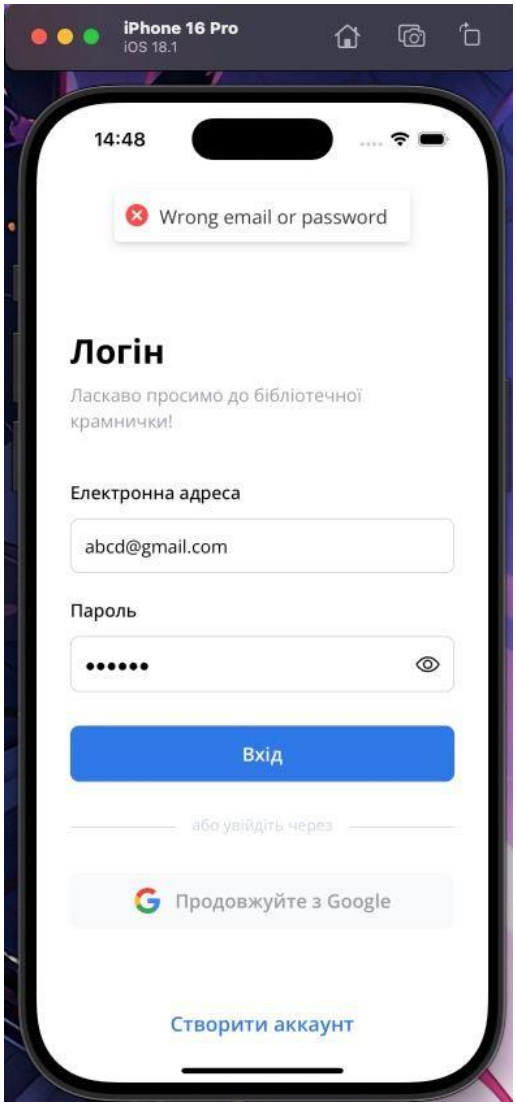


Рисунок 4.2 – Повідомлення про спробу входу з неіснуючим обліковим записом

Після успішного тестування авторизації програми, наступним кроком стало тестування модуля пошуку товарів. Користувач вводить ключове слово у відповідне поле на головному екрані застосунку. У відповідь система проводить пошук серед наявних позицій у базі даних та відображає релевантні результати

у вигляді карток товарів. На рисунку 4.3 представлений приклад коректного пошуку за запитом «агата», у результаті якого були знайдені відповідні книги, що містять вказане ключове слово в назві або ім’я автора.

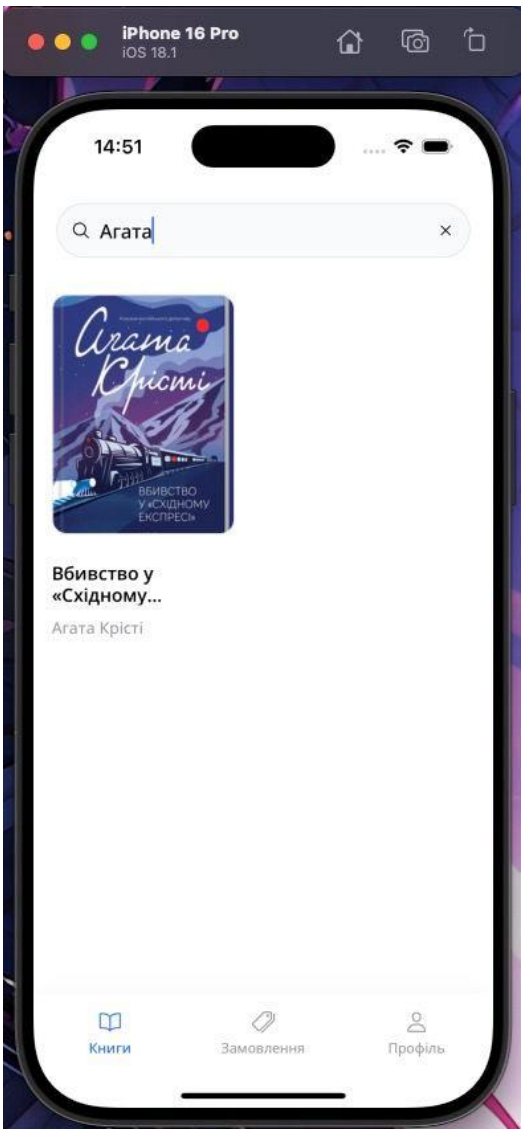


Рисунок 4.3 – Результати пошуку за ключовим словом

У межах цього ж тестування було перевірено поведінку системи у разі введення користувачем некоректного запиту або запиту, що не має збігів у базі. У цьому випадку застосунок має вивести зрозуміле повідомлення про відсутність результатів. На рисунку 4.4 продемонстровано, як система коректно інформує користувача про те, що результати не знайдено.

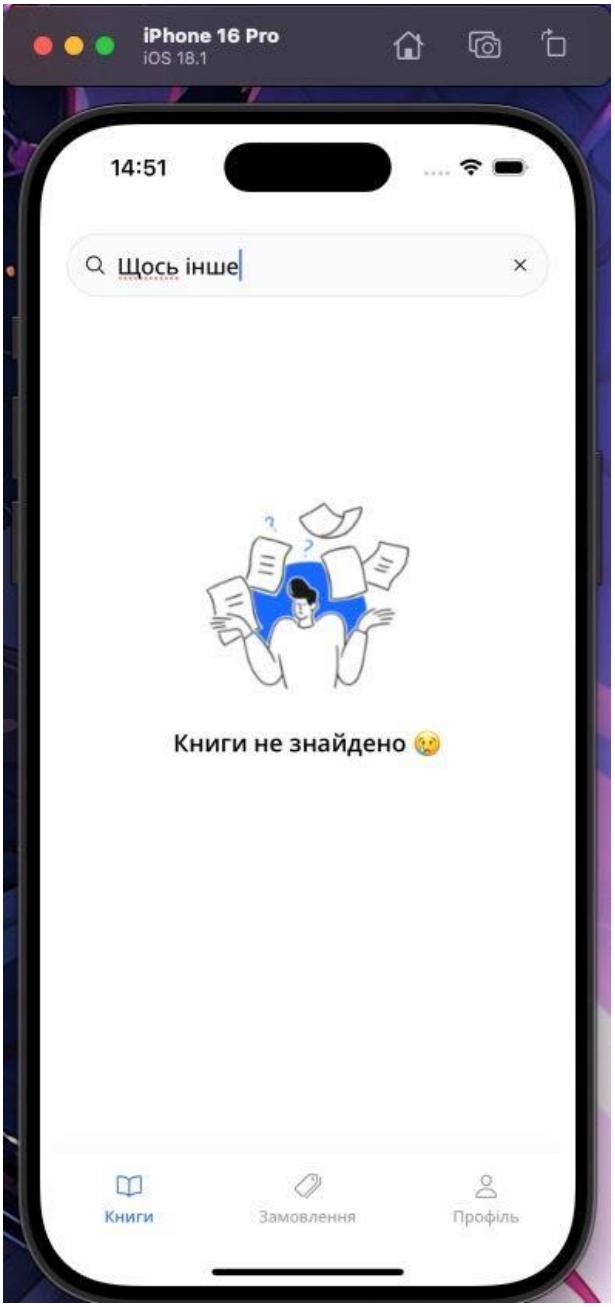


Рисунок 4.4 – Повідомлення про відсутність товарів за запитом

Після перевірки функціоналу пошуку товарів, наступним етапом стало тестування модуля додавання товарів до корзини. Користувач переходить на сторінку обраного товару та натискає кнопку "Додати до корзини". У відповідь система миттєво оновлює стан інтерфейсу, змінюючи іконку корзини у верхній панелі навігації, де з’являється індикатор із кількістю доданих товарів. На рисунку 4.5 продемонстровано, як виглядає інтерфейс після успішного додавання книги до корзини.

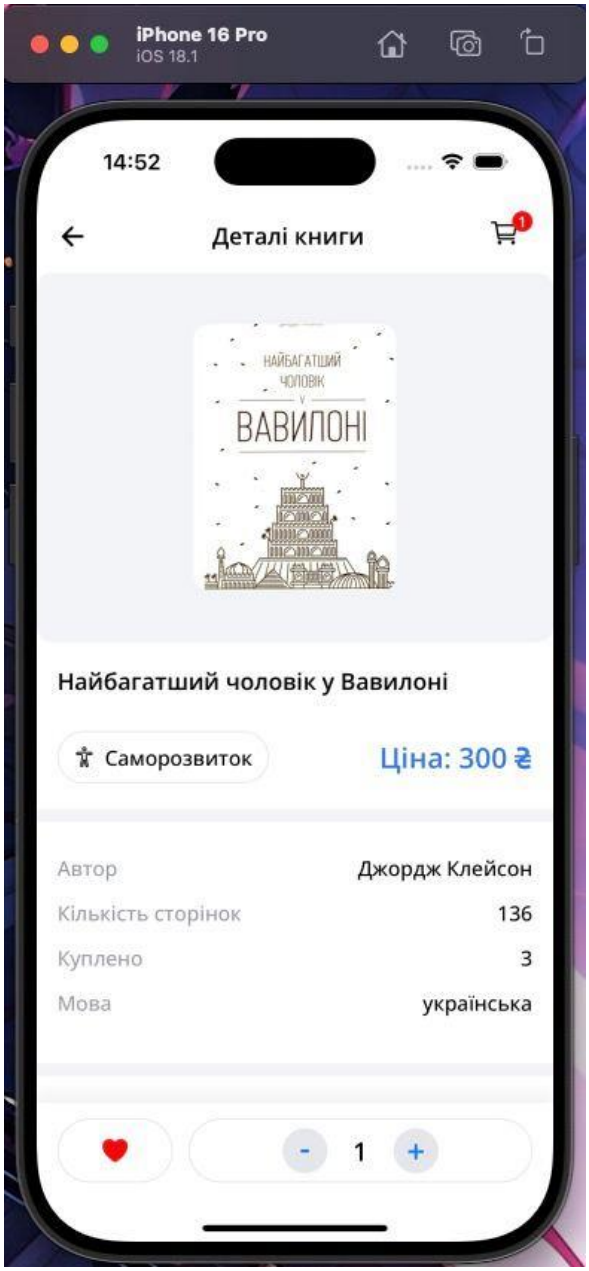


Рисунок 4.5 – Результат додавання товару в корзину

Ще одним важливим сценарієм тестування стала перевірка роботи функції «додати до корзини» при повторному додаванні того самого товару. Користувач обирає товар і натискає кнопку «Додати до корзини» двічі або більше разів. Система повинна не створювати дублікати товару, а збільшувати його кількість у поточній корзині. Результати тестування показали, що програмне забезпечення коректно обробляє таку взаємодію – у списку замовлень кількість товару змінюється, а загальна сума автоматично перераховується. На рисунку 4.6 показано відображення товару з оновленою кількістю в корзині.

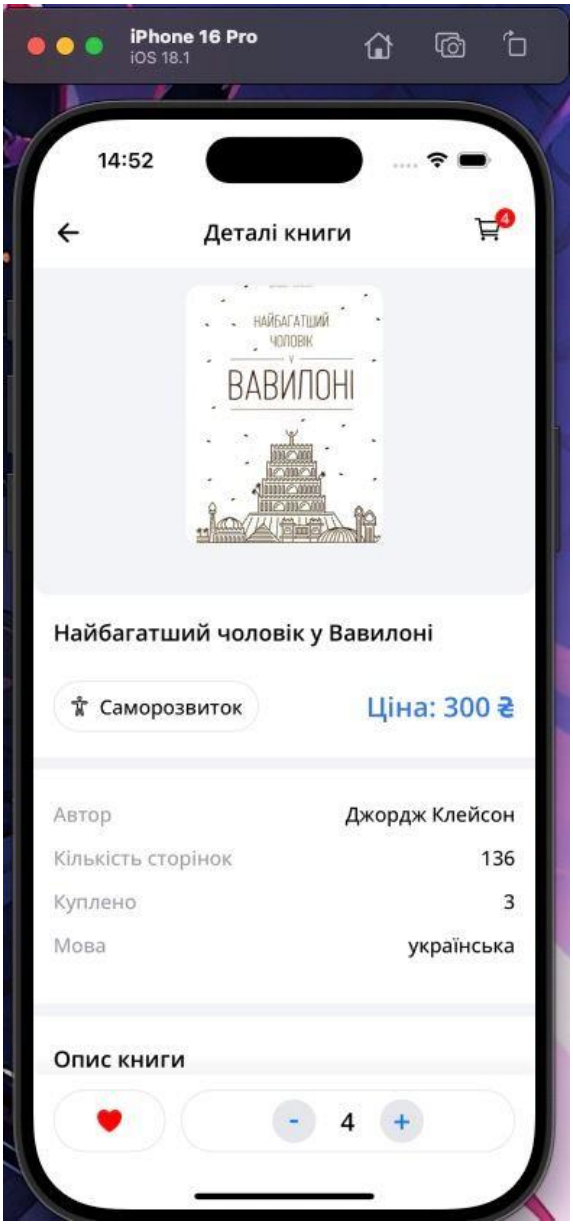


Рисунок 4.6 – Автоматичне оновлення кількості товару при повторному додаванні

Тестування функціональності особистого кабінету користувача включало перевірку правильності відображення та редагування персональних даних. Користувач входить в особистий кабінет і намагається змінити свої дані, такі як ім’я, аватарку та електронну пошту. Після збереження змін система повинна відобразити оновлені дані та підтвердити успішне оновлення. Результати тестування показали, що програмне забезпечення коректно обробляє зміни, і в особистому кабінеті миттєво відображаються актуальні дані користувача. На рисунку 4.7 показано зміну користувача при редагуванні інформації.

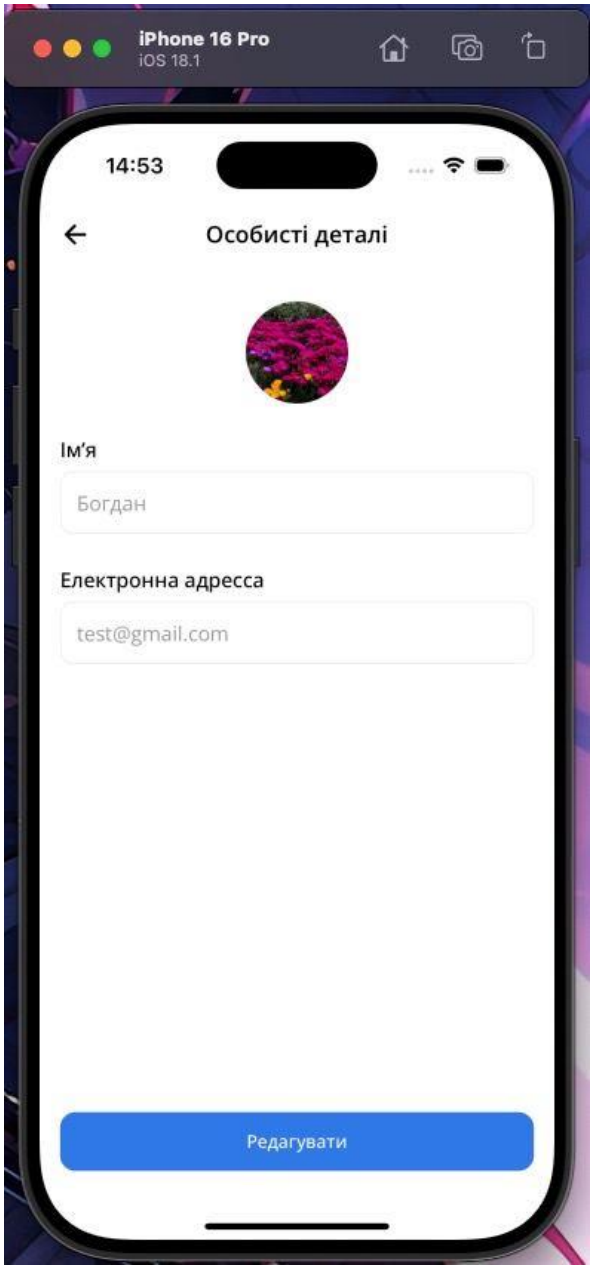


Рисунок 4.7 – Оновлені персональні дані користувача в особистому кабінеті

Тестування функціональності додавання відгуків передбачало перевірку процесу публікації відгуку користувачем про товар. Під час тестування користувач спробував додати відгук на товар, зокрема, перевірялося, чи система дозволяє публікувати лише один відгук від кожного користувача. Після того, як користувач залишив відгук, система повинна відобразити його на сторінці товару та підтвердити успішне додавання. На рисунку 4.8 зображено помилку, якщо користувач вже має відгук для цієї книги.

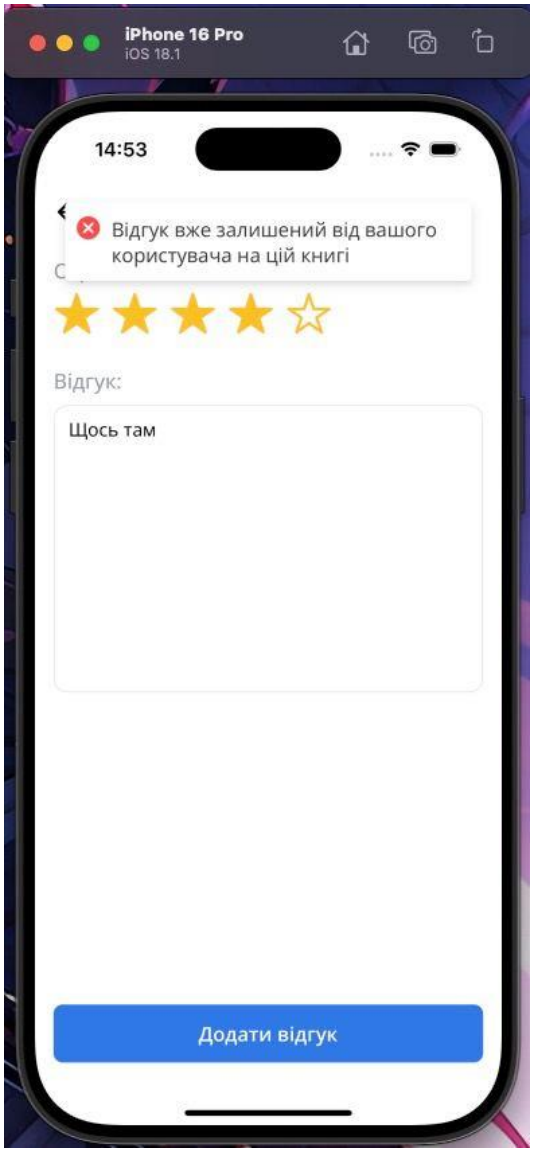


Рисунок 4.8 – Помилка, якщо відгук вже існує для цієї книги.

Наступним етапом стало тестування модуля перегляду замовлень, що реалізований у вигляді окремої вкладки в інтерфейсі програми. Цей модуль дозволяє користувачеві переглядати список усіх своїх замовлень у форматі карток, кожна з яких містить ключову інформацію: номер замовлення, дату оформлення, загальну суму та поточний статус.

У межах тестування було перевірено коректність відображення даних для замовлень різного типу та статусу. Система успішно відображає як нові замовлення. При цьому всі дані відповідають реальній інформації в базі даних. Після натискання на будь-яку картку замовлення відкривається деталізована сторінка, на якій користувач може переглянути:

- повний перелік товарів у замовленні (назва, кількість, ціна за одиницю, підсумкова вартість),
- обраний спосіб доставки,
- платіжну інформацію,
- актуальний статус із позначкою часу останнього оновлення.

На рисунку 4.9 зображено сторінку з деталями конкретного замовлення.

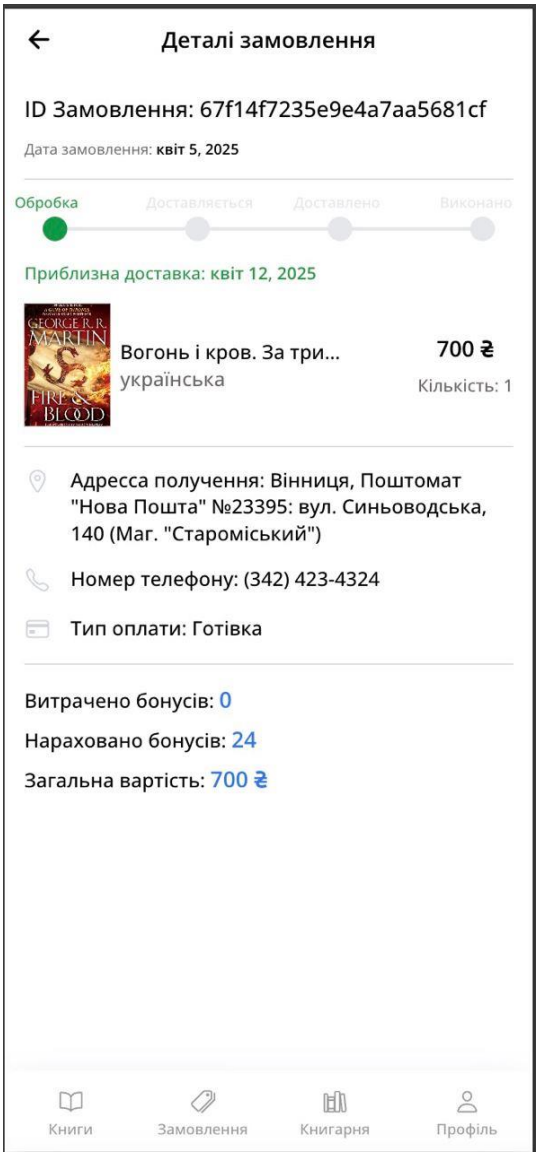


Рисунок 4.9 – Інтерфейс сторінки «Деталі замовлення».

У процесі тестування було перевірено зміну статусу, якщо адміністратор або менеджер вирішить змінити статус замовлення, це буде одразу змінено у користувача. На рисунку 4.10 зображено зміну статусу у замовлення.

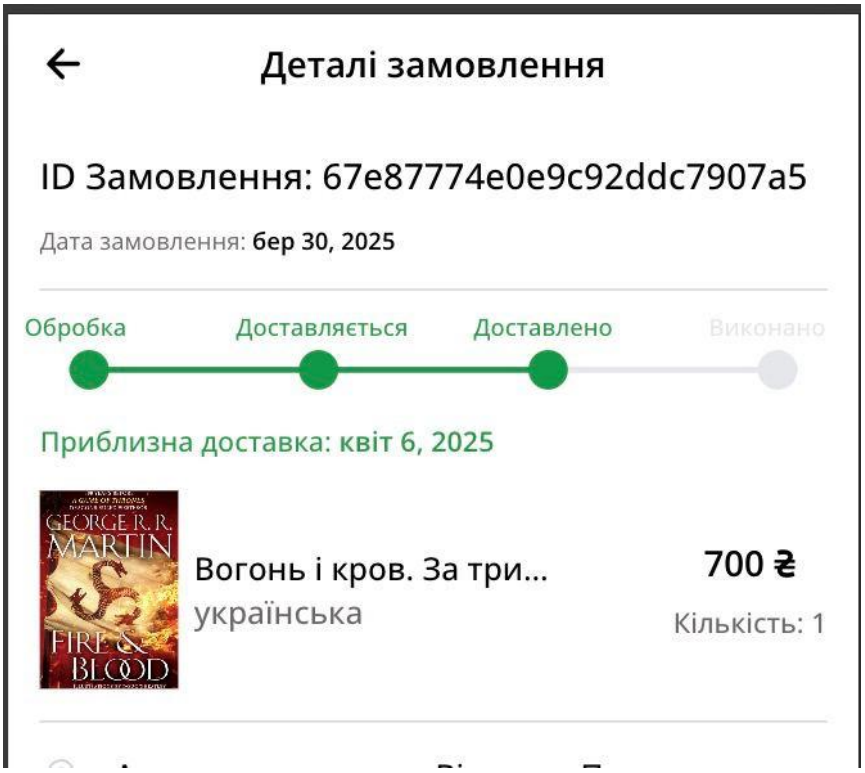


Рисунок 4.10 – Зміна статусу замовлення.

Також все тестування серверної сторони відбувалось через Swagger. Swagger – це технологія, яка дозволяє документувати REST-сервіси. Swagger підтримує безліч мов програмування і фреймворків. Також Swagger надає UI для перегляду документації, зрозумілому для користувача і комп'ютера [22]. Інтерфейс Swagger зображено на рисунку 4.11

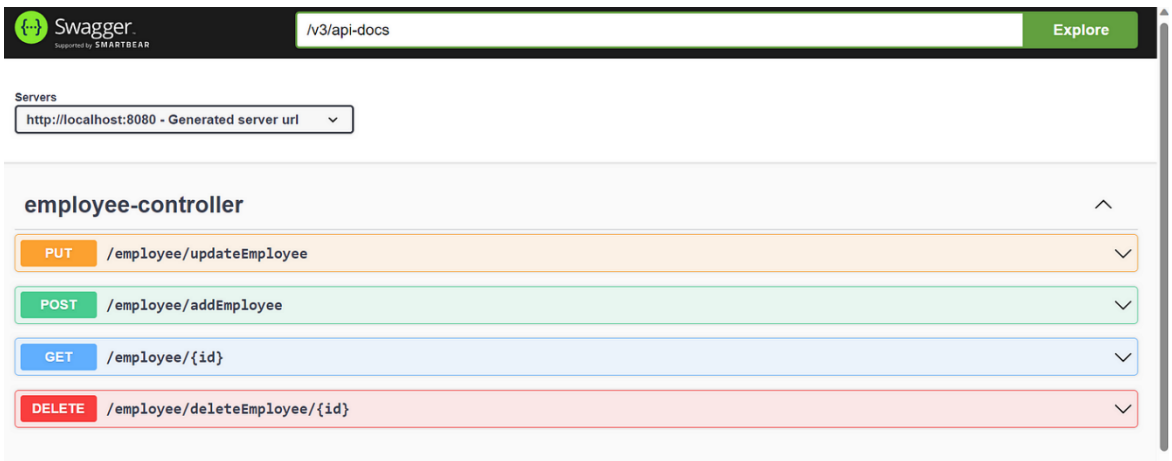


Рисунок 4.11 – Інтерфейс Swagger.

Додатково до тестування швидкості відгуку системи на дії користувача було проведено аналіз часу, необхідного для додавання товару в кошик. Під час тестування виявлено, що система здатна додавати товар до кошика з швидкістю, що не перевищує 0.3 секунди. Це означає, що користувачі можуть миттєво отримувати зворотний зв'язок при додаванні товару через інтерфейс програми.

Одночасно, час, необхідний для оновлення вмісту кошика та відображення кількості товарів у ньому, складає не більше ніж 1 секунду. Це свідчить про ефективність та високу швидкість обробки даних системою, що забезпечує зручний і безперебійний процес додавання товарів до кошика без затримок.

4.2 Розробка інструкції користувача

Інструкція користувача включає визначення технічних вимог для запуску мобільного застосунку [23]. Перед початком роботи із застосунком користувачеві необхідно переконатися, що пристрій відповідає мінімальним технічним вимогам. Деталі щодо мінімальної та рекомендованої конфігурації наведені в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурації	Рекомендовано	Мінімально
Процесор	Qualcomm Snapdragon 8xx або Apple A13 Bionic та вище	Qualcomm Snapdragon 6xx або Apple A10 Fusion та вище
Оперативна пам'ять	6 ГБ RAM	3 ГБ RAM
Вільний простір на диску	1ГБ	100МБ
Операційна система	Android 10 або iOS 14	Android 8.0 або iOS 12

Продовження таблиці 4.1

Вимоги до конфігурації	Рекомендовано	Мінімально
Дисплей	Роздільна здатність не менше 1080x2340 пікселів (Full HD+)	Роздільна здатність не менше 720x1520 пікселів (HD+)

Мобільний застосунок встановлюється через .apk-файл. Щоб здійснити встановлення, користувачу необхідно:

1. Завантажити .apk-файл з офіційного сайту, корпоративної пошти або хмарного сховища (наприклад, Google Drive, Dropbox).
2. Перейти до налаштувань смартфона, перейти до вкладки «Безпека», потім активувати опцію «Дозволити встановлення з невідомих джерел».
3. Відкрити завантажений .apk-файл і підтвердити встановлення.
4. Після завершення процесу на головному екрані пристрою з’явиться іконка застосунку.

Після інсталяції та запуску мобільного застосунку користувач переходить на вітальний екран і чекає на завантаження головного екрану «Onboarding» етапу застосунку. Після цього він може розпочати взаємодію з програмою.

У головному меню присутнє поле пошуку та доступ до фільтрів за жанрами. Після введення запиту система здійснює пошук у базі даних і відображає відповідні результати. Користувач може переглянути книгу, де вказано опис, ціну, наявність, рейтинг, відгуки.

На сторінці товару натискається кнопка «Додати до кошика». Після цього з’являється індикатор з кількістю товарів. Перейшовши до кошика, користувач має змогу змінювати кількість примірників або видаляти позиції.

Після цього, щоб завершити покупку, необхідно натискати кнопку «Оформити замовлення». У цьому процесі користувач вводить необхідні платіжні реквізити та дані для доставки. Також є можливість переглядати історію своїх замовлень та статус кожного з них у спеціальному розділі.

У вкладці «Профіль» користувач може змінити особисті дані, завантажити фото, оновити електронну адресу, змінити пароль, а також переглядати історію покупок.

Після покупки користувач має змогу залишити відгук про книгу. В системі передбачено обмеження: один відгук на одну книгу від одного користувача.

Застосунок дозволяє користувачеві налаштувати свій профіль, а також переглядати та залишати відгуки на книги після їх придбання. Окрім цього, є розділи «Довідка» та «Про програму», що містять основну інформацію про функціонування застосунку.

Інструкція користувача дозволяє забезпечити швидке ознайомлення з можливостями мобільного застосунку та дає покрокові рекомендації з його інсталяції, використання основних функцій і налаштування профілю.

4.3 Висновки

У межах четвертого розділу детально розглянуто прикладне тестування розробленого мобільного застосунку для онлайн-магазину книг, а також створено інструкцію користувача, що дозволяє ефективно ознайомитися з основними можливостями програмного продукту.

У ході тестування були перевірені ключові функціональні модулі, зокрема:

- авторизація користувача, включаючи обробку помилкових сценаріїв введення логіна та пароля;
- пошук товарів за ключовими словами та робота з фільтрами;
- додавання товарів до кошика, автоматичне оновлення кількості при повторному додаванні та виведення актуальної суми замовлення;
- редагування особистих даних у профілі користувача;
- публікація відгуків з перевіркою обмеження на дублювання;
- взаємодія з модулем замовлень, зокрема перегляд списку всіх замовлень, їх статусів і детальної інформації.

Тестування виконувалося в умовах змодельованого середовища на платформах Android та iOS. Таким чином, можна зробити висновок, що мобільний застосунок відповідає функціональним, технічним та ергономічним вимогам, що ставились до нього на початку проєкту. Реалізований функціонал забезпечує повний цикл користування онлайн-магазином: від ознайомлення з товарами до формування та контролю замовлень. Це свідчить про готовність продукту до подальшого масштабування, впровадження та використання в реальному середовищі.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи розроблено програмні засоби мобільного застосунку «Книжковий магазин та бібліотека», що дозволяє здійснювати повноцінну взаємодію користувача з електронною платформою для пошуку, придбання та оцінювання книжкової продукції.

Мобільний застосунок створений за допомогою фреймворку Angular разом із Capacitor, що забезпечує кросплатформенність і дає змогу запускати застосунок на Android та iOS-пристроях. Серверну частину було створено за допомогою NestJS, а збереження даних організовано за допомогою бази даних MongoDB. Бакалаврська кваліфікаційна робота розроблено згідно методичних вказівок [24].

У бакалаврській кваліфікаційній роботі виконано такі завдання:

- розроблено авторизацію та реєстрацію користувачів;
- розроблено каталог книг із можливістю пошуку, фільтрації та сортування;
- розроблено модуль управління кошиком;
- реалізовано особистий кабінет користувача;
- впроваджено модуль для інтегрованого читання книг;
- проведено тестування всіх основних функціональних;
- розроблено інструкцію користувача.

Під час розробки враховано вимоги до використання на різних системах (Anroid та IOS), UX дизайну та доступності функціоналу. У процесі розробки застос було застосовано принципи ООП, також розроблено структуру бази даних, реалізовано взаємодію з API, а також створено механізми для збереження даних, обробки подій і навігації в межах застосунку.

Описано аналіз існуючих аналогів до сучасних систем електронної комерції показав доцільність розробки саме спеціалізованого мобільного застосунку для роботи з книжковим асортиментом.

Також описано тестування, що підтвердило відповідність застосунку технічному завданню. Результати показали стабільну роботу в типових та виняткових ситуаціях, коректне відображення інтерфейсу на різних пристроях, а також відсутність критичних помилок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна комерція. [Електронний ресурс] URL: https://me.gov.ua/Documents/Detail?lang=uk-UA&id=6d3d0ba1-219b-470a-a9b9-a88f7dbf02ec&title=ElektronnaKomertsiia&isSpecial=true

2. «Мобільна комерція в Україні: стан та перспективи розвитку» [Електронний ресурс] URL: https://uaateam.agency/blog/trendy-ta-vyklyky-ukrainskogo-rynku-ecommerce/

3. «Тренди дизайну мобільних додатків» [Електронний ресурс] URL: https://lemon.school/blog/trendy-dyzajnu-mobilnyh-dodatkiv-2024

4. В. П. Майданюк, Яворський Б.М. Роль мобільних застосунків у популяризації читання: як цифрові технології можуть збільшити інтерес до книг. LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) [Електронний ресурс] URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23661/19549

5. Statista. Mobile e-commerce sales worldwide from 2016 to 2024 [Електронний ресурс] URL: https://www.statista.com/statistics/249863/

6. Просування мобільних додатків – Як розкрутити мобільний застосунок Електронний ресурс] URL: https://netpeak.net/uk/blog/prosuvannya-mobil-nikh-zastosunkiv-yak-rozkrutiti-mobil-niy-zastosunok/

7. Yakaboo [Електронний ресурс] URL: https://www.yakaboo.ua/

8. Ridme [Електронний ресурс] URL: https://ridmi.com.ua/?srsltid=AfmBOoqFsvWIu3qvNtYy4l1yTRjLp60Q5YScq4TW9xyLjx-oiQJ6s6NY

9. PocketBook Reader [Електронний ресурс] URL: https://pocketbook.com.ua/uk-ua/app-ua

10. MongoDB. NoSQL Database for Modern Applications [Електронний ресурс] URL: https://www.mongodb.com

11. Kleppmann M. Designing Data-Intensive Applications. – O'Reilly Media, 2017. – 616 с.

12. Capacitor [Електронний ресурс] URL: https://capacitorjs.com/

13. REST API як спосіб спілкування компонент веб-додатків. [Електронний ресурс] URL: https://foxminded.ua/shcho-take-rest-api/

14. Інтерфейс Користувача (UI) [Електронний ресурс] URL: https://wezom.com.ua/ua/blog/chto-takoe-ui-i-kak-polzovatelskij-interfejs-vliyaet-na-prodazhi-internet-magazina

15. Draw.io [Електронний ресурс] URL: https://app.diagrams.net/

16. Angular Framework [Електронний ресурс] URL: https://angular.dev/

17. Google. Material Design Guidelines. [Електронний ресурс] URL: https://m3.material.io/

18. Fondy – платіжна система. [Електронний ресурс URL: https://docs.fondy.eu/en/

19. Створення сайту з особистим кабінетом [Електронний ресурс] URL: https://surl.li/hwilxq.

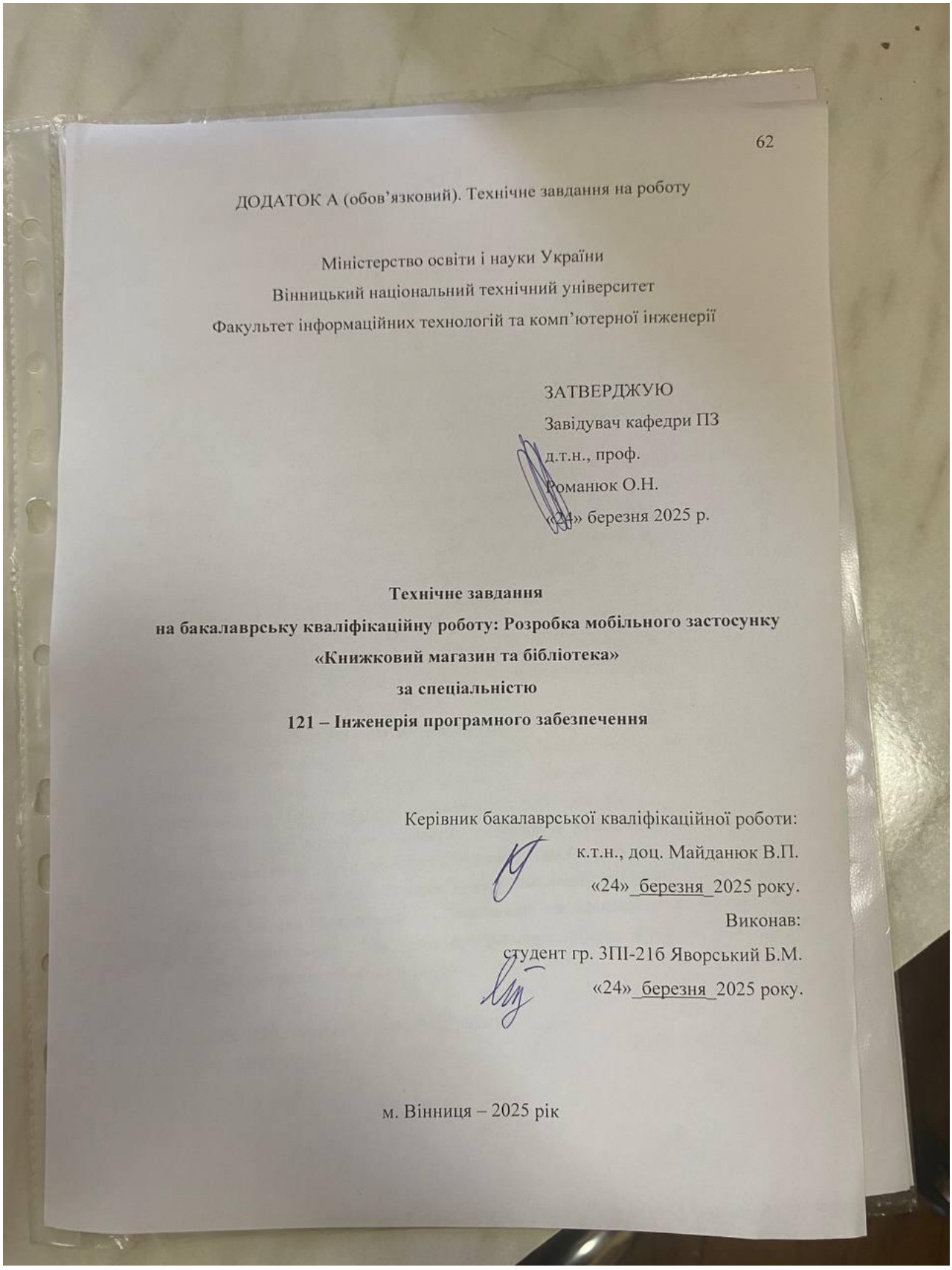
20. Специфікація вимог і як їх тестувати [Електронний ресурс] URL: https://training.qatestlab.com/blog/technical-articles/what-the-requirements-specification-looks-like-and-how-to-test-it/

21. Тестування продуктивності [Електронний ресурс] URL: https://training.qatestlab.com/blog/technical-articles/performance-testing/

22. Що таке Swagger. [Електронний ресурс] URL: https://qagroup.com.ua/publications/what-is-swagger/

23. Тестування мобільних додатків [Електронний ресурс] URL: https://surl.li/kgcyow.

24. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52 с



62

ДОДАТОК А (обов'язковий). Технічне завдання на роботу

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф.
Романюк О.Н.
«24» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу: Розробка мобільного застосунку «Книжковий магазин та бібліотека»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Майданюк В.П.
«24» березня 2025 року.
Виконав:
студент гр. ЗПІ-21б Яворський Б.М.
«24» березня 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка мобільного застосунку Книжковий магазин та бібліотека».

Галузь застосування – мобільні-технології, електрона комерція.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є поєднання можливостей електронної комерції та цифрового читання в межах одного мобільного застосунку, що автоматизують та спрощують процеси вибору, оформлення та читання книг, які дозволять зменшити час на пошук необхідних видань та підвищити точність рекомендацій та забезпечити зручний доступ до електронного контенту.

Основними задачами роботи є:

- провести аналіз аналогів і сформувати вимоги до мобільного застосунку;
- розробити швидкий алгоритм пошуку книг за назвою, автором, жанром та іншими критеріями;
- розробити функціонал авторизації та реєстрації користувачів з валідацією даних;
- розробити модуль формування кошика, який буде спроможний зберігати вибрані книги, відображати їх кількість та вартість;
- розробити модуль оформлення замовлення, який буде забезпечувати зручне введення даних для доставки та оплати;
- розробити модуль для можливості зберігання та читання електронних книг;
- провести тестування розробленого мобільного застосунку.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. MongoDB. NoSQL Database for Modern Applications [Електронний ресурс]. – Режим доступу: https://www.mongodb.com

2. Capacitor [Електронний ресурс] – Режим доступу до ресурсу: https://capacitorjs.com/

3. REST API як спосіб спілкування компонент веб-додатків. [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/shcho-take-rest-api/

4. Angular Framework [Електронний ресурс] – Режим доступу до ресурсу: https://angular.dev/

5. Google. Material Design Guidelines. [Електронний ресурс] – Режим доступу до ресурсу: https://m3.material.io/

6. Fondy – платіжна система. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.fondy.eu/en/

5. Технічні вимоги

– середовище розробки – WebStorm, XCode;

– мова програмування – JavaScript;

– метод захисту – шифрування;

– ступінь захисту – високий;

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР;

2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій мобільних систем для книжкових магазинів та постановка задач дослідження	25.03.25- 08.04.25
2	Розробка моделей та алгоритмів програмного продукту	09.04.25 - 20.04.25
3	Аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення	21.04.25 - 23.04.25
4	Розробка структури та програмних модулів мобільної системи книжкового магазину	24.04.25 - 20.05.25
5	Тестування застосунку	21.05.25 - 25.05.25
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

ДОДАТОК Б (обов'язковий). Протокол перевірки на наявність запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного застосунку «Книжковий магазин та бібліотека».

Тип роботи: БКР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.

Оригінальність	96%
Схожість	4%

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Strikeplagiarism

Автор роботи _____ Яворський Б.М..

Керівник роботи _____ Майданюк В.П.

ДОДАТОК В (довідниковий). Лістинг коду додатку

Front-End:

core/api/auth/auth-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class AuthApiService {
  public http = inject(HttpClient);

  public signUpUser(signUpDto: SignUpDto): Observable<{ message: string }> {
    return this.http.post<{ message: string }>(AuthEnum.SignUp, signUpDto);
  }

  public loginUser(loginDto: LoginDto): Observable<ILoginResponse> {
    return this.http.post<ILoginResponse>(AuthEnum.Login, loginDto);
  }

  public refreshToken(token: string | null): Observable<IRefreshResponse> {
    return this.http.post<IRefreshResponse>(AuthEnum.Refresh, { token });
  }
}
```

core/api/books/ books-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class BookApiService {
  public http = inject(HttpClient);

  public getMostBooks(): Observable<IBook[]> {
    return this.http.get<IBook[]>(BookEnum.BooksMost);
  }

  public getAllGenres(): Observable<IGenres[]> {
    return this.http.get<IGenres[]>(BookEnum.GenresAll);
  }

  public getSearchBook(value: string): Observable<IBook[]> {
    return this.http.get<IBook[]>(BookEnum.BooksSearch, { params: { value }
});
  }

  public getDetailBook(id: string): Observable<IBook> {
    return this.http.get<IBook>(BookEnum.BookDetail, { params: { id } });
  }

  public getBookByGenre(value: string): Observable<IBook[]> {
    return this.http.get<IBook[]>(BookEnum.BooksGenre, { params: { value }
});
  }
}
```

core/api/favorite/ favorite-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class FavoriteApiService {
  public http = inject(HttpClient);

  public getFavorite(): Observable<IBook[]> {
    return this.http.get<IBook[]>(FavoriteEnum.FavoriteGet);
  }
```

```
  public addFavorite(bookId: IBook['_id']): Observable<void> {
    return this.http.post<void>(FavoriteEnum.FavoriteAdd, { bookId });
  }

  public removeFavorite(bookId: IBook['_id']): Observable<void> {
    return this.http.delete<void>(`${FavoriteEnum.FavoriteRemove}${bookId}`);
  }
}
```

core/api/payments/payments-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class PaymentsApiService {
  public http = inject(HttpClient);

  public createPayment(
    payload: IPaymentDtoRequest
  ): Observable<{ message: string; paymentId: number }> {
    return this.http.post<{ message: string; paymentId: number }>(
      PaymentsEnum.CreatePayment,
      payload
    );
  }

  public createPayloadPayment(
    payload: IPaymentDtoRequest
  ): Observable<IPaymentCard> {
    return this.http.post<IPaymentCard>(
      PaymentsEnum.CreatePayloadPayment,
      payload
    );
  }

  public getPayments(): Observable<IPayment[]> {
    return this.http.get<IPayment[]>(PaymentsEnum.GetPayments);
  }
}
```

core/api/review/review-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class ReviewApiService {
  public http = inject(HttpClient);

  public getReviewByBook(bookId: string): Observable<IReview[]> {
    return
this.http.get<IReview[]>(`${ReviewEnum.GetReviewByBook}${bookId}`);
  }

  public getUserReview(userId: string): Observable<IReview[]> {
    return
this.http.get<IReview[]>(`${ReviewEnum.GetReviewByUser}${userId}`);
  }

  public deleteReview(reviewId: string): Observable<void> {
    return this.http.delete<void>(`${ReviewEnum.DeleteReview}${reviewId}`);
  }

  public addReview(payload: AddReviewDto): Observable<IReview> {
    return this.http.post<IReview>(ReviewEnum.AddReview, payload);
```

```
  }

  public editReview(payload: EditReviewDto): Observable<IReview> {
    return this.http.post<IReview>(ReviewEnum.EditReview, payload);
  }
}
```

core/api/user/user-api.service.ts

```
@Injectable({ providedIn: 'root' })
export class UserApiService {
  public http = inject(HttpClient);

  public getUser(userId: string): Observable<IUser> {
    return this.http.get<IUser>(`${UserEnum.GetUser}${userId}`);
  }

  public uploadPhoto(payload: FormData): Observable<void> {
    return this.http.post<void>(UserEnum.UploadPhoto, payload);
  }

  public updateUser(payload: UpdateUserDto): Observable<IUser> {
    return this.http.post<IUser>(UserEnum.UpdateUser, payload);
  }
}
```

core/components/book/book.component.html

```
<div class="container" (click)="navigateToBook()">
  <div class="book-picture">
    <img [ngSrc]="imageSrc" width="158" height="208" priority alt="book">
  </div>

  <div class="book-info">
    <h3>{{titleBook}}</h3>
    <p>{{authorName}}</p>
  </div>
</div>
```

core/components/book/book.component.ts

```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.scss'],
  standalone: true,
  imports: [NgOptimizedImage],
})
export class BookComponent {
  @Input() titleBook: string;
  @Input() authorName: string;
  @Input() imageSrc: string;
  @Input() bookId: string;

  public readonly router = inject(Router);

  public navigateToBook(): void {
    this.router.navigate(['book-details', this.bookId]);
  }
}
```

core/components/review/review.component.ts

```
<div class="review">
  <div class="review-user">
   <div class="review-user-info">
     <app-user-avatar
       [userSrc]="review.user.avatar"
       [userName]="review.user.name"
       [size]="32"
     ></app-user-avatar>

     <p class="review-user-name">{{ review.user.name }}</p>
   </div>

    <div class="review-user-wrap">
      <div class="review-user-rate">
        <ion-icon name="star"></ion-icon>
        <span class="review-user-rate-number">{{ review.rate }}/5</span>
      </div>

      @if (showMore) {
        <ion-icon (click)="setOpenActionSheet(true)" class="review-more"
name="ellipsis-vertical"></ion-icon>
      }
    </div>
  </div>

  <div class="review-description">
    <p class="review-description-text" [class.truncate]="isTruncate">{{
review.description }}</p>
  </div>

  <div class="review-date">
    <p class="review-date-time">{{ formatDate }}</p>
    <p class="review-date-time">{{ formatTime }}</p>
  </div>

  @if (isBookInterface._id) {
    <div class="review-book">
      <p>Назва книги: <span>{{ isBookInterface.title }}</span></p>
    </div>
  }
</div>

<ion-action-sheet
  [isOpen]="isActionSheetOpen"
  header="Дії для відгука"
  [buttons]="actionSheetButtons"
  (didDismiss)="setDisMissModal($event)"
></ion-action-sheet>
```

core/components/review/review.component.ts

```
@Component({
  selector: 'app-review',
  templateUrl: './review.component.html',
  styleUrls: ['./review.component.scss'],
  standalone: true,
  imports: [UserAvatarComponent, IonIcon, IonActionSheet],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
```

```
export class ReviewComponent {
  private reviewService = inject(ReviewService);
  private modalCtrl = inject(ModalController);

  @Input() review: IReview;
  @Input() isTruncate: boolean;
  @Input() showMore: boolean;

  public isActionSheetOpen = false;
  public actionSheetButtons = ActionSheetButtons;

  public get formatDate(): string {
    const formattedDate = dayjs(this.review.updatedAt).format('MMMM D,
YYYY');
    return formattedDate.charAt(0).toUpperCase() + formattedDate.slice(1);
  }

  public get formatTime(): string {
    return dayjs(this.review.updatedAt).format('HH.mm');
  }

  public get isBookInterface(): IBook {
    return this.review.book as IBook;
  }

  public setOpenActionSheet(isOpen: boolean) {
    this.isActionSheetOpen = isOpen;
  }

  public setDisMissModal(event: any) {
    this.setOpenActionSheet(false);
    if (event.detail?.data) {
      const action = event.detail.data.action;

      if (action === 'delete') {
        this.reviewService.removeReview(this.review._id);
      } else {
        this.openReviewModal();
      }
    }
  }

  public async openReviewModal(): Promise<void> {
    const bookId = this.isBookInterface._id || this.review.book;

    const modal = await this.modalCtrl.create({
      component: ReviewModalComponent,
      componentProps: {
        review: this.review,
        bookId,
      },
    });

    await modal.present();
  }
}
```

core/guards/auth.guard.ts

```
export const AuthGuard: CanActivateChildFn = (_route, _state) => {
  const authService = inject(AuthService);
  const onboardingService = inject(OnboardingService);
  const router = inject(Router);

  if (authService.isLoggedIn()) {
    return true;
  }

  if (!onboardingService.isOnboardingFlowFinish()) {
    router.navigate([RoutesEnum.Onboarding]);
  } else {
    router.navigate([RoutesEnum.Login]);
  }
  return false;
};
```

core/guards/onborading.guard.ts

```
export const OnboardingGuard: CanActivateFn = (_route, _state) => {
  const onboardingService = inject(OnboardingService);
  const router = inject(Router);

  if (onboardingService.isOnboardingFlowFinish()) {
    return true;
  }

  router.navigate([RoutesEnum.Onboarding]);
  return false;
};
```

core/interceptor/auth.interceptor.ts

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  private isRefreshing = false;
  private authService = inject(AuthService);
  private authApiService = inject(AuthApiService);
  private toastService = inject(ToastMobileService);

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    if (req.headers.has('InterceptorSkip')) {
      const headers = req.headers.delete('InterceptorSkip');
      return next.handle(req.clone({ headers }));
    }

    const baseUrl = environment.production
      ? BASEURL_PRODUCTION_CONST
      : BASEURL_CONST;
    const token = this.authService.getToken(LocalstorageEnum.AccessToken);

    if (req.url.endsWith('.svg')) {
      return next.handle(req);
    }

    const clonedRequest = req.clone({
      url: `${baseUrl}${req.url}`,
      setHeaders: token ? { Authorization: `Bearer ${token}` } : {},
```

```
    });

    return next.handle(clonedRequest).pipe(
      catchError((error) => {
        if (error instanceof HttpErrorResponse && error.status === 401) {
          return this.handle401Error(clonedRequest, next, error);
        }

        return throwError(() => error);
      })
    );
  }

  handle401Error(
    request: HttpRequest<any>,
    next: HttpHandler,
    error: any
  ): Observable<HttpEvent<any>> {
    if (!this.isRefreshing) {
      const accessToken = this.authService.getToken(
        LocalstorageEnum.AccessToken
      );
      const refreshToken = this.authService.getToken(
        LocalstorageEnum.RefreshToken
      );

      this.isRefreshing = true;

      if (accessToken && refreshToken) {
        return this.authApiService.refreshToken(refreshToken).pipe(
          switchMap((value) => {
            this.isRefreshing = false;
            this.authService.setToken(
              LocalstorageEnum.AccessToken,
              value.accessToken
            );
            this.authService.setToken(
              LocalstorageEnum.RefreshToken,
              value.refreshToken
            );

            const clonedRequest = request.clone({
              setHeaders: {
                Authorization: `Bearer ${value.accessToken}`,
              },
            });

            return next.handle(clonedRequest);
          }),
          catchError((error) => {
            this.isRefreshing = false;
            this.authService.handleLogout();
            this.toastService.infoToast('Ввійдіть в аккаунт ще раз!');
            return throwError(() => error);
          })
        );
      }
    }

    return throwError(() => error);
  }
}
```

core/modals/review-modal/review-modal.component.html

```
<div class="review">
  <div class="review-title">

    <ion-icon (click)="handleBack()" name="arrow-back-outline"></ion-icon>

    <h4>{{ review ? 'Редагувати відгук' : 'Додати відгук' }}</h4>
  </div>

  <form [formGroup]="reviewForm" (ngSubmit)="submitForm()">
    <h4 class="review-rate-title">Оцінка книги:</h4>
    <div class="review-rate">
      @for (star of stars; let index = $index; track star.id) {
        <div (click)="rate(index + 1)" class="review-star">
          @if (star.rate) {
            <ion-icon name="star"></ion-icon>
          } @else {
            <ion-icon name="star-outline"></ion-icon>
          }
        </div>
      }
    </div>

    <div class="review-description">
      <div class="review-description-title">Відгук:</div>
      <textarea
        formControlName="description"
        placeholder="Введіть ваш відгук..."
        rows="10"
        class="review-description-input"
      ></textarea>
    </div>

    <div class="review-submit">
      <button
        [disabled]="reviewForm.invalid || !reviewForm.dirty"
        class="review-submit-btn"
        type="submit"
      >
        {{ review ? 'Редагувати відгук' : 'Додати відгук' }}
      </button>
    </div>
  </form>
</div>

@if (reviewService.loading$ | async) {
  <app-loader></app-loader>
}
```

core/modals/review-modal/ review-modal.component.ts

```
@Component({
  selector: 'app-review-modal',
  templateUrl: './review-modal.component.html',
  styleUrls: ['./review-modal.component.scss'],
  standalone: true,
  imports: [IonIcon, ReactiveFormsModule, AsyncPipe, LoaderComponent],
})
export class ReviewModalComponent implements OnInit {
  @Input() public review: IReview;
  @Input() public bookId: IBook['_id'];
```

```
  public readonly reviewService = inject(ReviewService);
  private readonly modalCtrl = inject(ModalController);
  private readonly destroyRef = inject(DestroyRef);
  private readonly toastService = inject(ToastMobileService);
  private readonly fb = inject(FormBuilder);

  public reviewForm: FormGroup;
  public stars: IStar[] = stars;

  public ngOnInit(): void {
    this.initializeForm();
  }

  public handleBack(): Promise<boolean> {
    return this.modalCtrl.dismiss(null);
  }

  public rate(rating: number) {
    this.stars = this.stars.map((star, i) => ({ ...star, rate: rating > i
}));
    this.reviewForm.get('rate')?.setValue(rating);
    this.reviewForm.markAsDirty();
  }

  public submitForm(): void {
    if (this.reviewForm.invalid) return;

    const valueForm = this.reviewForm.value;

    if (this.review) {
      this.reviewService.editReview(
        this.review._id,
        valueForm.rate,
        valueForm.description,
        this.bookId
      );
    } else {
      this.reviewService.addReview(
        this.bookId,
        valueForm.rate,
        valueForm.description
      );
    }
  }

  private initializeForm(): void {
    this.reviewForm = this.fb.group({
      description: [this.review?.description || '', [Validators.required]],
      rate: [this.review?.rate || null, [Validators.required]],
    });

    if (this.review) {
      this.stars = this.stars.map((star, i) => ({
        ...star,
        rate: star.id > i,
      }));
    }

    this.reviewService.closeReviewModal$
      .pipe(takeUntilDestroyed(this.destroyRef))
      .subscribe(() => {
        this.handleBack();
```

```
        this.toastService.successToast(
          this.review ? 'Відгук був відредагований' : 'Відгук доданий'
        );
      });
  }
}
```

core/services/auth.service.ts

```
@Injectable({ providedIn: 'root' })
export class AuthService {
  private injector = inject(Injector);
  private authApi = inject(AuthApiService);
  private toastService = inject(ToastMobileService);
  private confirmService = inject(ConfirmService);
  private router = inject(Router);

  private _userService: UserService | null = null;

  private get userService(): UserService {
    if (!this._userService) {
      this._userService = this.injector.get(UserService);
    }
    return this._userService;
  }

  public getUserId(): string | null {
    return localStorage.getItem('userId');
  }

  public getToken(key: string): string | null {
    return localStorage.getItem(key);
  }

  public setToken(key: string, value: string) {
    localStorage.setItem(key, value);
  }

  private clearTokens(): void {
    localStorage.removeItem('accessToken');
    localStorage.removeItem('refreshToken');
    localStorage.removeItem('userId');
  }

  public isLoggedIn(): boolean {
    return !!this.getToken(LocalstorageEnum.AccessToken);
  }

  public handleLoginUser(loginUpDto: LoginDto) {
    this.authApi.loginUser(loginUpDto).subscribe(
      (tokens) => {
        this.setToken(LocalstorageEnum.AccessToken, tokens.accessToken);
        this.setToken(LocalstorageEnum.RefreshToken, tokens.refreshToken);
        this.setToken(LocalstorageEnum.UserId, tokens.userId);

        this.userService.getUser();
        this.router.navigate([RoutesEnum.Home]);
      },
      (error) => {
        this.toastService.errorToast(error.error.message);
        return of(null);
      }
```

```
    );
  }

  public handleSignUpUser(signUpDto: SignUpDto) {
    this.authApi.signUpUser(signUpDto).subscribe(
      () => {
        this.router.navigate([RoutesEnum.Login]);
        this.confirmService.showConfirmation(CONFIRM_REGISTRATION_CONST);
      },
      (error) => {
        this.toastService.errorToast(error.error.message);
        return of(null);
      }
    );
  }

  public handleLogout() {
    this.clearTokens();
    localStorage.removeItem('cart');
    this.router.navigate([RoutesEnum.Login]);
  }
}
```

core/services/book.service.ts

```
@Injectable({ providedIn: 'root' })
export class BookService {
  public bookApi = inject(BookApiService);

  private topBooks = new BehaviorSubject<IBook[] | null>(null);
  private loadingMost = new BehaviorSubject<boolean>(true);
  public topBooks$ = this.topBooks.asObservable();
  public loadingMost$ = this.loadingMost.asObservable();

  private searchBooks = new BehaviorSubject<IBook[] | null>(null);
  private loadingSearch = new Subject<boolean>();
  public searchBooks$ = this.searchBooks.asObservable();
  public loadingSearch$ = this.loadingSearch.asObservable();

  private detailBook = new BehaviorSubject<IBook | null>(null);
  private loadingDetail = new BehaviorSubject<boolean>(true);
  public detailBook$ = this.detailBook.asObservable();
  public loadingDetail$ = this.loadingDetail.asObservable();

  public getMostBooks(): void {
    this.loadingMost.next(true);

    this.bookApi
      .getMostBooks()
      .pipe(take(1))
      .subscribe((books) => {
        this.topBooks.next(books);
        this.loadingMost.next(false);
      });
  }

  public searchBook(value: string): void {
    this.loadingSearch.next(true);

    this.bookApi
      .getSearchBook(value)
      .pipe(take(1))
```

```
      .subscribe((books) => {
        this.searchBooks.next(books);
        this.loadingSearch.next(false);
      });
  }

  public getBook(id: string): void {
    this.loadingDetail.next(true);

    this.bookApi
      .getDetailBook(id)
      .pipe(take(1))
      .subscribe((book) => {
        this.detailBook.next(book);
        this.loadingDetail.next(false);
      });
  }

  public clearTopBooks(): void {
    this.topBooks.next(null);
  }

  public clearDetailBook(): void {
    this.detailBook.next(null);
  }

  public clearSearchBooks(): void {
    this.searchBooks.next(null);
  }
}
```

core/services/cart.service.ts

```
@Injectable({ providedIn: 'root' })
export class CartService {
  public toast = inject(ToastMobileService);
  public router = inject(Router);
  public confirmService = inject(ConfirmService);
  public paymentsApiService = inject(PaymentsApiService);
  public userService = inject(UserService);

  public selectedPayment: IPayment | null;

  private payments = new BehaviorSubject<IPayment[]>([]);
  public payments$ = this.payments.asObservable();

  private isLoading = new BehaviorSubject(false);
  public isLoading$ = this.isLoading.asObservable();

  public getCart(): TCart {
    return JSON.parse(localStorage.getItem('cart')!) || {};
  }

  public isExistBook(bookId: IBook['_id']): boolean {
    const cart = this.getCart();
    return !!cart[bookId];
  }

  public selectPayment(payment: IPayment) {
    this.selectedPayment = payment;
  }
```

```
  public clearSelectedPayment() {
    this.selectedPayment = null;
  }

  public countBookInCart(bookId: IBook['_id']): number {
    const cart = this.getCart();
    return cart[bookId].count;
  }

  public addToCart(book: IBook): boolean {
    const cart = this.getCart();

    if (this.getTotalCount >= 15) {
      this.toast.infoToast('В корзину можна додати максимум 15 книг');
      return false;
    }

    if (cart[book._id]) {
      cart[book._id].count++;
    } else {
      cart[book._id] = { ...book, count: 1 };
    }

    localStorage.setItem('cart', JSON.stringify(cart));
    return true;
  }

  public removeFromCart(bookId: IBook['_id']) {
    const cart = this.getCart();
    if (cart[bookId]) {
      cart[bookId].count--;

      if (cart[bookId].count <= 0) {
        delete cart[bookId];
      }

      localStorage.setItem('cart', JSON.stringify(cart));
    }
  }

  public clearCart() {
    localStorage.removeItem('cart');
  }

  public createPayment(payload: IPaymentDtoRequest): void {
    this.isLoading.next(true);

    this.paymentsApiService
      .createPayment(payload)
      .pipe(
        take(1),
        finalize(() => this.isLoading.next(false))
      )
      .subscribe(() => {
        this.userService.getUser();
        this.confirmService
          .showConfirmation({
            title: 'Замовлення успішно оформлено!',
            text: 'Статус замовлення можете переглянути у вкладці
"Замовлення"',
            confirmButtonText: 'Закрити',
            icon: 'success',
          })
```

```
        .then((result) => {
          if (result.isConfirmed) {
            this.router.navigate(['/home']);
          }
        });

      this.clearCart();
    });
}

public createPayloadPayment(payload: IPaymentDtoRequest): void {
  this.isLoading.next(true);

  this.paymentsApiService
    .createPayloadPayment(payload)
    .pipe(
      take(1),
      finalize(() => this.isLoading.next(false))
    )
    .subscribe((payload) => {
      this.openBrowser(payload.checkout_url);
    });
}

public getPayments(): void {
  this.isLoading.next(true);

  this.paymentsApiService
    .getPayments()
    .pipe(
      take(1),
      finalize(() => this.isLoading.next(false))
    )
    .subscribe((payments) => {
      this.payments.next(payments);
    });
}

public clearPayments(): void {
  this.payments.next([]);
}

public get getTotalCount() {
  const cart = this.getCart();

  if (!cart) {
    return 0;
  }

  return Object.values(cart).reduce((acc, item) => acc + item.count, 0);
}

public get cartArray(): IBook[] {
  return Object.values(this.getCart());
}

public async paymentSuccess(data: IPaymentSocket): Promise<void> {
  const userId = this.userService.getUserId();
  if (data.id !== userId) return;

  await Browser.close();
  if (data.status !== 'success') {
    this.toast.errorToast('Виникла проблема під час оплати!');
```

```
      return;
    }

    this.clearCart();
    this.userService.getUser();
    this.router.navigate(['/payment-success']);
  }

  private async openBrowser(checkoutUrl: string): Promise<void> {
    if (Capacitor.isNativePlatform()) {
      await Browser.open({ url: checkoutUrl });
    } else {
      window.location.href = checkoutUrl;
    }
  }
}
```

core/services/favorite.service.ts

```
@Injectable({ providedIn: 'root' })
export class FavoriteService {
  private favoriteApi = inject(FavoriteApiService);
  private toastService = inject(ToastMobileService);

  private favoriteBooks = new BehaviorSubject<IBook[]>([]);
  public favoriteBooks$ = this.favoriteBooks.asObservable();

  private loading = new BehaviorSubject<boolean>(false);
  public loading$ = this.loading.asObservable();

  public isLoading(): boolean {
    return this.loading.value;
  }

  public getFavorite(): void {
    this.loading.next(true);

    this.favoriteApi
      .getFavorite()
      .pipe(take(1))
      .subscribe((favorite) => {
        this.loading.next(false);
        this.favoriteBooks.next(favorite);
        console.log(favorite);
      });
  }

  public clearFavorite(): void {
    this.favoriteBooks.next([]);
  }

  public addFavorite(bookId: IBook['_id']): void {
    this.loading.next(true);

    this.favoriteApi
      .addFavorite(bookId)
      .pipe(
        take(1),
        catchError((error) => {
          this.loading.next(false);
          this.toastService.errorToast(error.error.message);
          return of();
```

```
        })
      )
      .subscribe(() => {
        this.loading.next(false);
        this.toastService.successToast('Було додано до улюблених');
      });
  }

  public removeFavorite(bookId: IBook['_id']): void {
    this.loading.next(true);

    this.favoriteApi
      .removeFavorite(bookId)
      .pipe(
        take(1),
        catchError((error) => {
          this.loading.next(false);
          this.toastService.errorToast(error.error.message);
          return of();
        })
      )
      .subscribe(() => {
        this.loading.next(false);
        this.toastService.successToast('Було успішно видалено з улюблених');
      });
  }
}
```

core/services/nova-poshta.service.ts

```
@Injectable({ providedIn: 'root' })
export class NovaPoshtaService {
  private http = inject(HttpClient);

  private cities = new BehaviorSubject<ICity[]>([]);
  public cities$ = this.cities.asObservable();

  private address = new BehaviorSubject<any>([]);
  public address$ = this.address.asObservable();

  private URL = 'https://api.novaposhta.ua/v2.0/json/';
  private API = environment.apiNovaPoshta;

  private headers = new HttpHeaders().set('InterceptorSkip', '');

  public getCities(query: string) {
    const body = {
      apiKey: this.API,
      modelName: 'Address',
      calledMethod: 'getCities',
      methodProperties: {
        FindByString: query,
        Limit: 20,
      },
    };

    this.http
      .post<any>(this.URL, body, { headers: this.headers })
      .pipe(take(1))
      .subscribe((cities) => {
        this.cities.next(cities.data);
      });
```

```
  }

  public getWareHouses(cityRef: string, value: string) {
    const body = {
      apiKey: this.API,
      modelName: 'Address',
      calledMethod: 'getWarehouses',
      methodProperties: { CityRef: cityRef, FindByString: value },
    };

    this.http
      .post<any>(this.URL, body, { headers: this.headers })
      .pipe(take(1))
      .subscribe((wareHouses) => {
        this.address.next(wareHouses.data);
      });
  }

  public clearCities(): void {
    this.cities.next([]);
  }

  public clearAddress(): void {
    this.address.next([]);
  }
}
```

core/services/review.service.ts

```
@Injectable({ providedIn: 'root' })
export class ReviewService {
  private reviewApiService = inject(ReviewApiService);
  private toastService = inject(ToastMobileService);

  private reviewsBook = new BehaviorSubject<IReview[] | null>(null);
  public reviewsBook$ = this.reviewsBook.asObservable();

  private reviewsProfile = new BehaviorSubject<IReview[] | null>(null);
  public reviewsProfile$ = this.reviewsProfile.asObservable();

  private closeReviewModal = new Subject<void>();
  public closeReviewModal$ = this.closeReviewModal.asObservable();

  private handleRemoveProfile = new Subject<void>();
  public handleRemoveProfile$ = this.handleRemoveProfile.asObservable();

  private loading = new BehaviorSubject<boolean>(true);
  public loading$ = this.loading.asObservable();

  public getBookId(bookId: string) {
    this.loading.next(true);

    this.reviewApiService
      .getReviewByBook(bookId)
      .pipe(take(1))
      .subscribe((reviews) => {
        this.loading.next(false);
        this.reviewsBook.next(reviews);
      });
  }

  public getReviewUser() {
```

```
  const userId = localStorage.getItem('userId');
  this.loading.next(true);

  if (!userId) return;

  this.reviewApiService
    .getUserReview(userId)
    .pipe(take(1))
    .subscribe((reviews) => {
      this.loading.next(false);
      this.reviewsProfile.next(reviews);
    });
}

public addReview(bookId: string, rate: number, description: string) {
  this.loading.next(true);
  const userId = localStorage.getItem('userId');

  const payload: AddReviewDto = {
    bookId,
    rate,
    description,
    userId: userId || '',
  };

  this.reviewApiService
    .addReview(payload)
    .pipe(
      take(1),
      catchError((error) => {
        this.loading.next(false);
        this.toastService.errorToast(error.error.message);
        return of();
      })
    )
    .subscribe(() => {
      this.closeReviewModal.next();
      this.getBookId(bookId);
      this.getReviewUser();
    });
}

public editReview(
  reviewId: string,
  rate: number,
  description: string,
  bookId: string
) {
  this.loading.next(true);
  const userId = localStorage.getItem('userId');

  const payload: EditReviewDto = {
    reviewId,
    rate,
    description,
    userId: userId || '',
  };

  this.reviewApiService
    .editReview(payload)
    .pipe(
      take(1),
      catchError((error) => {
```

```
          this.loading.next(false);
          this.toastService.errorToast(error.error.message);
          return of();
        })
      )
      .subscribe(() => {
        this.closeReviewModal.next();
        this.getBookId(bookId);
        this.getReviewUser();
      });
  }

  public removeReview(reviewId: string) {
    this.loading.next(true);

    this.reviewApiService
      .deleteReview(reviewId)
      .pipe(take(1))
      .subscribe(() => {
        this.loading.next(false);
        this.handleRemoveProfile.next();

        const allReviews = this.reviewsBook.value;
        if (allReviews) {
          const filteredReview = allReviews.filter(
            (review) => review._id !== reviewId
          );

          this.reviewsBook.next(filteredReview);
        }
      });
  }

  public clearUserReview(): void {
    this.reviewsProfile.next(null);
  }
}
```

core/services/user.service.ts

```
@Injectable({ providedIn: 'root' })
export class UserService {
  public userApiService = inject(UserApiService);
  public authService = inject(AuthService);
  public toastService = inject(ToastMobileService);

  private user = new BehaviorSubject<IUser | null>(null);
  public user$ = this.user.asObservable();

  private loading = new BehaviorSubject<boolean>(true);
  public loading$ = this.loading.asObservable();

  public getUser(): void {
    const userId = localStorage.getItem('userId');

    if (!userId) {
      this.authService.handleLogout();
      return;
    }

    this.userApiService
      .getUser(userId)
```

```
      .pipe(finalize(() => this.loading.next(false)))
      .subscribe((user) => {
        this.user.next(user);
      });
  }

  public getUserId(): string {
    return localStorage.getItem('userId')!;
  }

  public get getUserEntity(): IUser | null {
    return this.user.value;
  }

  public updateUser(email: string, name: string, file?: File): void {
    const userId = localStorage.getItem('userId');

    if (!userId) {
      this.authService.handleLogout();
      return;
    }

    this.loading.next(true);

    if (file) {
      const formData = this.updateAvatar(userId, file);

      this.userApiService.uploadPhoto(formData).subscribe({
        next: () => this.updateUserDetails({ email, name, userId }),
        error: (error) => {
          console.error('Error uploading photo:', error);
          this.loading.next(false);
        },
      });
    } else {
      this.updateUserDetails({ email, name, userId });
    }
  }

  public updateAvatar(userId: string, file: File): FormData {
    const formData = new FormData();
    formData.append('file', file);
    formData.append('userId', userId);

    return formData;
  }

  private updateUserDetails(payload: UpdateUserDto): void {
    this.userApiService.updateUser(payload).subscribe({
      next: (user) => {
        this.toastService.successToast('Дані були оновлені!');
        this.user.next(user);
      },
      error: (error) => console.error('Error updating user:', error),
      complete: () => this.loading.next(false),
    });
  }
}
```

pages/book-details/book-details.component.html

```
<div class="container">
  <div class="header">
    <ion-icon (click)="handleBack()" name="arrow-back-outline"></ion-icon>

    <h2>Деталі книги</h2>

    <div class="cart" (click)="navigateToCart()">
      <ion-icon name="cart-outline"></ion-icon>
      @if (cartService.getTotalCount) {
        <span class="count">{{ cartService.getTotalCount }}</span>
      }
    </div>

  </div>

  @if (book) {
    <div class="book-image">
      <img [ngSrc]="book.image" width="158" height="208" priority alt="book">
    </div>
  } @else {
    <ngx-skeleton-loader class="book-image" [theme]="{'height': '198px',
'width': '160px', 'border-radius': '12px'}"></ngx-skeleton-loader>
  }

  @if (book) {
    <div class="book-wrapper">
      <app-book-name
        [genreTitle]="book.genre.title"
        [title]="book.title"
        [price]="book.price"
      ></app-book-name>

      <app-info-item [book]="book"></app-info-item>

      <app-book-description [description]="book.description"></app-book-
description>

      <app-book-reviews
        [bookId]="book._id"
        [loading]="reviewService.loading$ | async"
        [reviews]="reviewService.reviewsBook$ | async"
      ></app-book-reviews>

      <app-book-footer [isFavorite]="book.isFavorite" [book]="book"></app-
book-footer>
    </div>
  } @else {
    <div class="book-wrapper-load">
      <ngx-skeleton-loader [theme]="{'height': '800px', 'width': '100%',
'border-radius': '12px'}"></ngx-skeleton-loader>
    </div>
  }
</div>
```

pages/book-details/book-details.component.ts

```
@Component({
  selector: 'app-book-details',
  templateUrl: './book-details.component.html',
  styleUrls: ['./book-details.component.scss'],
  standalone: true,
  imports: [
```

```
    IonIcon,
    NgOptimizedImage,
    NgxSkeletonLoaderModule,
    InfoItemComponent,
    BookNameComponent,
    BookDescriptionComponent,
    BookReviewsComponent,
    AsyncPipe,
    BookFooterComponent,
  ],
})
export class BookDetailsComponent implements OnInit, OnDestroy {
  public readonly reviewService = inject(ReviewService);
  public readonly cartService = inject(CartService);
  private readonly route = inject(ActivatedRoute);
  private readonly router = inject(Router);
  private readonly location = inject(Location);
  private readonly destroyRef = inject(DestroyRef);
  private readonly bookService = inject(BookService);

  public book: IBook | null = null;

  public ngOnInit(): void {
    this.initializeSubscribe();
  }

  public ngOnDestroy(): void {
    this.bookService.clearDetailBook();
  }

  public navigateToCart(): void {
    if (!this.cartService.getTotalCount) return;
    this.router.navigate(['cart']);
  }

  public handleBack(): void {
    this.location.back();
  }

  private initializeSubscribe(): void {
    this.route.params
      .pipe(takeUntilDestroyed(this.destroyRef))
      .subscribe((params) => {
        if (!params.id) this.router.navigate(['home']);

        this.bookService.getBook(params.id);
        this.reviewService.getBookId(params.id);
      });

    this.bookService.detailBook$
      .pipe(takeUntilDestroyed(this.destroyRef))
      .subscribe((book) => {
        this.book = book;
      });
  }
}
```

pages/cart/cart.component.ts

```
@Component({
  selector: 'app-cart',
  templateUrl: './cart.component.html',
```

```
  styleUrls: ['./cart.component.scss'],
  imports: [
    IonIcon,
    NgOptimizedImage,
    ReactiveFormsModule,
    NgxMaskDirective,
    SearchDropdownCityComponent,
    SearchDropdownAddressComponent,
    AsyncPipe,
    LoaderComponent,
  ],
  standalone: true,
})
export class CartComponent implements OnInit, OnDestroy {
  public cartService = inject(CartService);
  private router = inject(Router);
  private userService = inject(UserService);
  private fb = inject(FormBuilder);

  public paymentForm: FormGroup;
  public showError = false;
  public maxValue = 0;
  public totalValue = 0;
  public cityRef: ICity['Ref'];
  private socket: Socket;

  private readonly serverUrl = environment.production
    ? BASEURL_PRODUCTION_CONST
    : BASEURL_CONST;

  constructor() {
    if (Capacitor.isNativePlatform()) {
      this.socket = io(this.serverUrl);
    }
  }

  public ngOnInit(): void {
    this.paymentForm = this.fb.group({
      phone: ['', [Validators.required]],
      city: ['', [Validators.required]],
      department: ['', [Validators.required]],
      bonus: ['0', [Validators.required]],
      typePayment: ['', [Validators.required]],
    });

    this.setMaxValue();

    if (Capacitor.isNativePlatform()) {
      this.listenStatus();
    }
  }

  public ngOnDestroy() {
    if (Capacitor.isNativePlatform()) {
      this.socket.disconnect();
    }
  }

  public get showErrorPhone(): boolean {
    return this.showError && !!this.paymentForm.get('phone')?.errors;
  }

  public get showErrorCity(): boolean {
```

```
  return this.showError && !!this.paymentForm.get('city')?.errors;
}

public get showErrorDepartment(): boolean {
  return this.showError && !!this.paymentForm.get('department')?.errors;
}

public get bonusBalls(): number {
  if (!this.userService.getUserEntity) return 0;
  return this.userService.getUserEntity.bookPoints;
}

public get getTotalAmount(): number {
  return this.totalValue - +this.paymentForm.get('bonus')?.value;
}

public handleBack(): void {
  this.router.navigate(['home']);
}

public selectCity(city: ICity): void {
  if (!city) {
    this.paymentForm.get('city')?.setValue('');
    this.cityRef = '';
    return;
  }

  this.cityRef = city.Ref;
  this.paymentForm.get('city')?.setValue(city.Description);
}

public onInputBonusChange(event: any) {
  let inputValue = event.target.value;
  inputValue = inputValue.replace(/[^0-9]/g, '');

  if (inputValue !== '') {
    const numericValue = parseInt(inputValue, 10);
    if (numericValue > this.maxValue) {
      inputValue = this.maxValue.toString();
    }
  }

  this.paymentForm.get('bonus')?.setValue(inputValue);
}

public selectAddress(address: any): void {
  if (!address) {
    this.paymentForm.get('department')?.setValue('');
    return;
  }

  this.paymentForm.get('department')?.setValue(address.Description);
}

public addToCartItem(book: IBook): void {
  this.cartService.addToCart(book);
  this.setTotalAmount();
}

public removeCartItem(bookId: IBook['_id']): void {
  this.cartService.removeFromCart(bookId);
  this.setTotalAmount();
}
```

```
  public submitForm(): void {
    if (this.paymentForm.invalid) {
      this.showError = true;
      return;
    }

    this.showError = false;
    const values = this.paymentForm.value;
    const cart = this.cartService.cartArray;
    const userId = this.userService.getUserId();

    const payload: IPaymentDtoRequest = {
      userId,
      bonus: +values.bonus,
      phoneNumber: values.phone,
      city: values.city,
      department: values.department,
      typePayment: values.typePayment,
      books: cart.map((book: IBook) => ({ _id: book._id, count: book.count
})),
    };

    if (payload.typePayment === 'Card') {
      this.cartService.createPayloadPayment(payload);
    } else {
      this.cartService.createPayment(payload);
    }
  }

  private setTotalAmount(): void {
    this.totalValue = this.cartService.cartArray.reduce(
      (acc, item) => acc + item.price * item.count,
      0
    );
  }

  private listenStatus(): void {
    this.socket.on('statusUpdate', (data: IPaymentSocket) => {
      this.cartService.paymentSuccess(data);
    });
  }

  private setMaxValue(): void {
    this.setTotalAmount();
    const maxValueFromTotal = this.totalValue * 0.2;

    if (!this.userService.getUserEntity) return;

    this.maxValue =
      this.userService.getUserEntity.bookPoints < maxValueFromTotal
        ? this.userService.getUserEntity?.bookPoints
        : maxValueFromTotal;
  }
}
```

pages/cart/cart.component.html

```
<div class="container" [class.container-
full]="!cartService.cartArray.length">
  <div class="header">
    <ion-icon (click)="handleBack()" name="arrow-back-outline"></ion-icon>
```

```
    <h2>Корзина книг</h2>
  </div>

  <div class="books">
    <div class="books-wrapper">
      @for(book of cartService.cartArray; track book._id) {
        <div class="book">
          <img [ngSrc]="book.image" height="100" width="70" alt="book">
          <div class="book-info">
            <div class="book-title">{{ book.title }}</div>
            <div class="book-author">{{ book.author.name }}</div>

            <div class="book-price">
              <div class="price">{{ book.price * book.count }} ₴</div>

              <div class="buttons">
                <button class="toggle-cart minus"
(click)="removeCartItem(book._id)">
                  <span>-</span>
                </button>
                <span class="count">{{book.count}}</span>
                <button class="toggle-cart" (click)="addToCartItem(book)">
                  <span>+</span>
                </button>
              </div>
            </div>
          </div>
        </div>
      } @empty {
        <div class="empty-state">
          <img ngSrc="assets/image/empty-cart.webp" width="200" height="200"
alt="empty">
          <div class="empty-state-title">Корзина пуста!</div>
          <div class="empty-state-desc">Додай нові книги, щоб оформити
замовлення</div>
        </div>
      }
    </div>
  </div>

  @if (cartService.cartArray.length) {
    <form [formGroup]="paymentForm" class="form">
      <div class="input-wrapper">
        <label for="phone">Телефон</label>
        <input
          [class.input-error]="showErrorPhone"
          formControlName="phone"
          class="email"
          placeholder="Введіть номер телефону..."
          id="phone"
          type="text"
          mask="(000) 000-0000"
        />
        @if(showErrorPhone) {
          @if(paymentForm.get('phone')?.errors?.required) {
            <span class="error">Це поле обов’язкове</span>
          } @else {
            <span class="error">Неправильний номер телефону</span>
          }
        }
      </div>

      <div class="delivery_method">
```

```
        <label>Спосіб доставки</label>
        <div class="methods">
          <div class="method">
            <input class="disabled-input" type="radio" checked disabled />
            <div class="method-name">
              <span>Нова пошта</span>
              <img ngSrc="assets/image/nova-poshta.png" width="20"
height="20" alt="logo">
            </div>
          </div>
        </div>

      <div class="input-wrapper">
        <label>Місто</label>
        <app-search-dropdown-city (handleCity)="selectCity($event)"></app-
search-dropdown-city>

        @if(showErrorCity) {
          @if(paymentForm.get('city')?.errors?.required) {
            <span class="error">Це поле обов’язкове</span>
          }
        }
      </div>

      <div class="input-wrapper">
        <label>Віділення Нової Пошти</label>
        <app-search-dropdown-address [cityRef]="cityRef"
(handleAddress)="selectAddress($event)"></app-search-dropdown-address>

        @if(showErrorDepartment) {
          @if(paymentForm.get('department')?.errors?.required) {
            <span class="error">Це поле обов’язкове</span>
          }
        }
      </div>

      <div class="pay_method">
        <label class="title">Спосіб оплати</label>
        <div class="methods">
          <div class="method">
            <input class="radio-pay" type="radio" id="Cash" value="Cash"
formControlName="typePayment">
            <label for="Cash" class="description">Оплата готівкою ( при
отримання замовлення )</label>
          </div>

          <div class="method">
            <input class="radio-pay" type="radio" id="Card" value="Card"
formControlName="typePayment">
            <label for="Card" class="description">Картою ( оплатити одразу
)</label>
          </div>
        </div>
      </div>

      <div class="bonus_system">
        <div class="title">Бонусні бали ( максимум 20% ). У вас на балансі
стільки балів: <span>{{bonusBalls}}</span></div>
        <input
          type="text"
          formControlName="bonus"
          [attr.max]="maxValue"
```

```
          (input)="onInputBonusChange($event)"
          placeholder="Введіть ваші бали" />
      </div>
    </form>

    <div class="confirm">
      <div class="confirm-total">Загальна вартість: <span>{{ getTotalAmount
}} ₴</span></div>
      <button class="confirm-btn" (click)="submitForm()"
[disabled]="paymentForm.invalid">Оплатити</button>
    </div>
  }
</div>

@if (cartService.isLoading$ | async) {
  <app-loader></app-loader>
}
```

pages/favorite/favorite.component.html

```
<div class="container">
  <div class="header">
    <ion-icon (click)="handleBack()" name="arrow-back-outline"></ion-icon>
    <h2>Улюбленні книги</h2>
  </div>

  <div class="wrapper">
    @if(!(favoriteService.loading$ | async)) {
      <div class="book-container">
        @for (book of favoriteService.favoriteBooks$ | async; track book._id)
{
          <app-book [bookId]="book._id" [titleBook]="book.title"
[authorName]="book.author.name" [imageSrc]="book.image"></app-book>
        } @empty {
          <div class="empty-state">
            <img ngSrc="assets/image/empty-state.svg" alt="empty-state"
width="160" height="160">
            <h4>Ви ще не додали улюблені книги в список 😢</h4>
          </div>
        }
      </div>
    }

    @if(favoriteService.loading$ | async) {
      <ngx-skeleton-loader class="book-container" count="6"
[theme]="{'height': '260px', 'width': '160px', 'border-radius':
'12px'}"></ngx-skeleton-loader>
    }
  </div>
</div>
```

pages/favorite/favorite.component.ts

```
@Component({
  selector: 'app-favorite',
  templateUrl: './favorite.component.html',
  styleUrls: ['./favorite.component.scss'],
  standalone: true,
  imports: [
    IonIcon,
    AsyncPipe,
    BookComponent,
```

```
    NgOptimizedImage,
    NgxSkeletonLoaderModule,
  ],
})
export class FavoriteComponent implements OnInit, OnDestroy {
  public router = inject(Router);
  public favoriteService = inject(FavoriteService);

  public ngOnInit() {
    this.favoriteService.getFavorite();
  }

  public ngOnDestroy() {
    this.favoriteService.clearFavorite();
  }

  public handleBack(): void {
    this.router.navigate(['profile']);
  }
}
```

pages/home/home.component.ts

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss'],
  standalone: true,
  imports: [
    HeaderHomeComponent,
    WelcomePostComponent,
    IonIcon,
    ReactiveFormsModule,
    GenresFilterComponent,
    AsyncPipe,
    MostBooksComponent,
    SearchBooksComponent,
    IonInput,
  ],
})
export class HomeComponent implements OnInit, OnDestroy {
  public genreService = inject(GenreService);
  public bookService = inject(BookService);

  public searchInput = new FormControl<string>('');
  public isFocus = false;

  public readonly allGenres$ = this.genreService.allGenres$;
  public readonly allMostBooks$ = this.bookService.topBooks$;
  public readonly searchBooks$ = this.bookService.searchBooks$;
  public readonly loadingSearch$ = this.bookService.loadingSearch$;
  public readonly loadingMost$ = this.bookService.loadingMost$;

  public ngOnInit(): void {
    this.genreService.getAllGenres();
    this.bookService.getMostBooks();

    this.searchInput.valueChanges
      .pipe(distinctUntilChanged(), debounceTime(400))
      .subscribe((result) => {
        if (!result || result.length <= 3) {
          this.bookService.clearSearchBooks();
```

```
          return;
        }

        this.bookService.searchBook(result);
      });
  }

  public ngOnDestroy(): void {
    this.genreService.clearGenres();
    this.bookService.clearTopBooks();
    this.bookService.clearSearchBooks();
  }

  public clearInput(): void {
    this.searchInput.setValue('');
  }

  public filterGenre(genre: ISelectGenre): void {
    this.genreService.filterByGenre(genre);
  }
}
```

pages/home/home.component.html

```
<div class="container">
  <div class="container-search-top" [class.hide]="searchInput.value ||
isFocus">
    <app-header-home></app-header-home>

    <app-welcome-post></app-welcome-post>
  </div>

  <div class="search">
    <div class="search-input">
      <ion-icon name="search-outline"></ion-icon>
      <ion-input [formControl]="searchInput" (ionFocus)="isFocus = true"
(ionBlur)="isFocus = false" type="text" placeholder="Шукайте будь-які
книги"></ion-input>

      @if (searchInput.value) {
        <ion-icon (click)="clearInput()" name="close-outline"></ion-icon>
      }
    </div>
  </div>

  @if(!searchInput.value && !isFocus) {
    <app-genres-filter [genres]="allGenres$ | async"
(filterGenre)="filterGenre($event)"></app-genres-filter>

    <app-most-books [loading]="loadingMost$ | async"
[mostBooks]="allMostBooks$ | async"></app-most-books>
  }

  @if (searchInput.value) {
    <app-search-books [loading]="loadingSearch$ | async"
[searchBooks]="searchBooks$ | async"></app-search-books>
  }
</div>
```

pages/login/login.component.html

```
<div class="login">
  <div class="container">
    <div class="info">
      <h2 class="title">Логін</h2>
      <p class="description">Ласкаво просимо до бібліотечної крамнички!</p>
    </div>
    <form [formGroup]="loginForm" (ngSubmit)="submitForm()">
      <div class="input-wrapper">
        <label for="email">Електронна адреса</label>
        <input
          [class.input-error]="showErrorEmail"
          formControlName="email"
          class="email"
          placeholder="Введіть пошту..."
          id="email"
          type="text"
        />
        @if(showErrorEmail) {
          @if(loginForm.get('email')?.errors?.required) {
            <span class="error">Це поле обов’язкове</span>
          } @else {
            <span class="error">Неправильна пошта</span>
          }
        }
      </div>

      <div class="input-wrapper">
        <label for="password">Пароль</label>
        <div [class.input-error]="showErrorPassword" class="password-
wrapper">
          <input formControlName="password" class="password"
placeholder="Введіть пароль..." id="password" [type]="showPassword ? 'text' :
'password'" />
          <ion-icon (click)="updateShowPassword()" [name]="showPassword ?
'eye-off-outline' : 'eye-outline'"></ion-icon>
        </div>
        @if(showErrorPassword) {
          @if(loginForm.get('password')?.errors?.required) {
            <span class="error">Це поле обов’язкове</span>
          }
        }
      </div>

      <button type="submit" class="submit">Вхід</button>
    </form>

    <div class="line">
      <span>або увійдіть через</span>
    </div>

    <button [disabled]="true" class="google">
      <img ngSrc="assets/image/google.png" alt="google" priority="true"
width="25" height="25">
      <span>Продовжуйте з Google</span>
    </button>
  </div>

  <button class="outline" (click)="createAccount()">Створити аккаунт</button>
</div>
```

pages/login/login.component.ts

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
  standalone: true,
  imports: [ReactiveFormsModule, IonIcon, NgOptimizedImage],
})
export class LoginComponent implements OnInit {
  public loginForm: FormGroup;
  public showPassword = false;
  public showError = false;

  constructor(
    private fb: FormBuilder,
    private router: Router,
    private authService: AuthService
  ) {}

  public get showErrorEmail(): boolean {
    return this.showError && !!this.loginForm.get('email')?.errors;
  }

  public get showErrorPassword(): boolean {
    return this.showError && !!this.loginForm.get('password')?.errors;
  }

  public ngOnInit(): void {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required]],
    });
  }

  public updateShowPassword(): void {
    this.showPassword = !this.showPassword;
  }

  public submitForm(): void {
    if (this.loginForm.invalid) {
      this.showError = true;
      return;
    }

    const payload = this.loginForm.value;
    this.authService.handleLoginUser(payload);
  }

  public createAccount(): void {
    this.router.navigate([RoutesEnum.SignUp]);
  }
}
```

pages/payments/payments.component.html

```
<div class="container">
  <h3 class="title">Список замовлень</h3>

  @if (cartService.isLoading$ | async) {
    <ngx-skeleton-loader class="payments" count="6" [theme]="{'height':
'250px', 'width': '100%', 'border-radius': '12px'}"></ngx-skeleton-loader>
  } @else {
    <div class="payments">
```

```
      @for (payment of cartService.payments$ | async; track payment._id) {
        <app-payment-item [payment]="payment"></app-payment-item>
      } @empty {
        <div class="empty-state">
          <img ngSrc="assets/image/empty-state.svg" priority alt="empty-
state" width="160" height="160">
          <h4>Ви ще не зробили жодного замовлення 😢</h4>
        </div>
      }
    </div>
  }
</div>
```

pages/payments/payments.component.ts

```
@Component({
  selector: 'app-payments',
  templateUrl: './payments.component.html',
  styleUrls: ['./payments.component.scss'],
  standalone: true,
  imports: [
    AsyncPipe,
    NgxSkeletonLoaderModule,
    NgOptimizedImage,
    PaymentItemComponent,
  ],
})
export class PaymentsComponent implements OnInit, OnDestroy {
  public cartService = inject(CartService);

  public ngOnInit() {
    this.cartService.getPayments();
  }

  public ngOnDestroy() {
    this.cartService.clearPayments();
  }
}
```

pages/profile/profile.component.html

```
<div class="container">
  <h3 class="title">Профіль</h3>

  <app-user-info [user]="user$ | async"></app-user-info>

  <div class="container-items">
    <h4 class="container-items-title">Аккаунт</h4>
    <div class="items">
      @for(item of PROFILE_TABS_ACCOUNT; track item.title) {
        <app-profile-item
          [title]="item.title"
          [icon]="item.icon"
          (handleClick)="handleProfileTab(item.label)"
        ></app-profile-item>
      }
    </div>
  </div>

  <div class="container-items">
    <h4 class="container-items-title">Загальні</h4>
```

```
    <div class="items">
      @for(item of PROFILE_TABS_GENERAL; track item.title) {
        <app-profile-item
          [title]="item.title"
          [icon]="item.icon"
          [classes]="item.label === eProfileTabEnum.Logout ? 'logout' : ''"
          (handleClick)="handleProfileTab(item.label)"
        ></app-profile-item>
      }
    </div>
  </div>
</div>
```

pages/profile/profile.component.ts

```
@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.scss'],
  standalone: true,
  imports: [UserInfoComponent, AsyncPipe, ProfileItemComponent],
})
export class ProfileComponent {
  public readonly userService = inject(UserService);
  public readonly authService = inject(AuthService);
  public readonly router = inject(Router);
  private readonly confirmService = inject(ConfirmService);

  public readonly user$ = this.userService.user$;

  public readonly PROFILE_TABS_ACCOUNT = PROFILE_TABS_ACCOUNT;
  public readonly PROFILE_TABS_GENERAL = PROFILE_TABS_GENERAL;
  public readonly eProfileTabEnum = ProfileTabEnum;

  public async handleProfileTab(action: ProfileTabEnum): Promise<void> {
    switch (action) {
      case ProfileTabEnum.Logout:
        const result = await this.confirmService.showConfirmation(
          CONFIRM_LOGOUT
        );
        if (result.isConfirmed) {
          this.authService.handleLogout();
        }
        return;
      case ProfileTabEnum.ProfileDetail:
        this.router.navigate(['profile', 'details']);
        return;
      case ProfileTabEnum.ProfileReview:
        this.router.navigate(['profile', 'reviews']);
        return;
      case ProfileTabEnum.Favorite:
        this.router.navigate(['favorite']);
        return;
    }
  }
}
```

pages/profile-detail/profile-detail.component.ts

```
@Component({
  selector: 'app-profile-detail',
```

```
  templateUrl: './profile-detail.component.html',
  styleUrls: ['./profile-detail.component.scss'],
  standalone: true,
  imports: [
    IonIcon,
    UserAvatarComponent,
    AsyncPipe,
    NgxSkeletonLoaderModule,
    ReactiveFormsModule,
    LoaderComponent,
  ],
})
export class ProfileDetailComponent implements OnInit {
  private readonly router = inject(Router);
  private readonly userService = inject(UserService);
  private readonly cdkRef = inject(ChangeDetectorRef);
  private readonly destroyRef = inject(DestroyRef);
  private readonly fb = inject(FormBuilder);

  public readonly loading$ = this.userService.loading$;

  public user: IUser;
  public uploadedSrc: string;
  public fileToUpload: File;
  public profileForm: FormGroup;
  public isEdit = false;
  public showError = false;

  public ngOnInit(): void {
    this.initializeForm();
    this.getUser();
  }

  public get showErrorName(): boolean {
    return this.showError && !!this.profileForm.get('name')?.errors;
  }

  public get showErrorEmail(): boolean {
    return this.showError && !!this.profileForm.get('email')?.errors;
  }

  public handleBack(): void {
    this.router.navigate(['profile']);
  }

  public handleClickEdit(): void {
    if (!this.isEdit) {
      this.isEdit = true;
      this.toggleForm();
      return;
    }

    if (this.profileForm.invalid && this.isEdit) {
      this.showError = true;
      return;
    }

    this.cdkRef.detectChanges();
    this.showError = false;

    if (!this.profileForm.dirty && !this.fileToUpload) {
      this.isEdit = false;
      this.toggleForm();
```

```
      return;
    }

    const { email, name } = this.profileForm.value;

    this.userService.updateUser(email, name, this.fileToUpload);
  }

  public uploadPhoto({ imageSrc, file }: IUploadPhoto): void {
    this.uploadedSrc = imageSrc as string;
    this.fileToUpload = file;
  }

  private setFormFields(): void {
    this.profileForm.setValue({ name: this.user.name, email: this.user.email
});
    this.toggleForm();
  }

  private toggleForm(): void {
    const nameForm = this.profileForm.get('name');
    const emailForm = this.profileForm.get('email');

    if (this.isEdit) {
      nameForm?.enable();
      emailForm?.enable();
    } else {
      nameForm?.disable();
      emailForm?.disable();
    }
  }

  private getUser(): void {
    this.userService.user$
      .pipe(takeUntilDestroyed(this.destroyRef))
      .subscribe((user) => {
        if (!user) return;

        this.user = user;
        this.setFormFields();
      });
  }

  private initializeForm(): void {
    this.profileForm = this.fb.group({
      name: ['', [Validators.required]],
      email: ['', [Validators.required, Validators.email]],
    });

    this.userService.loading$
      .pipe(takeUntilDestroyed(this.destroyRef))
      .subscribe((loading) => {
        if (!loading) {
          this.isEdit = false;
          this.toggleForm();
        }
      });
  }
}
```

pages/profile-detail/profile-detail.component.html

```
<div class="container">
  <div class="header">
    <ion-icon (click)="handleBack()" name="arrow-back-outline"></ion-icon>
    <h2>Особисті деталі</h2>
  </div>

  @if (user) {
    <div class="avatar">
      <app-user-avatar
        [size]="80"
        [userName]="user.name"
        [userSrc]="uploadedSrc || user.avatar"
        [isEditable]="isEdit"
        (handleUploadPhoto)="uploadPhoto($event)"
      ></app-user-avatar>
    </div>
  } @else {
    <ngx-skeleton-loader class="avatar" count="1" [theme]="{'height': '80px',
'width': '80px', 'border-radius': '100%'}"></ngx-skeleton-loader>
  }

  <form [formGroup]="profileForm" class="form-profile">
    <div class="input-container">
      <label for="name">Ім’я</label>
      <input
        [class.input-error]="showErrorName"
        formControlName="name"
        class="field"
        placeholder="Введіть ім’я..."
        id="name"
        type="text"
      />
      @if(showErrorName) {
        @if(profileForm.get('name')?.errors?.required) {
          <span class="error">Це поле обов’язкове</span>
        }
      }
    </div>

    <div class="input-container">
      <label for="email">Електронна адресса</label>
      <input
        [class.input-error]="showErrorEmail"
        formControlName="email"
        class="field"
        placeholder="Введіть пошту..."
        id="email"
        type="text"
      />
      @if(showErrorEmail) {
        @if(profileForm.get('email')?.errors?.required) {
          <span class="error">Це поле обов’язкове</span>
        } @else {
          <span class="error">Неправильна пошта</span>
        }
      }
    </div>
  </form>

  <div class="edit-container">
    <button (click)="handleClickEdit()" class="edit">{{isEdit ? 'Зберегти' :
'Редагувати'}}</button>
  </div>
```

```
</div>

@if (loading$ | async) {
  <app-loader></app-loader>
}
```

Back-End:

schemas/author.schema.ts

```
@Schema({ versionKey: false })
export class Author extends Document {
  @Prop({ required: true })
  name: string;

  @Prop({ required: true })
  age: number;
}

export const AuthorSchema = SchemaFactory.createForClass(Author);
```

schemas/book.schema.ts

```
@Schema({ versionKey: false })
export class Book extends Document {
  @Prop({ required: true })
  title: string;

  @Prop({ required: true })
  description: string;

  @Prop({ required: true })
  language: string;

  @Prop()
  image: string;

  @Prop({ required: true })
  pages: number;

  @Prop({ required: true })
  price: number;

  @Prop({ default: 0 })
  purchaseCount: number;

  @Prop({ default: 0 })
  availableCount: number;

  @Prop({ type: Types.ObjectId, ref: 'Genre', required: true })
  genre: string;

  @Prop({ type: Types.ObjectId, ref: 'Author', required: true })
  author: string;
}
```

schemas/favorite-list.schema.ts

```
@Schema({ versionKey: false })
export class FavoriteList extends Document {
  @Prop({ type: Types.ObjectId, ref: 'User', required: true })
  user: string;

  @Prop({ type: Types.ObjectId, ref: 'Book', required: true })
  book: string;
}

export const FavoriteListSchema = SchemaFactory.createForClass(FavoriteList);
```

schemas/payment.schema.ts

```
@Schema({ versionKey: false, timestamps: true })
export class Payment extends Document {
  @Prop({ required: true })
  totalPrice: number;

  @Prop({ required: true })
  status: string;

  @Prop({ type: Types.ObjectId, ref: 'User', required: true })
  user: string;

  @Prop({ required: true })
  address: string;

  @Prop({ required: true })
  typePayment: string;

  @Prop()
  checkId: number;

  @Prop()
  createdAt: Date;

  @Prop()
  updatedAt: Date;

  @Prop({ required: true })
  phoneNumber: string;

  @Prop({
    type: [
      {
        book: { type: Types.ObjectId, ref: 'Book', required: true },
        count: { type: Number, required: true },
      },
    ],
    required: true,
  })
  books: { book: Types.ObjectId; count: number }[];
}

export const PaymentSchema = SchemaFactory.createForClass(Payment);
```

schemas/review.schema.ts

```
@Schema({ versionKey: false, timestamps: true })
export class Review extends Document {
  @Prop({ type: Types.ObjectId, ref: 'Book', required: true })
```

```
  book: string;

  @Prop({ type: Types.ObjectId, ref: 'User', required: true })
  user: string;

  @Prop({ required: true })
  rate: number;

  @Prop()
  description: string;

  @Prop()
  createdAt: Date;

  @Prop()
  updatedAt: Date;
}

export const ReviewSchema = SchemaFactory.createForClass(Review);
```

schemas/user.schema.ts

```
@Schema({
  versionKey: false,
  toJSON: {
    transform: (_doc, ret) => {
      delete ret.password;
      delete ret.avatarId;
      return ret;
    },
  },
})
export class User extends Document {
  @Prop({ required: true })
  name: string;

  @Prop({ required: true, unique: true })
  email: string;

  @Prop()
  avatar: string;

  @Prop()
  avatarId: string;

  @Prop({ required: true })
  password: string;

  @Prop({ default: 0 })
  bookPoints: number;
}

export const UserSchema = SchemaFactory.createForClass(User);
```

sockets/payment.socket.ts

```
@WebSocketGateway({ cors: { origin: '*' } })
export class StatusGateway {
  @WebSocketServer() server: Server;

  sendStatusToClients(id: string, status: string) {
```

```
    this.server.emit('statusUpdate', { id, status });
  }
}
```

modules/auth/auth.service.ts

```
@Injectable()
export class AuthService {
  constructor(
    @InjectModel(User.name) private userModel: Model<User>,
    @InjectModel(RefreshToken.name)
    private refreshTokenModel: Model<RefreshToken>,
    private readonly jwtService: JwtService,
  ) {}

  async signUp(signupData: SignupDto): Promise<{ message: string }> {
    const { email, password, name } = signupData;
    const emailInUse = await this.userModel.findOne({ email });

    if (emailInUse) {
      throw new BadRequestException('Email already in use');
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    await this.userModel.create({ name, email, password: hashedPassword });

    return { message: 'success' };
  }

  async login(credentials: LoginDto) {
    const { email, password } = credentials;

    const user = await this.userModel.findOne({ email }).exec();
    if (!user) {
      throw new UnauthorizedException('Wrong email or password');
    }

    const passwordMatch = await bcrypt.compare(password, user.password);
    if (!passwordMatch) {
      throw new UnauthorizedException('Wrong email or password');
    }

    const tokens = await this.generateUserTokens(user.id);

    return { ...tokens, userId: user.id };
  }

  async refreshToken(refreshToken: string) {
    const token = await this.refreshTokenModel
      .findOne({
        token: refreshToken,
        expiryDate: { $gte: new Date() },
      })
      .exec();

    if (!token) {
      throw new UnauthorizedException();
    }

    return this.generateUserTokens(token.userId);
  }
```

```
  async generateUserTokens(userId) {
    const accessToken = this.jwtService.sign({ userId });
    const refreshToken = uuidv4();

    await this.storeRefreshToken(refreshToken, userId);

    return { accessToken, refreshToken };
  }

  async storeRefreshToken(token: string, userId: string) {
    const expiryDate = new Date();
    expiryDate.setDate(expiryDate.getDate() + 7);

    await this.refreshTokenModel.updateOne(
      { userId },
      { $set: { expiryDate, token } },
      {
        upsert: true,
      },
    );
  }
}
```

modules/author/author.service.ts

```
@Injectable()
export class AuthorService {
  constructor(@InjectModel(Author.name) private authorModel: Model<Author>)
{}

  async getAllAuthor(): Promise<Author[]> {
    return this.authorModel.find().exec();
  }

  async addAuthor(authorData: AddAuthorDto): Promise<Author> {
    const { name, age } = authorData;
    const isAuthorExist = await this.authorModel.findOne({ name });

    if (isAuthorExist) {
      throw new BadRequestException('Author already exist');
    }

    return await this.authorModel.create({ name, age });
  }
}
```

modules/book/book.service.ts

```
@Injectable()
export class BookService {
  constructor(
    @InjectModel(Book.name) private bookModel: Model<Book>,
    @InjectModel(Author.name) private authorModel: Model<Author>,
    @InjectModel(FavoriteList.name) private favoriteModel:
Model<FavoriteList>,
    private cloudUpload: CloudinaryService,
  ) {}

  async getAllBooks(): Promise<Book[]> {
    return this.bookModel.find().exec();
```

```
  }

  async getTopBooks(): Promise<Book[]> {
    return this.bookModel
      .find()
      .sort({ purchaseCount: -1 })
      .limit(10)
      .populate('author')
      .populate('genre')
      .exec();
  }

  async addBook(
    bookData: AddBookDto,
    file: Express.Multer.File,
  ): Promise<Book> {
    const isBookExist = await this.bookModel.findOne({ title: bookData.title
});

    if (isBookExist) {
      throw new BadRequestException('Book already exist');
    }

    try {
      const image = await this.cloudUpload.uploadImage(file, ImageEnum.Book);
      return this.bookModel.create({ ...bookData, image: image.secure_url });
    } catch (error) {
      throw new BadRequestException('Invalid file type.');
    }
  }

  async detailBook(id: string, userId: string) {
    const book = await this.bookModel
      .findById(id)
      .populate('author')
      .populate('genre')
      .exec();

    if (!book) {
      throw new NotFoundException('Book not found');
    }

    const isFavorite = await this.favoriteModel.exists({
      user: userId,
      book: id,
    });

    return { ...book.toObject(), isFavorite: !!isFavorite };
  }

  async genreBooks(query: string): Promise<Book[]> {
    if (!query) {
      return [];
    }

    return this.bookModel.find({ genre: query }).exec();
  }

  async searchBooks(query: string): Promise<Book[]> {
    if (!query) {
      return [];
    }
```

```
    const authorsByName = await this.authorModel
      .find({
        name: { $regex: query, $options: 'i' },
      })
      .distinct('_id')
      .exec();

    const authorIds = authorsByName.map((i: any) => i.toString());

    const searchQuery = {
      $or: [
        { title: { $regex: query, $options: 'i' } },
        {
          author: {
            $in: authorIds,
          },
        },
      ],
    };

    return await this.bookModel.find(searchQuery).populate('author').exec();
  }
}
```

modules/favorites/favorites.service.ts

```
@Injectable()
export class FavoriteService {
  constructor(
    @InjectModel(FavoriteList.name) private favoriteModel:
Model<FavoriteList>,
  ) {}

  public async getFavoriteByUser(userId: string) {
    const favorites = await this.favoriteModel
      .find({ user: userId })
      .populate('book')
      .exec();

    return favorites.map((fav) => fav.book);
  }

  public async addFavorite(
    bookId: string,
    userId: string,
  ): Promise<FavoriteList> {
    const existingFavorite = await this.favoriteModel.findOne({
      book: bookId,
      user: userId,
    });

    if (existingFavorite) {
      throw new BadRequestException('Favorite already exists');
    }

    return await this.favoriteModel.create({ book: bookId, user: userId });
  }

  public async removeFavorite(
    bookId: string,
    userId: string,
  ): Promise<{ message: string }> {
```

```
    const existingFavorite = await this.favoriteModel.findOne({
      book: bookId,
      user: userId,
    });

    if (!existingFavorite) {
      throw new BadRequestException('Favorite not found');
    }

    await this.favoriteModel.deleteOne({ book: bookId, user: userId });
    return { message: 'Favorite removed successfully' };
  }
}
```

modules/payments/payments.service.ts

```
@Injectable()
export class PaymentsService {
  private readonly merchantId: string;
  private readonly secretKey: string;

  constructor(
    @InjectModel(User.name) private userModel: Model<User>,
    @InjectModel(Book.name) private bookModel: Model<Book>,
    @InjectModel(Payment.name) private paymentModel: Model<Payment>,
    private configService: ConfigService,
    private statusGateway: StatusGateway,
  ) {
    this.merchantId = this.configService.get<string>('MERCHANT_ID') || '';
    this.secretKey = this.configService.get<string>('SECRET_KEY_PAYMENT') ||
'';
  }

  public async createPayment(
    payloadPayment: CreatePaymentDto,
    checkId?: number,
  ): Promise<{ message: string; paymentId: unknown }> {
    const user = await this.userModel.findById(payloadPayment.userId).exec();

    if (!user) {
      throw new BadRequestException('User not found');
    }

    if (payloadPayment.bonus > user.bookPoints) {
      throw new BadRequestException('Not enough bonus points');
    }

    const books = await this.bookModel
      .find({
        _id: { $in: payloadPayment.books.map((book) => book._id) },
      })
      .exec();

    if (books.length !== payloadPayment.books.length) {
      throw new BadRequestException('Some books were not found');
    }

    const totalPrice =
      payloadPayment.books.reduce((acc, book) => {
        const foundBook = books.find(
          (b) => (b._id as any).toString() === book._id,
        );
```

```
        return acc + (foundBook ? foundBook.price * book.count : 0);
      }, 0) - payloadPayment.bonus;

    if (payloadPayment.bonus > 0) {
      user.bookPoints -= payloadPayment.bonus;
    }

    const bonusPoints = Math.floor(totalPrice * 0.035);
    user.bookPoints += bonusPoints;
    await user.save();

    await Promise.all(
      books.map(async (book: any) => {
        const purchasedBook = payloadPayment.books.find(
          (b) => b._id === book._id.toString(),
        );

        if (purchasedBook) {
          book.availableCount -= purchasedBook.count;
          book.purchaseCount += purchasedBook.count;
          await book.save();
        }
      }),
    );

    const payment = new this.paymentModel({
      totalPrice,
      status: 'pending',
      user: payloadPayment.userId,
      address: `${payloadPayment.city}, ${payloadPayment.department}`,
      typePayment: payloadPayment.typePayment,
      phoneNumber: payloadPayment.phoneNumber,
      books: payloadPayment.books.map((book) => ({
        book: book._id,
        count: book.count,
      })),
    });

    if (checkId) {
      payment.checkId = checkId;
    }

    await payment.save();
    return { message: 'Payments added successfully', paymentId: payment._id
};
  }

  public async getPaymentUser(userId: string): Promise<Payment[]> {
    const user = await this.userModel.findById(userId).exec();

    if (!user) {
      throw new BadRequestException('User not found');
    }

    return await this.paymentModel
      .find({ user: userId })
      .populate({
        path: 'books.book',
        model: 'Book',
      })
      .exec();
  }
```

```
public async createPayloadIntegration(payloadPayment: CreatePaymentDto) {
  const totalPrice = await this.calculateTotalPrice(payloadPayment);
  const generateDesc = await this.generateDescription(payloadPayment);

  const orderId = Date.now();
  const paymentData = {
    order_id: orderId,
    merchant_id: this.merchantId,
    order_desc: `Найкращі книги BookBy: ${generateDesc}`,
    amount: totalPrice * 100,
    merchant_data: JSON.stringify(payloadPayment),
    currency: 'UAH',
    response_url:
      *',
  };

  const sortedKeys = Object.keys(paymentData).sort((a, b) => {
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;
  });

  const signatureRaw = sortedKeys.map((key) => paymentData[key]).join('|');
  const signature = crypto
    .createHash('sha1')
    .update(`${this.secretKey}|${signatureRaw}`)
    .digest('hex');

  const requestPayload = {
    request: {
      ...paymentData,
      signature,
    },
  };

  const { data } = await axios.post(
    *',
    requestPayload,
  );

  return data.response;
}

public async successPayment(req: Request, res: Response) {
  const body = req.body;
  const data: CreatePaymentDto = JSON.parse(body.merchant_data);

  try {
    console.log(body.response_status);
    if (body.response_status === 'success') {
      this.statusGateway.sendStatusToClients(data.userId, 'success');
      await this.createPayment(data, body.order_id);
      return res.json(true);
    }

    this.statusGateway.sendStatusToClients(data.userId, 'failed');
    return res.json(false);
  } catch (e) {
    this.statusGateway.sendStatusToClients(data.userId, 'failed');
    return res.json(e);
  }
}
```

```
  private async generateDescription(
    payload: CreatePaymentDto,
  ): Promise<string> {
    const books = await this.bookModel
      .find({
        _id: { $in: payload.books.map((book) => book._id) },
      })
      .exec();

    return payload.books.reduce((acc, book) => {
      const foundBook = books.find(
        (b) => (b._id as any).toString() === book._id,
      );

      if (!acc) {
        return `${foundBook?.title} (${book.count} шт.)`;
      }

      return acc + `, ${foundBook?.title} (${book.count} шт.)`;
    }, '');
  }

  private async calculateTotalPrice(
    payload: CreatePaymentDto,
  ): Promise<number> {
    const books = await this.bookModel
      .find({
        _id: { $in: payload.books.map((book) => book._id) },
      })
      .exec();

    return (
      payload.books.reduce((acc, book) => {
        const foundBook = books.find(
          (b) => (b._id as any).toString() === book._id,
        );
        return acc + (foundBook ? foundBook.price * book.count : 0);
      }, 0) - payload.bonus
    );
  }
}
```

modules/review/review.service.ts

```
@Injectable()
export class ReviewsServices {
  constructor(@InjectModel(Review.name) private reviewModel: Model<Review>)
{}

  async getReviewsByBook(bookId: string): Promise<Review[]> {
    return this.reviewModel.find({ book: bookId }).populate('user').exec();
  }

  async getReviewsByUser(userId: string): Promise<Review[]> {
    return this.reviewModel
      .find({ user: userId })
      .populate('user')
      .populate('book')
      .exec();
  }

  async addReviewToBook(payload: ReviewAddDto): Promise<Review> {
```

```
    const { bookId, userId, rate, description } = payload;

    const isExistReview = await this.reviewModel
      .findOne({ book: bookId, user: userId })
      .exec();

    if (isExistReview) {
      throw new BadRequestException(
        'Відгук вже залишений від вашого користувача на цій книгі',
      );
    }

    return await this.reviewModel.create({
      book: bookId,
      user: userId,
      rate,
      description,
    });
  }

  async updateReview(payload: ReviewEditDto): Promise<Review> {
    const review = await this.reviewModel.findById(payload.reviewId).exec();

    if (!review) {
      throw new BadRequestException('Відгук не знайдено');
    }

    if (review.user.toString() !== payload.userId) {
      throw new BadRequestException('Ви не маєте права редагувати цей
відгук');
    }

    review.rate = payload.rate;
    review.description = payload.description;
    await review.save();

    return review;
  }

  async deleteReview(reviewId: string, userId: string): Promise<void> {
    const review = await this.reviewModel.findById(reviewId).exec();

    if (!review) {
      throw new BadRequestException('Відгук не знайдено');
    }

    if (review.user.toString() !== userId) {
      throw new BadRequestException('Ви не маєте права видаляти цей відгук');
    }

    await this.reviewModel.findByIdAndDelete(reviewId).exec();
  }
}
```

modules/users/users.service.ts

```
@Injectable()
export class UsersService {
  constructor(
    @InjectModel(User.name) private userModel: Model<User>,
    private cloudUpload: CloudinaryService,
  ) {}
```

```
  async getUser(id: string): Promise<User> {
    const user = await this.userModel.findById(id).exec();

    if (!user) {
      throw new NotFoundException(`User with ID ${id} not found`);
    }

    return user;
  }

  async updateUser(payload: UpdateUserDto): Promise<User> {
    const { userId, name, email } = payload;
    const user = await this.userModel.findById(userId).exec();

    if (!user) {
      throw new NotFoundException(`User with ID ${userId} not found`);
    }

    const updatedUser = await this.userModel
      .findByIdAndUpdate(userId, { name, email }, { new: true })
      .exec();

    if (!updatedUser) {
      throw new NotFoundException(`User with ID ${userId} not found`);
    }

    return updatedUser;
  }

  async uploadPhoto(userId: string, file: Express.Multer.File): Promise<void>
{
    const user = await this.userModel.findById(userId).exec();

    if (!user) {
      throw new NotFoundException(`User with ID ${userId} not found`);
    }

    try {
      if (user.avatar && user.avatarId) {
        await this.cloudUpload.removeImage(user.avatarId);
      }

      const image = await this.cloudUpload.uploadImage(file, ImageEnum.User);
      await this.userModel.findByIdAndUpdate(
        userId,
        {
          avatar: image.secure_url,
          avatarId: image.public_id,
        },
        { new: true },
      );
    } catch (error) {
      throw new Error(`Failed to upload avatar: ${error.message}`);
    }
  }
}
```

ДОДАТОК Г (обов’язковий). Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА
РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ «КНИЖКОВИЙ МАГАЗИН ТА БІБЛІОТЕКА»

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:

«Розробка мобільного застосунку Книжковий магазин та книгарня »

Автор: ст. групи 3ПІ-21б Яворський Б.М.
Науковий керівник к.т.н., доц. каф. ПЗМайданюк В.П.

Вінниця ВНТУ - 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

Мобільні технології стали невіддільною частиною повсякденного життя сучасної людини. Протягом останніх років електронна комерція швидко розвивається у всьому світі, забезпечуючи при цьому безпрецедентні засоби і можливості для розвитку світової торгівлі. Одним із основних напрямів цього розвитку є перехід електронної комерції у мобільний формат, і книжкові онлайн-магазини активно долучаються до цієї тенденції. Тому розробка мобільного застосунку книжкового магазину стає актуальною та перспективною задачею.

Однією з головних проблем електронної комерції книг є забезпечення того, щоб користувачі з мобільними пристроями отримували швидкий і легкий доступ до широкого асортименту літератури. Крупні бренди створюють власні маркетплейси що створює конкуренцію з іншими eCommerce проектами та забирає в них частину ринку. Програмна система мобільного книжкового магазину, яка забезпечує зручний та інтуїтивно зрозумілий інтерфейс, відкриває нові горизонти у покращенні якості послуг, спрямованих на обслуговування клієнтів та максимізацію продажів. Окрім того, мобільні технології відкривають нові горизонти для персоналізації користувацького досвіду.

Актуальність розробки мобільної системи для книжкового магазину зумовлена зростаючим попитом на зручні цифрові сервіси для придбання та читання книг. Сучасний користувач зазвичай розраховує на швидкий доступ до широкого асортименту літератури, індивідуальні рекомендації та зручні інструменти для оформлення замовлення. З огляду на велику популярність мобільних пристроїв, інтеграція книжкових магазинів із мобільними платформами є логічним і необхідним кроком. Таким чином, розробка мобільного додатку для книжкового магазину є актуальною.

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

Мета та завдання дослідження. Метою роботи є поєднання можливостей електронної комерції та цифрового читання в межах одного мобільного застосунку, що автоматизують та спрощують процеси вибору, оформлення та читання книг, які дозволять зменшити час на пошук необхідних видань та підвищити точність рекомендацій та забезпечити зручний доступ до електронного контенту.

Об'єкт дослідження – процес розробки мобільного застосунку для книжкових магазинів та бібліотек.

Предмет дослідження – методи та засоби розробки мобільних застосунків для книжкових магазинів та бібліотек.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Завдання дослідження

- провести аналіз аналогів і сформувати вимоги до мобільного застосунку;
- розробити швидкий алгоритм пошуку книг за назвою, автором, жанром та іншими критеріями;
- розробити функціонал авторизації та реєстрації користувачів з валідацією даних;
- розробити модуль формування кошика, який буде спроможний зберігати вибрані книги, відображати їх кількість та вартість;
- розробити модуль оформлення замовлення, який буде забезпечувати зручне введення даних для доставки та оплати;
- розробити модуль для можливості зберігання та читання електронних книг;
- провести тестування розробленого мобільного застосунку.

Рисунок Г.4 – Завдання дослідження

Новизна отриманих результатів

1. Подальшого розвитку отримала система для книжкової торгівлі, у яку на відміну від існуючих, інтегровано модуль читання електронних книг з підтримкою базових функцій перегляду (гортання сторінок, масштабування), що виключає потребу в зовнішніх рідерах.
2. Подальшого розвитку отримала система обробки платежів, у якій на відміну існуючих, забезпечена підтримка інтеграції банківських карток, Apple Pay та Google Pay безпосередньо у мобільному застосунку, що підвищує зручність та безпеку здійснення оплат в процесі оформлення замовлення та не потребує переходу до сторонніх платіжних систем.

Рисунок Г.5 – Новизна отриманих результатів

Практична цінність отриманих результатів

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для автоматизації процесів купівлі та продажу книжкової продукції через мобільні пристрої.

Рисунок Г.6 – Практична цінність отриманих результатів

Аналіз сучасних систем

У сучасних умовах цифровізації мобільні застосунки відіграють ключову роль у розвитку книжкової торгівлі. Понад 60% онлайн-покупок здійснюється через мобільні пристрої.

- Більшість наявних рішень не забезпечують достатнього рівня зручності, персоналізації та інтеграції з сервісами доставки й оплати.
- Мобільні додатки мають значні переваги: доступність у будь-який час, персоналізовані рекомендації, відгуки, рейтинги та зручний інтерфейс.
- Рекомендується розробити сучасну систему, яка враховує ці потреби, забезпечує безпеку даних і покращує користувацький досвід, що сприятиме зростанню лояльності клієнтів і конкурентоспроможності бізнесу.

Рисунок Г.7 – Аналіз сучасних систем

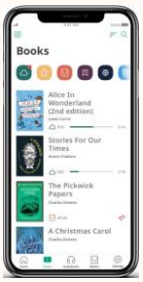
Аналоги мобільного застосунку для книг

До найбільш відомих мобільних застосунків належать:

- Yakaboo;
- Ridmi;
- PocketBook Reader.

Yakaboo

Ridmi



PocketBook Reader.

Рисунок Г.8 – Аналоги мобільного застосунку для книг

Загальний алгоритм роботи системи

Під час запуску перевіряється наявність активного сеансу. Якщо його немає – користувач проходить реєстрацію (ел. пошта, ім'я, пароль). Дані перевіряються на сервері, у разі помилки – виводиться повідомлення.

Після авторизації відкривається головна сторінка з каталогом книг, пошуком і фільтрами. Запити обробляються через API, а результати миттєво оновлюються. Натиснувши на книгу, можна переглянути детальну інформацію або додати її до кошика.

Обмін даними відбувається через HTTP-запити Angular-сервісами. Завдяки Capacitor реалізовано кешування для роботи без стабільного інтернету.

Система має обробку помилок і повідомлення для користувача, клієнтську валідацію форм, а також lazy loading для швидкої та ефективної роботи інтерфейсу.

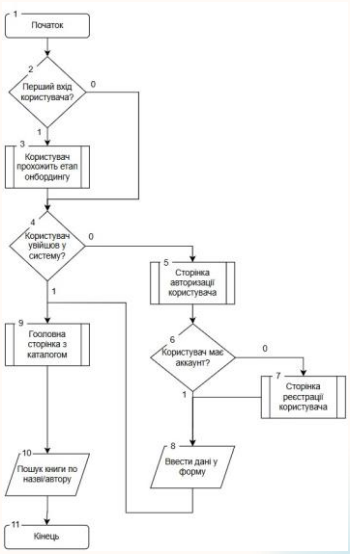


Рисунок Г.9 – Загальний алгоритм роботи системи

Алгоритм створення замовлення

Процес оформлення замовлення розпочинається з додавання вибраних книг до кошика. Користувач має можливість переглянути вміст кошика, змінити кількість товарів або видалити позиції.

Далі йде заповнення форми із контактною інформацією: ім'я, номер телефону, електронна пошта та адреса доставки. Для зручності частина полів може бути автоматично підставлена з профілю користувача.

Після перевірки даних система формує замовлення, зберігає його в базі даних і надсилає повідомлення про успішне створення. Користувач бачить статус замовлення в інтерфейсі (наприклад, "очікує обробки", "передано в доставку", "виконано").

У розділі "Мої замовлення" доступний перегляд усіх попередніх покупок зі статусами та деталями. Це забезпечує прозорість процесу та підвищує довіру користувача до сервісу.

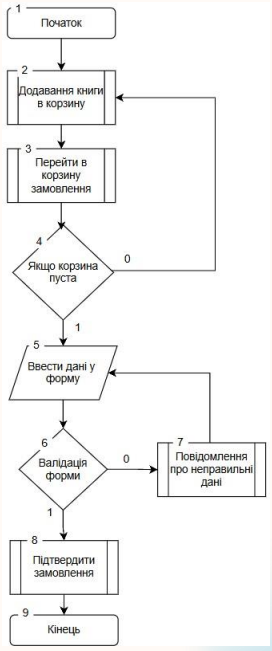


Рисунок Г.10 – Алгоритм створення замовлення

Розробка графічного інтерфейсу

Інтерфейс мобільного застосунку розроблено з урахуванням обмеженого екранного простору та принципів Material Design. Для реалізації використано Angular, Capacitor, Angular Material та Ionic UI.

Передбачено основні екрани: каталог книг, деталі книги, авторизація/реєстрація, особистий кабінет, пошук і фільтри. Всі компоненти створено відповідно до сучасних UX-стандартів.

Інтерфейс взаємодіє з сервером через Angular-сервіси, забезпечуючи динамічне завантаження актуальних даних.

Форми авторизації мають валідацію в реальному часі, що підвищує зручність та зменшує кількість помилок.

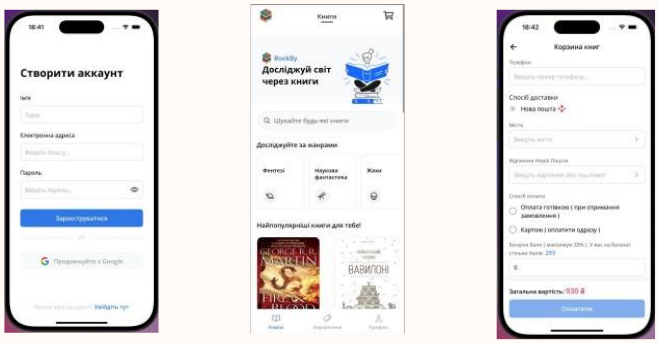


Рисунок Г.11 – Розробка графічного інтерфейсу

Тестування застосунку

Тестування застосунку проводилось для перевірки функціональності, стабільності, продуктивності та сумісності.

Спершу було створено тестове середовище з інстальованою збіркою для Android та iOS. Перевірялися сценарії авторизації, пошуку, додавання товарів у кошик, редагування особистих даних, публікації відгуків та перегляду замовлень. Особливу увагу приділено обробці помилок, реакції на некоректні дані та відсутність результатів.

Було протестовано зміну статусу замовлень у реальному часі, швидкість реакції інтерфейсу (до 1 секунди) та стабільність роботи під навантаженням. Для перевірки API застосовувався Swagger, що дозволило протестувати серверну логіку.

Тестування підтвердило коректну роботу системи, її зручність і надійність при взаємодії з користувачем.

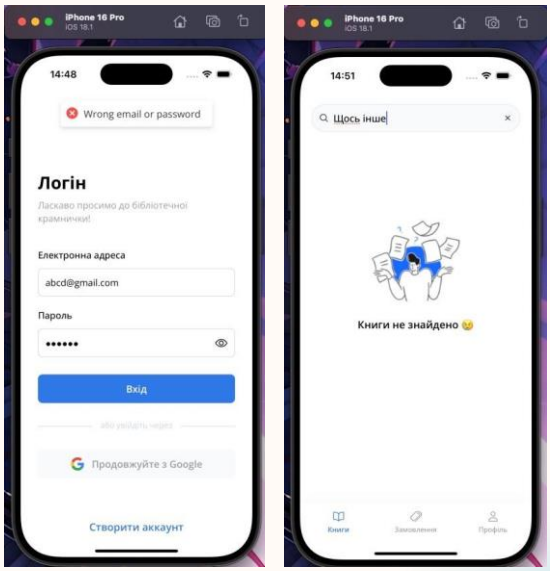


Рисунок Г.12 – Тестування застосунку

Апробація матеріалів бакалаврської кваліфікаційної роботи

Усі результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованій праці, опублікованій у співавторстві, автору належать такі результати: роль мобільних застосунків у популяризації читання: як цифрові технології можуть збільшити інтерес до книг. Результати роботи доповідались на «Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженері» (Вінниця, 2025).

Рисунок Г.13 – Апробація матеріалів

Висновки

У бакалаврській кваліфікаційній роботі було розроблено мобільний застосунок «Книжковий магазин та бібліотека», що забезпечує повноцінну взаємодію користувача з платформою для пошуку, придбання та оцінювання книг. У процесі роботи проведено аналіз сучасного ринку мобільних застосунків у сфері електронної комерції, виявлено недоліки існуючих рішень та обґрунтовано доцільність створення спеціалізованого мобільного інтерфейсу.

Для реалізації застосунку було використано Angular та Capacitor для клієнтської частини, NestJS - для серверної логіки, а MongoDB - для збереження даних. Реалізовано основні модулі: реєстрацію та авторизацію, каталог з пошуком і фільтрацією, управління кошиком, особистий кабінет, а також модуль читання книг. Окрему увагу приділено адаптивності інтерфейсу, зручності користування та відповідності UX-стандартам.

Проведене тестування підтвердило функціональну повноту та стабільність роботи системи на Android і iOS-пристроях. Додаток успішно пройшов перевірку в умовах типових та виняткових ситуацій, показавши відсутність критичних помилок.

Рисунок Г.14 – Висновки