ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Розроблення вебдодатку для книжкового інтернет магазину з використання фреймворку Angular»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_2022
Небрат Мирослав Олександрович
Керівник: Панасенко Наталія
Леонідівна
Рецензент: Муравльов Володимир
Вячеславович

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Небрату Мирославу Олександровичу

1. Тема роботи:
«Розроблення вебдодатку для книжкового інтернет магазину з використання фреймворку Angular»,
керівник роботи к.е.н., доцент, доцент кафедри інформаційних систем та технологій Панасенко Наталія Леонідівна.
Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 383 ст.

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року.

3. Вихідні дані до роботи: стандарти проєктування та розробки вебдодатків, дані офіційних сайтів компаній https://angular.dev/ (для Angular), https://nestjs.com/ (для NestJS), https://www.mysql.com/ (для MySQL), https://typeorm.io/ (для TypeORM), https://www.docker.com/ (для Docker Compose), середовища прикладних програм Adminer, Visual Studio Code, PuTTY, VirtualBox.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

Розділ 1. Аналіз предметної області та вибір технологій розроблення вебдодатку книжкового інтернет-магазину
Розділ 2. Проєктування вебдодатку книжкового інтернет-магазину
Розділ 3. Реалізація та тестування вебдодатку книжкового інтернет-магазину

5. Перелік графічного матеріалу: схеми, рисунки, графіки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Дивнич О. Д., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Оцінювання економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р. - 18.06.2026 р.	

Здобувач вищої освіти **Мирослав НЕБРАТ**

Керівник роботи **Наталія ПАНАСЕНКО**

ЗМІСТ

ВСТУП

Актуальність. Електронна комерція є одним із найбільш динамічних секторів цифрової економіки України. За даними аналітичних досліджень [1, 2], у 2023 році обсяг українського ринку e-commerce склав 151 млрд грн, що на 17% більше, ніж у попередньому році, а вже навесні 2023 року онлайн-торгівля повернулася на довоєнний рівень і продовжила позитивну динаміку. Серед усіх товарних категорій особливе місце посідає книжкова торгівля: український книжковий ринок переживає фазу структурного зростання, де ключовими драйверами є дерусифікація попиту, підсилення україномовної пропозиції та розвиток онлайн-каналів збуту. Лідери сегменту – Yakaboo та «Книгарня "Є"» – демонструють щорічне зростання онлайн-продажів, що свідчить про стійкий попит на цифрові платформи для купівлі книг [3].

Незважаючи на зростання популярності книжкових інтернет-магазинів, багато існуючих рішень мають обмеження: недостатня гнучкість фільтрації каталогу, повільне оновлення інтерфейсу при навігації, відсутність сучасного адаптивного дизайну або надмірна складність технологічного стеку, що ускладнює підтримку і розвиток системи. Це створює потребу у розробці вебдодатків, які поєднують зручність для кінцевого користувача з технологічною сучасністю: односторінкова архітектура (SPA), компонентний підхід до побудови інтерфейсу, типобезпечний серверний API та контейнеризоване середовище розгортання.

Метою кваліфікаційної роботи є розроблення вебдодатку книжкового інтернет-магазину з використанням фреймворку Angular для клієнтської частини та NestJS REST API з базою даних MySQL для серверної частини, що забезпечує автоматизацію процесів онлайн-продажу книг, зручність для користувачів і економічну ефективність впровадження.

Відповідно до мети роботи необхідно вирішити наступні *завдання:*

1. Провести аналіз предметної галузі книжкової електронної комерції, огляд існуючих рішень та архітектурних підходів до побудови вебдодатків, обґрунтувати вибір технологічного стеку;

2. Спроєктувати архітектуру реляційної бази даних, специфікацію REST API та компонентну структуру клієнтської частини відповідно до сформульованих функціональних і нефункціональних вимог;

3. Реалізувати вебдодаток, що включає серверну частину на NestJS у середовищі Docker Compose, клієнтську SPA-частину на Angular та провести техніко-економічне обґрунтування ефективності розробленої системи.

Об'єкт дослідження – процеси проєктування та розробки вебдодатків для електронної комерції з клієнт-серверною архітектурою на основі сучасних JavaScript-фреймворків.

Предмет дослідження – проєктування та розробка вебдодатку книжкового інтернет-магазину з використанням фреймворку Angular, серверного фреймворку NestJS, реляційної бази даних MySQL та середовища контейнеризації Docker Compose.

Методи досліджень – дослідження базуються на методах програмної інженерії, аналізу вимог, проєктування реляційних баз даних (ER-моделювання, нормалізація до третьої нормальної форми), розробки вебдодатків із використанням компонентної SPA-архітектури та REST API, функціонального тестування, а також техніко-економічного аналізу на основі показників ROI та NPV.

Інформаційна база – інтернет-ресурси, офіційна документація використаних технологій (Angular, NestJS, TypeORM, Docker), наукові статті, стандарти розробки програмного забезпечення (ДСТУ 3918-99 / ISO/IEC 12207), а також статистичні дані про ринок електронної комерції та книжкову торгівлю в Україні.

Практична значущість – розроблений вебдодаток забезпечує покупцям зручний інтерфейс для перегляду каталогу книг із фільтрацією за жанром і текстовим пошуком, управління кошиком та оформлення замовлення. Наскрізна типізація на TypeScript, модульна архітектура NestJS і контейнеризація засобами Docker Compose спрощують підтримку та масштабування системи, а відкритість усіх використаних технологій унеможливлює ліцензійні витрати, що робить

рішення привабливим для малого та середнього бізнесу у сфері книжкової торгівлі.

Результати роботи апробовані в рамках наукової конференції здобувачів вищої освіти за результатами науково-дослідної роботи у 2025-2026 рр.

Структура кваліфікаційної роботи логічно пов’язана з задачами досліджень і містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел. Загальний обсяг текстової частини кваліфікаційної роботи складає 54 сторінки формату А4. Вона містить 13 рисунків і 12 таблиць.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Характеристика предметної області та огляд існуючих рішень книжкових інтернет-магазинів в Україні

Електронна комерція є одним із найбільш динамічних секторів цифрової економіки України. У 2023 році обсяг українського ринку електронної комерції склав 151 млрд грн, що на 17% більше, ніж у 2022 році. Незважаючи на воєнні виклики, вже навесні 2023 року обсяг електронної комерції в Україні повернувся на довоєнний рівень і продовжив позитивну динаміку. Серед усіх товарних категорій особливе місце посідає книжкова торгівля, яка демонструє власну траєкторію зростання в умовах підвищеного суспільного попиту на україномовний контент.

Український книжковий ринок у 2023-2025 роках переходить у фазу структурного зростання, де ключовими драйверами є дерусифікація попиту, підсилення україномовної пропозиції, державні програми стимулювання читання та розвиток мереж і онлайн-каналів. Онлайн-торгівля книжками також демонструє щорічне зростання, а лідером у цьому сегменті поряд із «Книгарнею "Є"» є Yakaboo. Таким чином, розроблення вебдодатків для книжкових інтернет-магазинів є актуальним напрямком як з практичної, так і з освітньої точки зору.

Аналіз чинного ринку книжкових вебдодатків в Україні дозволяє виділити кілька ключових гравців. Yakaboo вирізняється найбільшою жанровою різноманітністю – 24 напрямки, що підтверджує його статус універсального маркетплейсу. «Книгарня "Є"» традиційно залишається флагманом у сфері лояльності – 30 активних знижкових і бонусних програм. Bookovka пропонує понад 1 мільйон книг, коміксів, електронних видань, а Book24 – понад 10 тисяч книжкових видань та електронні версії відомої літератури.

Інтернет-магазин є різновидом електронного торговельного майданчика, реалізованого у вигляді вебсервісу. На відміну від традиційної торгівлі, він забезпечує взаємодію між продавцем і покупцем без прямого особистого контакту через засоби цифрової комунікації. У таблиці 1.1 наведено порівняння ключових аспектів традиційної та онлайн-торгівлі книгами.

Таблиця 1.1 – Порівняння традиційної та онлайн-торгівлі книгами

Аспект	Традиційна книгарня	Книжковий інтернет-магазин
Торговий простір	Фізичний зал із полицями	Вебсайт із каталогом
Перегляд товарів	Фізичне ознайомлення	Перегляд сторінок, фільтрація, пошук
Консультація	Продавець-консультант	Онлайн-чат, опис товару, відгуки
Оплата	Готівка або термінал	Картки, мобільний банкінг, Apple/Google Pay
Асортимент	Обмежений площею залу	Практично необмежений каталог
Години роботи	Фіксовані	Цілодобово

З точки зору функціональних вимог до вебдодатку інтернет-магазину, усі згадані рішення реалізують типовий мінімальний набір можливостей: перегляд каталогу з фільтрацією, картка товару з детальним описом, кошик покупця та оформлення замовлення. Такий набір функцій і визначає предмет даної роботи – розроблення вебдодатку для книжкового інтернет-магазину, що охоплює перегляд каталогу та функціонал кошика.

Важливим аспектом є вибір архітектурного підходу до побудови вебдодатку. Сучасні вебзастосунки поділяються на одно- та багатосторінкові (SPA та MPA). Галузь електронної комерції виграє від застосування SPA завдяки динамічним можливостям – фільтрації та функціям кошика. У SPA-підході сторінка завантажується лише один раз, а подальша взаємодія відбувається через асинхронні запити до сервера у форматі JSON, без повного перезавантаження браузером. SPA забезпечує безперервний, схожий на мобільний застосунок досвід завдяки динамічному оновленню контенту без повного перезавантаження сторінки, використовуючи клієнтське рендеринг та технології на кшталт Angular і Vue.js.

Для формалізації поняття швидкодії вебзастосунку в контексті порівняння SPA і MPA можна виразити загальний час відгуку системи через таку залежність:

$$T_{\text{відгуку}} = T_{\text{мережі}} + T_{\text{сервера}} + T_{\text{рендерингу}}, \quad (1.1)$$

де $T_{\text{мережі}}$ – час передачі даних,

$T_{\text{сервера}}$ – час обробки запиту на сервері,

$T_{\text{рендерингу}}$ – час побудови інтерфейсу на стороні клієнта.

У SPA-архітектурі складова $T_{\text{рендерингу}}$ при повторних навігаціях суттєво зменшується, оскільки JavaScript-бандл завантажується лише один раз під час першого відкриття сторінки. На рис. 1.1 показано принципову різницю в обміні даними між SPA і MPA.

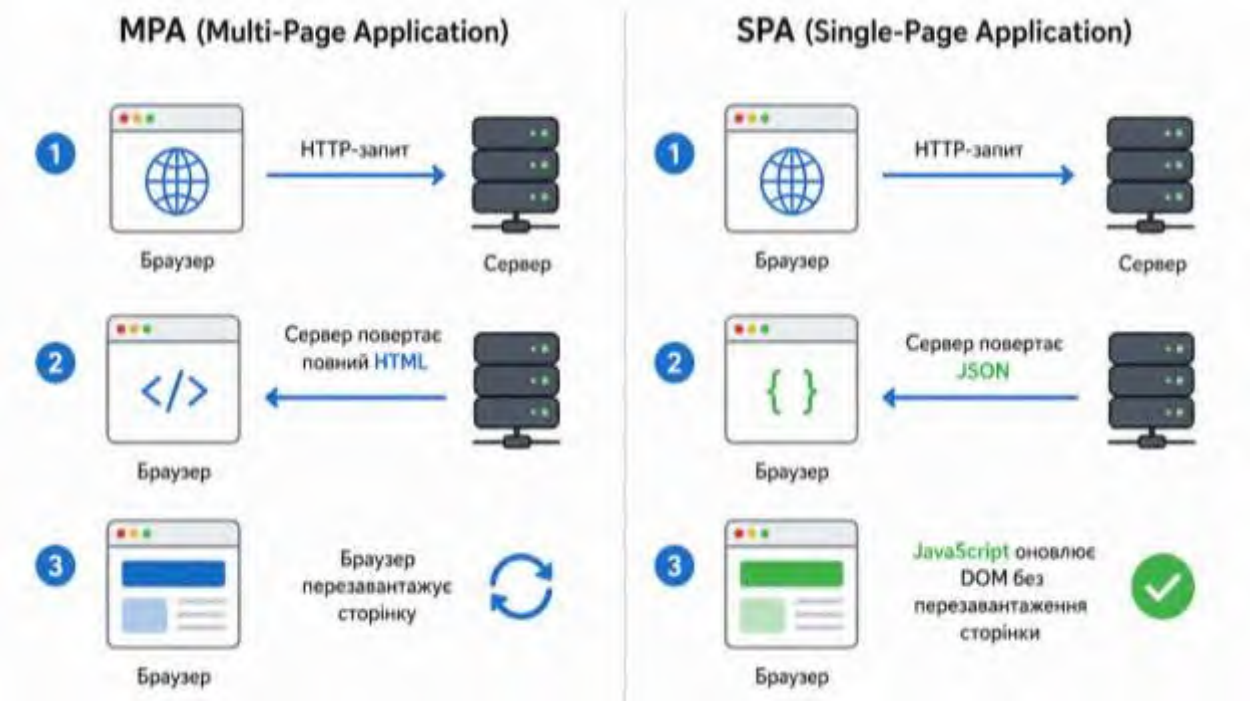

Рисунок 1.1 – Порівняння роботи одно- та багатосторінкового вебдодатку

Обраний у роботі стек – Angular (frontend) + PHP REST API (backend) + MySQL – відповідає типовій для українського ринку навчальних і комерційних проектів архітектурі. Angular є повноцінним фреймворком для побудови SPA з

підтримкою TypeScript, модульністю та вбудованими інструментами для роботи з HTTP-запитами, що робить його доречним вибором для розроблення інтерактивного каталогу з кошиком. PHP разом із OpenServer забезпечує просте розгортання серверної частини в локальному середовищі, а MySQL є стандартом реляційних баз даних у навчальних закладах та малому бізнесі України.

1.2 Аналіз архітектурних підходів до розроблення вебдодатків

Розроблення сучасного вебдодатку інтернет-магазину передбачає вибір відповідної архітектурної моделі як для клієнтської, так і для серверної частини. Два ключові рішення, що визначають структуру системи, це тип застосунку (SPA або MPA) та механізм взаємодії між клієнтом і сервером.

Клієнтська частина вебдодатку відповідає за відображення інтерфейсу користувача й виконується у браузері. У сучасній веброзробці домінує підхід компонентного програмування, за якого інтерфейс поділяється на незалежні, повторно використовувані блоки – компоненти. Кожен компонент інкапсулює власну логіку, шаблон і стилі. Серверна частина, своєю чергою, приймає HTTP-запити від клієнта, виконує бізнес-логіку та повертає відповідь, найчастіше у форматі JSON. Взаємодія між цими рівнями реалізується через REST API.

REST (Representational State Transfer) – це архітектурний стиль, що визначає набір принципів побудови вебсервісів. Основні принципи REST охоплюють: відсутність збереження стану між запитами (stateless), використання стандартних методів HTTP (GET, POST, PUT, DELETE) та адресацію ресурсів через URL. Ключовою перевагою REST API є чітке розмежування між клієнтом і сервером, що дозволяє розробляти та масштабувати їх незалежно.

Для оцінки навантаження на серверну частину застосовується показник пропускної здатності, що виражає кількість успішно оброблених запитів за одиницю часу:

$$T = \frac{N_{\text{успішних}}}{t}, \tag{1.2}$$

де T – пропускна здатність (запитів/с),

$N_{\text{успішних}}$ – кількість успішно оброблених запитів,

t – час спостереження.

Цей показник є одним із критеріїв проєктування серверного API при визначенні технологічного стеку. Серед JavaScript-фреймворків для побудови клієнтської частини SPA найбільшого поширення набули три рішення: React, Angular та Vue. Їхнє порівняння наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння популярних JavaScript-фреймворків для розроблення SPA

Критерій	React	Angular	Vue
Тип	Бібліотека	Повноцінний фреймворк	Прогресивний фреймворк
Мова	JavaScript / TypeScript	TypeScript (обов'язково)	JavaScript / TypeScript
Архітектура	Компонентна, без жорстких правил	Компонентна, MVC, DI	Компонентна, опційний Composition API
Вбудовані інструменти	Мінімум (потрібні сторонні бібліотеки)	Маршрутизація, HTTP, форми, тести	Маршрутизація, Vuex/Pinia
Складність навчання	Середня	Висока	Низька
Підходить для	Гнучких SPA різного масштабу	Великих корпоративних проєктів	Малих і середніх проєктів

React підходить для гнучких та інтерактивних додатків, Angular – для великих корпоративних проєктів, а Vue.js – для легких та зручних у використанні вебдодатків. У контексті даної роботи вибір зупинено на Angular, оскільки саме цей фреймворк забезпечує чітку архітектурну дисципліну та містить усі необхідні інструменти «з коробки».

Angular є повноцінним фреймворком, що розробляється компанією Google і використовує TypeScript як основну мову програмування. Механізм

впровадження залежностей (Dependency Injection, DI) не лише спрощує тестування, дозволяючи заміщати реальні залежності на моки, а й покращує модульність, оскільки залежності можуть бути замінені без необхідності зміни бізнес-логіки. Відокремлюючи компоненти від своїх залежностей, Angular спрощує повторне використання коду: компоненти можна легко застосовувати в різних частинах програми або навіть у кількох проєктах, що призводить до створення зручніших у супроводі та масштабованих кодових баз.

Починаючи з версії Angular 17, фреймворк пройшов суттєву модернізацію. Standalone-компоненти стали типовим підходом за замовчуванням, що значно спрощує структуру застосунку – NgModule більше не є обов'язковим для більшості сценаріїв. Сигнали (Signals), впроваджені як експериментальна можливість в Angular 16, у версії 17 отримали статус стабільних і забезпечують новий реактивний примітив для управління станом, простіший і продуктивніший за виявлення змін на основі Zone.js. Новий синтаксис управління потоком у шаблонах (@if, @for) забезпечує значні покращення продуктивності та кращу зручність авторингу шаблонів. На рисунку 1.2 зображено узагальнену архітектуру застосунку, який розробляється в даній роботі.

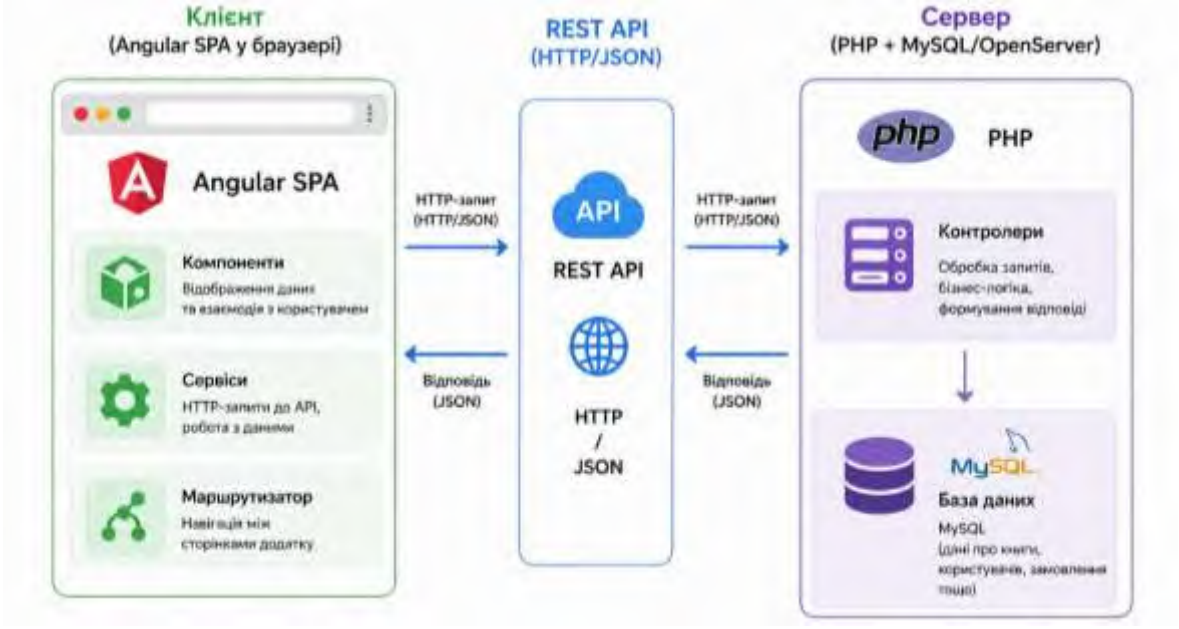

Рисунок 1.2 – Архітектура вебдодатку книжкового інтернет-магазину

Серверна частина реалізується мовою PHP у вигляді мінімального REST API. PHP є однією з найпоширеніших мов серверного програмування у навчальних середовищах та малому бізнесі.

MySQL виступає як система управління реляційними базами даних. Реляційна модель забезпечує цілісність даних через механізм зовнішніх ключів, підтримує транзакції та надає гнучкий механізм запитів через мову SQL. У контексті книжкового інтернет-магазину реляційна СКБД є природним вибором, оскільки доменні сутності – книги, автори, жанри, замовлення – мають чітко визначені зв'язки.

Таким чином, обрана архітектура Angular + NestJS REST API + MySQL представляє повністю розділені рівні клієнта і сервера, що взаємодіють виключно через HTTP-запити з JSON-відповідями. Це відповідає сучасним принципам побудови вебдодатків і водночас забезпечує однорідний технологічний стек: TypeScript використовується як основна мова програмування на обох рівнях системи – як у клієнтській частині на Angular, так і в серверній частині на NestJS.

1.3 Порівняльний аналіз та вибір інструментів і технологій розроблення

Вибір конкретних технологій є практичним завданням, яке безпосередньо випливає з аналізу архітектурних підходів. У цьому підрозділі обґрунтовується добір засобів, що застосовуватимуться в розробленні вебдодатку: клієнтського фреймворку, серверного фреймворку, СКБД, середовища розгортання та UI-бібліотеки.

Angular як платформа для клієнтської частини забезпечує повний інструментальний ланцюжок, необхідний для побудови SPA. Angular CLI дозволяє одною командою створити повністю налаштований проєкт, генерувати

компоненти, сервіси й маршрути, а також зібрати оптимізований бандл для продуктивного середовища [40]. Вибір TypeScript як основної мови є суттєвою перевагою Angular над бібліотеками, де TypeScript є опційним: явна типізація інтерфейсів даних, що передаються між клієнтом і сервером, унеможливлює цілий клас помилок, пов'язаних із невідповідністю структури отриманих даних. У даній роботі використовується Angular версії 19, що підтверджується файлом package.json проєкту.

Для оформлення інтерфейсу вибрано Angular Material – офіційну UI-бібліотеку компонентів, що реалізує специфікацію Material Design 3 компанії Google [41]. Вона надає готові компоненти: картки (MatCard), таблиці (MatTable), форми (MatFormField), кнопки, індикатори завантаження та навігаційні елементи, стилізовані у єдиному дизайн-стилі. Альтернативою є Bootstrap 5, яка пропонує більш нейтральний стиль та гнучку сітку для побудови адаптивних макетів. Порівняння цих двох підходів наведено у таблиці 1.3.

Таблиця 1.3 – Порівняння UI-бібліотек для Angular-застосунків

Критерій	Angular Material	Bootstrap 5 + ng-bootstrap
Стиль	Material Design 3 (Google)	Нейтральний, класичний
Інтеграція з Angular	Нативна, офіційна	Через ng-bootstrap або ngx-bootstrap
Готових компонентів	~35	~40+
Адаптивна сітка	CSS Grid / Flexbox	Вбудована сітка (12 колонок)
Підходить для	Корпоративних SPA	Будь-яких проєктів

У даній роботі обрано Angular Material, оскільки він підтримується тією самою командою Google, що розробляє Angular, гарантує сумісність з актуальними версіями фреймворку та надає компоненти, оптимізовані для роботи зі standalone-компонентами й новим синтаксисом керування потоком.

Серверна частина реалізується на платформі Node.js з використанням фреймворку NestJS. Node.js є середовищем виконання JavaScript поза браузером, побудованим на рушії V8, що забезпечує асинхронну, неблокуючу модель введення-виведення. NestJS – це прогресивний фреймворк для побудови серверних застосунків на Node.js, який використовує TypeScript за

замовчуванням і реалізує архітектурні концепції, безпосередньо інспіровані Angular: модульну систему, декораторний підхід до визначення контролерів і сервісів, а також вбудовану систему впровадження залежностей (Dependency Injection) [42]. Завдяки цьому розробник, знайомий з Angular, може працювати з NestJS без суттєвої зміни ментальної моделі. Порівняння основних серверних фреймворків для Node.js наведено у таблиці 1.4.

Таблиця 1.4 – Порівняння серверних фреймворків для Node.js

Критерій	Express	Fastify	NestJS
Рівень абстракції	Мінімальний	Мінімальний	Високий
TypeScript	Опційний	Опційний	Обов'язковий
Архітектура	Без структури	Без структури	Модульна, DI
Продуктивність	Висока	Дуже висока	Висока (на базі Express або Fastify)
Складність навчання	Низька	Низька	Середня
Підходить для	Мікросервісів, прототипів	Високонавантажених API	Структурованих корпоративних застосунків

На рисунку 1.3 наведено технологічний стек вебдодатку, що розробляється.

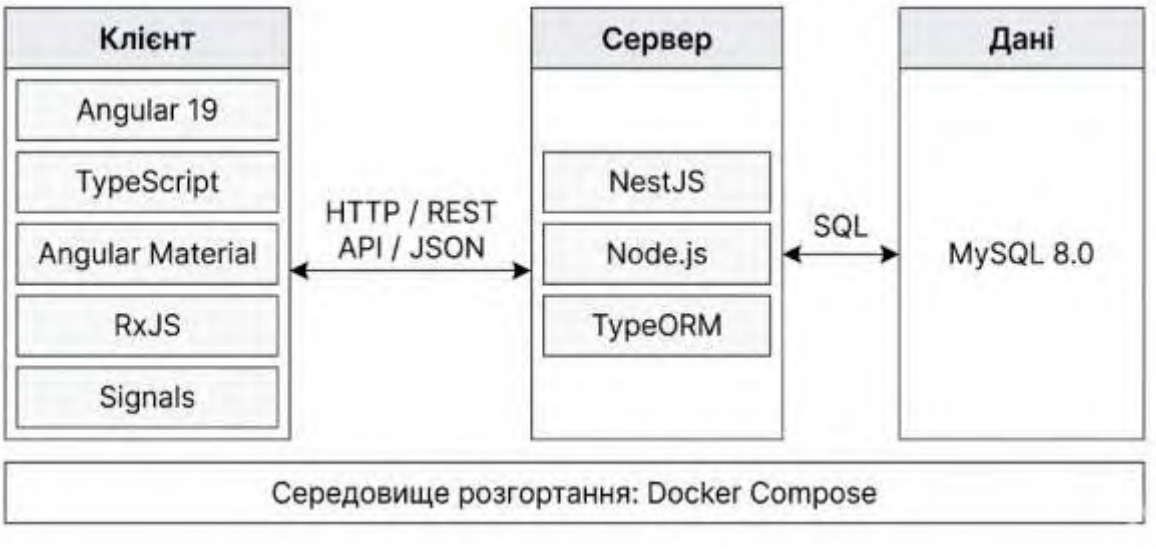

Рисунок 1.3 – Технологічний стек вебдодатку книжкового інтернет-магазину

Ключовою перевагою NestJS у контексті даної роботи є наскрізна типізація: оскільки і клієнтська (Angular), і серверна (NestJS) частини написані на TypeScript, інтерфейси моделей даних – такі як Book, Genre, Order – можуть бути описані один раз і використовуватись для перевірки типів на обох рівнях системи. Це суттєво зменшує кількість помилок, пов'язаних із невідповідністю форматів даних між рівнями.

Як СКБД обрано MySQL версії 8.0. Переваги MySQL – висока швидкість читання, стабільність, зріла екосистема, підтримка зовнішніх ключів та транзакцій. У зв'язці з NestJS вона використовується через бібліотеку TypeORM – об'єктно-реляційний відображувач (ORM), що дозволяє описувати таблиці бази даних у вигляді TypeScript-класів із декораторами (@Entity, @Column, @ManyToOne) та виконувати запити без написання сирого SQL [43]. Для адміністрування бази даних у середовищі розгортання використовується Adminer – легкий вебінтерфейс, що розгортається як окремий Docker-контейнер і доступний через браузер на порті 8080.

Середовищем розгортання є Docker Compose, що запускається всередині віртуальної машини з Alpine Linux. Docker Compose дозволяє описати всі сервіси системи – базу даних, серверну частину, клієнтську частину та Adminer – в одному файлі docker-compose.yml і запустити їх однією командою. Кожен сервіс ізолюється у власному контейнері з чітко визначеними залежностями та мережевими правилами. Дані MySQL зберігаються у Docker-волюмі, що забезпечує їхнє збереження між перезапусками контейнерів. Використання Alpine Linux як базової ОС для віртуальної машини обумовлено її мінімальним розміром (~130 МБ) та низькими вимогами до ресурсів, що є критичним при розгортанні на обмеженому залізі.

Основним середовищем розробки обрано Visual Studio Code – безкоштовний редактор коду від Microsoft з підтримкою TypeScript, підсвічуванням синтаксису Angular-шаблонів, інтегрованим терміналом та розширенням Angular Language Service. Для роботи з кодом у середовищі Alpine Linux використовується з'єднання через SSH-клієнт PuTTY та файловий

менеджер Midnight Commander (mc), що забезпечує зручну навігацію файловою системою без графічного інтерфейсу.

Ключовим показником для оцінки коректності REST API є відсоток успішних HTTP-відповідей від загальної кількості запитів:

$$R_{\text{успіх}} = \frac{N_{2xx}}{N_{\text{всього}}} \times 100\%, \quad (1.3)$$

де N_2xx – кількість відповідей зі статусом 2xx,

$N_{\text{всього}}$ – загальна кількість запитів за період тестування.

Цей показник застосовується при функціональному тестуванні системи. Усі обрані технології є відкритим програмним забезпеченням з активними спільнотами та регулярними оновленнями. Поєднання Angular + TypeScript + Angular Material (клієнт), NestJS + TypeORM (сервер), MySQL (СКБД) та Docker Compose (середовище розгортання) утворює логічно цілісний, наскрізно типізований стек, достатній для реалізації вебдодатку з каталогом книг і кошиком покупця.

1.4 Постановка задачі та формування вимог до системи

Чітке формулювання задачі та систематизація вимог є необхідним підготовчим кроком перед переходом до проєктування системи. Постановка задачі визначає межі розробки, цільову аудиторію та сукупність функцій, яку має реалізувати кінцевий продукт. У сучасній інженерії програмного забезпечення вимоги прийнято поділяти на функціональні, що визначають, що саме має робити система, і нефункціональні, які встановлюють стандарти якості її роботи [15]. Таке розмежування дозволяє однозначно визначити обсяг розробки та сформувати критерії приймання готового продукту.

Метою даної роботи є створення вебдодатку книжкового інтернет-магазину з використанням фреймворку Angular для клієнтської частини та NestJS REST API з базою даних MySQL для серверної частини, розгорнутих у середовищі Docker Compose. Вебдодаток має надавати покупцям зручний інтерфейс для перегляду каталогу книг із фільтрацією за жанром і текстовим пошуком, а також функціонал кошика покупця з можливістю формування замовлення. Об'єктом автоматизації є процес онлайн-продажу книг. Предметом розробки є клієнтська SPA-частина на Angular і серверний REST API на NestJS, що забезпечують взаємодію користувача із каталогом та кошиком.

Кінцевий користувач системи – покупець, що відвідує книжковий вебсайт, переглядає каталог, відбирає книги та формує замовлення. Додаткового рольового розмежування, зокрема адміністраторського інтерфейсу для керування каталогом, у межах цієї роботи не передбачено – дані каталогу вносяться безпосередньо до бази даних через SQL-скрипт ініціалізації при першому запуску контейнера.

Функціональні вимоги визначають конкретні дії, які система має виконувати з точки зору користувача [16]. Вони формулюються у вигляді атомарних тверджень, кожне з яких описує одну спостережувану поведінку системи.

Загальна сума замовлення в кошику обчислюється реактивно при кожній зміні його складу. Математично це виражається як:

$$S = \sum_{i=1}^{n} p_i \cdot q_i, \tag{1.4}$$

де S – загальна сума замовлення (грн),

p_i – ціна i-ї книги (грн),

q_i – кількість примірників i-ї книги в кошику,

n – загальна кількість різних позицій у кошику.

У реалізованому CartService ця формула відповідає обчислюваному сигналу total, що автоматично перераховується при кожній зміні масиву items. На основі аналізу предметної області та цілей розробки для даного вебдодатку сформульовано функціональні вимоги, перелік яких наведено у таблиці 1.5.

Таблиця 1.5 – Функціональні вимоги до вебдодатку книжкового інтернет-магазину

№	Ідентифікатор	Опис вимоги
1	FR-01	Система відображає каталог книг у вигляді карток із назвою, автором, жанром, обкладинкою та ціною
2	FR-02	Користувач може фільтрувати каталог за жанром та виконувати текстовий пошук за назвою або автором
3	FR-03	Система відображає сторінку окремої книги з повним описом, роком видання та можливістю вибору кількості примірників
4	FR-04	Користувач може додати книгу до кошика з будь-якої сторінки – як із каталогу, так і зі сторінки деталей
5	FR-05	Система зберігає вміст кошика між сесіями за допомогою localStorage браузера
6	FR-06	Кошик відображає загальну суму замовлення з автоматичним перерахунком при зміні кількості або складу
7	FR-07	Користувач може змінити кількість примірників, видалити окремий товар або очистити кошик повністю
8	FR-08	Система надсилає дані замовлення на сервер через REST API і підтверджує отримання повідомленням користувачу

Нефункціональні вимоги встановлюють критерії якості системи, що не пов'язані безпосередньо з конкретними функціями, але визначають, наскільки добре система їх виконує [45]. До нефункціональних вимог даного проєкту належать: адаптивність інтерфейсу (коректне відображення на екранах від 320 px до 1920 px), час завантаження початкової сторінки у локальному середовищі не більше 3 с, коректна робота у сучасних браузерах (Chrome, Firefox, Edge), відповідь REST API не пізніше ніж через 500 мс на типовий GET-запит до каталогу, а також збереження даних бази даних між перезапусками контейнерів завдяки використанню Docker-волюму.

Діаграма прецедентів (Use Case) є стандартним засобом UML для формального опису взаємодії акторів із системою [27]. Прецеденти визначають

очікувану поведінку системи та відповідають на питання, що вона робить, але не відповідають на запитання, як саме. На рисунку 1.4 зображено діаграму прецедентів вебдодатку, що розробляється. Прецеденти, визначені на діаграмі, безпосередньо відповідають функціональним вимогам FR-01 – FR-08 з таблиці 1.5. Єдиним актором системи є покупець, що підтверджує відсутність адміністраторської ролі в поточній версії застосунку. Зв'язки типу extend між прецедентами «Переглядати каталог» та «Фільтрувати за жанром» / «Шукати за назвою або автором» відображають необов'язковість цих дій: користувач може переглядати каталог без застосування фільтрів.

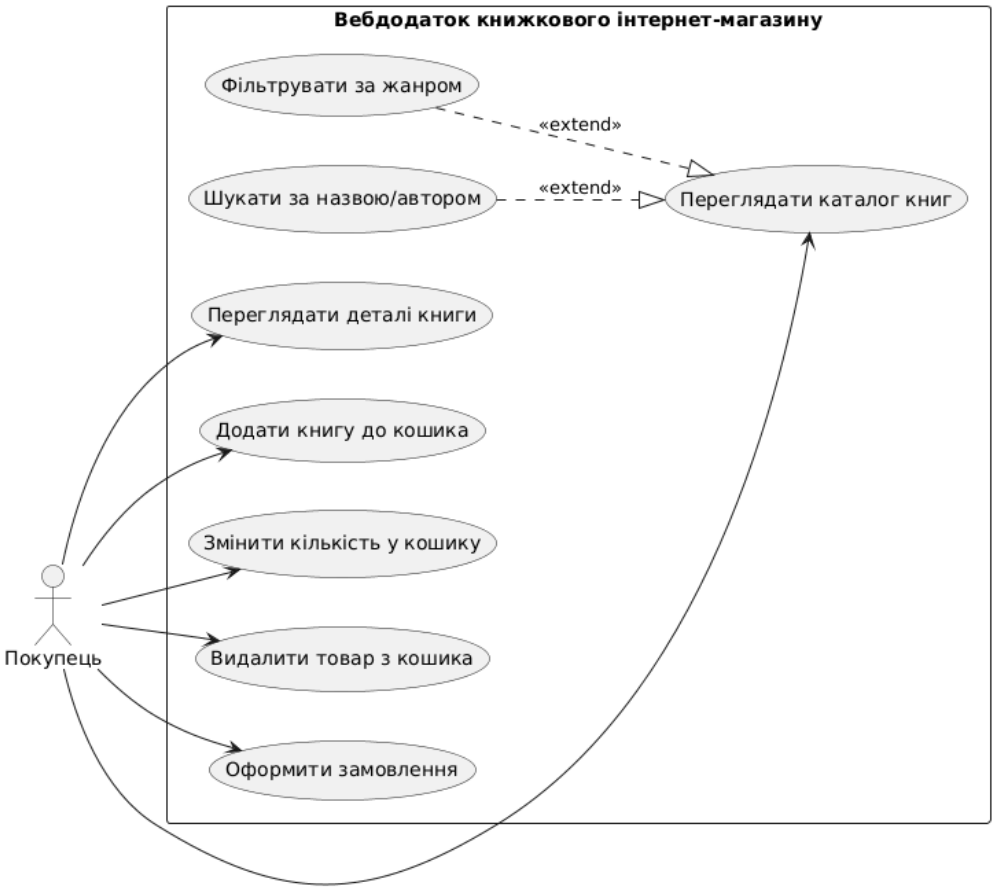

Рисунок 1.4 – Діаграма прецедентів вебдодатку книжкового інтернет-магазину

Сформовані вимоги дозволяють однозначно визначити обсяг робіт і є основою для технічного завдання. Усі функціональні вимоги з таблиці 1.5 підлягають перевірці під час тестування на відповідність критеріям приймання.

Проведено аналіз предметної області книжкової електронної комерції, огляд існуючих рішень та архітектурних підходів до побудови вебдодатків. Встановлено, що SPA-архітектура на базі Angular є обґрунтованим вибором для реалізації інтерактивного каталогу з кошиком покупця. Проведено порівняльний аналіз JavaScript-фреймворків, серверних технологій та UI-бібліотек, за результатами якого обрано стек Angular 19 + TypeScript + Angular Material (клієнт), NestJS + TypeORM (сервер) та MySQL 8.0 у середовищі Docker Compose. Сформульовано вісім функціональних та п'ять нефункціональних вимог до системи, побудовано діаграму прецедентів, що визначає межі розробки та взаємодію актора-покупця із системою, і встановлено кількісні критерії приймання готового продукту.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБДОДАТКУ КНИЖКОВОГО ІНТЕРНЕТ-МАГАЗИНУ

2.1 Розроблення технічного завдання відповідно до ДСТУ 3918-1999

Технічне завдання (ТЗ) є вихідним документом для проєктування програмного продукту. Воно встановлює призначення системи, вимоги до її функціонування, технологічний стек і критерії приймання. У вітчизняній практиці розробки програмного забезпечення нормативну основу для формування ТЗ становить ДСТУ 3918-99, який є гармонізованою версією міжнародного стандарту ISO/IEC 12207:1995 «Інформаційні технології. Процеси життєвого циклу програмного забезпечення» [31]. Цей стандарт встановлює загальну структуру процесів життєвого циклу програмного забезпечення із чітко визначеною термінологією і охоплює процеси, діяльності та завдання, які потрібно застосовувати під час постачання, розробки, експлуатації та супроводу програмних продуктів.

Продукт, що розробляється, має назву «Вебдодаток книжкового інтернет-магазину з використанням фреймворку Angular». Планові терміни виконання визначаються відповідно до графіку написання кваліфікаційної роботи.

Система призначена для надання вебінтерфейсу відвідувачам книжкового онлайн-магазину: перегляду каталогу книг, пошуку та фільтрації видань, а також формування і відправлення замовлення через кошик покупця. Метою є автоматизація процесу онлайн-вибору і замовлення книжкової продукції для кінцевих користувачів без необхідності реєстрації. Об'єктом автоматизації є процес онлайн-торгівлі книгами. Система орієнтована на одну роль – покупця. Дані каталогу зберігаються в реляційній базі даних і надаються клієнтській частині через REST API.

З точки зору загальних технічних вимог, вебдодаток реалізується як одностранична аплікація (SPA) на фреймворку Angular, що взаємодіє із

серверним REST API, реалізованим засобами фреймворку NestJS на платформі Node.js, та базою даних MySQL, розгорнутими у середовищі Docker Compose на віртуальній машині Alpine Linux [32]. Інтерфейс має бути адаптивним, підтримувати сучасні браузери (Chrome, Firefox, Edge) та коректно відображатись на пристроях з шириною екрана від 320 до 1920 пікселів.

Інформаційна модель системи охоплює чотири основні сутності з відповідними атрибутами, що наведені у таблиці 2.1. Між сутностями визначено такі зв'язки: «Автор – Книга» і «Жанр – Книга» – один до багатьох.

Таблиця 2.1 – Інформаційна модель об'єктів системи

Сутність	Атрибути
Книга (book)	id, title (рядок, до 255 символів), description (текст), price (десяткове, 2 знаки), year (ціле), cover_url (рядок), genre_id, author_id
Автор (author)	id, first_name (рядок, до 64 символів), last_name (рядок, до 64 символів)
Жанр (genre)	id, name (рядок, до 64 символів)
Замовлення (order)	id, first_name, last_name, email, phone, delivery_address, items (JSON), created_at

Функціональні вимоги до системи відповідають переліку FR-01 – FR-08, сформульованому у підрозділі 1.4. REST API повинен підтримувати такі ендпоінти: GET /api/books – список книг з підтримкою фільтрів за жанром та текстовим пошуком; GET /api/books/{id} – деталі окремої книги; GET /api/genres – список жанрів; POST /api/orders – збереження замовлення. Відповіді повертаються у форматі JSON.

Щодо програмного забезпечення, клієнтська частина реалізується засобами Angular 17+, TypeScript 5+ та Angular Material. Серверна частина використовує NestJS 10+ на платформі Node.js 20+, для взаємодії з базою даних застосовується TypeORM. СКБД – MySQL 8.0. Середовище розробки – Visual Studio Code з розширенням Angular Language Service. Середовище розгортання – Docker Compose; усі сервіси (база даних, серверна частина, клієнтська частина, Adminer) запускаються як окремі контейнери у віртуальній машині з Alpine Linux.

Результати розробки супроводжуються пояснювальною запискою, що включає аналіз предметної області, проєктну документацію (ER-діаграму, специфікацію API), а також опис реалізації і тестування системи. На рисунку 2.1 зображено зв'язок між ключовими аспектами технічного завдання та відповідними стадіями розробки, визначеними ДСТУ 3918-99.

Рисунок 2.1 – Відповідність аспектів ТЗ стадіям життєвого циклу за ДСТУ 3918-99

Приймання системи здійснюється за результатами функціонального тестування відповідно до вимог FR-01 – FR-08. Система вважається такою, що відповідає ТЗ, якщо відсоток успішно пройдених тест-кейсів задовольняє умову:

$$P = \frac{N_{\text{пройдено}}}{N_{\text{всього}}} \times 100\% \geq 90\%, \tag{2.1}$$

де $N_{\text{пройдено}}$ – кількість тест-кейсів, що пройдені успішно,

$N_{\text{всього}}$ – загальна кількість тест-кейсів програми випробувань.

Поріг у 90% обрано як стандартний критерій готовності для навчальних програмних проєктів [33].

Сформоване технічне завдання є вичерпним документом, що однозначно визначає обсяг, склад і критерії якості програмного продукту, який

розробляється. Воно слугує орієнтиром для всіх наступних етапів – проєктування бази даних, визначення архітектури API та розробки клієнтської частини на Angular.

2.2 Проєктування бази даних та інформаційної структури системи

Проєктування бази даних є одним із ключових етапів розробки будь-якого вебдодатку, що взаємодіє з даними. Цей процес включає концептуальне моделювання предметної області у вигляді ER-діаграми, логічне проєктування реляційних таблиць та нормалізацію схеми. Основи проєктування схеми бази даних передбачають аналіз бізнес-вимог і потреб користувачів для визначення необхідних даних та їх взаємозв'язків, нормалізацію даних для уникнення повторень, встановлення зв'язків між таблицями та оптимізацію продуктивності через вибір відповідних типів даних і створення індексів. На рисунку 2.2 зображено ER-діаграму бази даних вебдодатку книжкового інтернет-магазину.

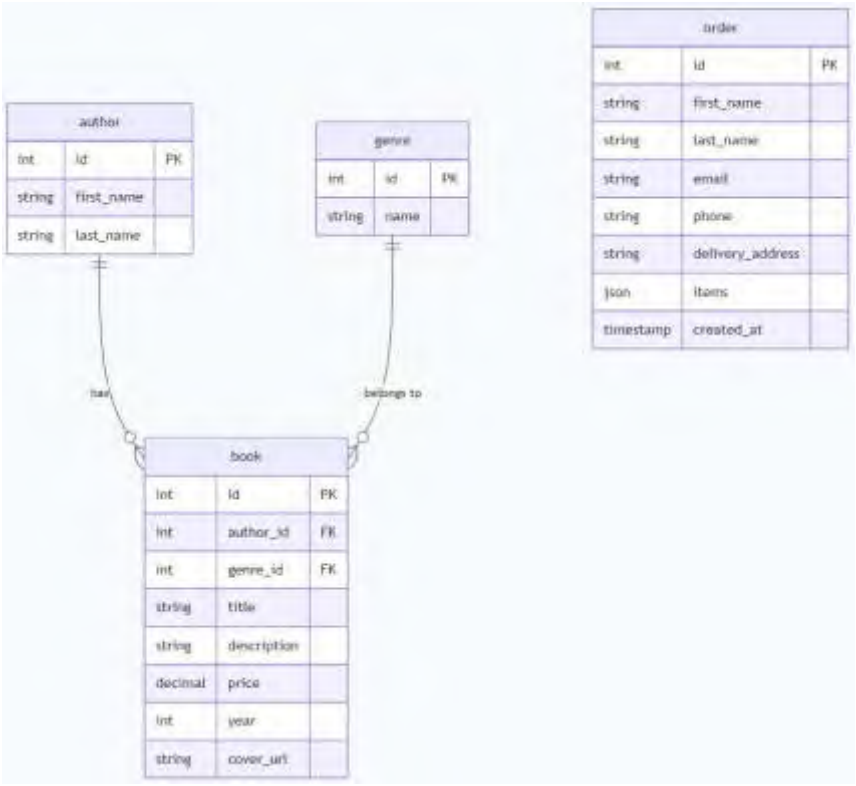

Рисунок 2.2 – ER-діаграма бази даних вебдодатку

Концептуальна модель даних будується на основі сутностей, виявлених при аналізі предметної області та зафіксованих у ТЗ. Для книжкового інтернет-магазину визначено чотири основні сутності: author (автор), genre (жанр), book (книга) та order (замовлення). Між ними встановлено зв'язки типу «один до багатьох»: один автор може написати багато книг, один жанр може містити багато книг. Замовлення зберігає список обраних книг у полі типу JSON, що дозволяє обійтися без окремої проміжної таблиці позицій замовлення в межах поточного масштабу системи.

ER-діаграма (Entity-Relationship diagram) є стандартним графічним засобом для відображення сутностей, їхніх атрибутів і зв'язків між ними. Діаграма «сутність-зв'язок» дозволяє представити базу даних у вигляді візуальних графічних об'єктів, що визначають конкретну предметну область. Детальна специфікація таблиць бази даних наведена у таблиці 2.2.

Таблиця 2.2 – Специфікація таблиць бази даних

Таблиця	Поле	Тип даних	Обмеження	Опис
author	id	INT	PK, AUTO_INCREMENT	Ідентифікатор автора
	first_name	VARCHAR(64)	NOT NULL	Ім'я автора
	last_name	VARCHAR(64)	NOT NULL	Прізвище автора
genre	id	INT	PK, AUTO_INCREMENT	Ідентифікатор жанру
	name	VARCHAR(64)	NOT NULL, UNIQUE	Назва жанру
book	id	INT	PK, AUTO_INCREMENT	Ідентифікатор книги
	title	VARCHAR(255)	NOT NULL	Назва книги
	description	TEXT	–	Опис книги
	price	DECIMAL(10,2)	NOT NULL	Ціна (грн)
	year	INT	–	Рік видання
	cover_url	VARCHAR(500)	–	URL обкладинки
	author_id	INT	FK → author.id	Автор книги
	genre_id	INT	FK → genre.id	Жанр книги
order	id	INT	PK, AUTO_INCREMENT	Ідентифікатор замовлення
	first_name	VARCHAR(64)	NOT NULL	Ім'я покупця
	last_name	VARCHAR(64)	NOT NULL	Прізвище покупця
	email	VARCHAR(255)	NOT NULL	Електронна пошта
	phone	VARCHAR(20)	–	Телефон
	delivery_address	VARCHAR(500)	NOT NULL	Адреса доставки
	items	JSON	NOT NULL	Склад замовлення
	created_at	TIMESTAMP	DEFAULT NOW()	Дата і час замовлення

Переходячи від концептуальної до логічної моделі, кожна сутність відображається у відповідну таблицю реляційної бази даних. Первинний ключ ідентифікує запис у таблиці й не може бути NULL. Зовнішній ключ створює зв'язки між таблицями, дозволяючи одній таблиці посилатися на дані в іншій. У таблиці book поле author_id є зовнішнім ключем, що посилається на author.id, а поле genre_id – зовнішнім ключем, що посилається на genre.id. Це забезпечує посилальну цілісність: неможливо додати книгу з неіснуючим автором або жанром.

Важливим кроком після побудови логічної моделі є перевірка схеми на відповідність нормальним формам. Нормалізація – це техніка проєктування, метою якої є зменшення надмірності та залежності даних шляхом розбиття таблиць на менші пов'язані між собою відношення [35]. Третя нормальна форма (3НФ) вимагає, щоб дані в таблиці залежали виключно від первинного ключа: будь-яке поле, що залежить від первинного ключа і від будь-якого іншого поля, має виноситися в окрему таблицю.

Спроєктована схема відповідає вимогам 3НФ. У таблиці book відсутні транзитивні залежності: ім'я автора та назва жанру не зберігаються безпосередньо в записі книги, а виносяться в окремі таблиці author і genre, до яких звертаються через зовнішні ключі. Завдяки цьому оновлення імені автора або назви жанру виконується в одному місці й автоматично відображається на всіх пов'язаних книгах.

Збереження складу замовлення у полі items типу JSON є свідомим відступом від суворої нормалізації на користь простоти реалізації. Позаяк при збереженні замовлення потрібно зафіксувати не лише ідентифікатор книги, а й її ціну та назву на момент замовлення – що є типовою практикою в e-commerce для захисту від зміни ціни після оформлення, – зберігання цих даних як JSON-знімка є технічно виправданим рішенням [36].

Для оцінки нормалізованості схеми застосовують поняття функціональної залежності. Відношення знаходиться в 3НФ, якщо для будь-якої нетривіальної функціональної залежності $X \rightarrow Y$ виконується умова:

$$X \text{ є суперключем} \quad \vee \quad Y \text{ є атрибутом первинного ключа.}$$

У спроєктованій схемі всі неключові атрибути кожної таблиці функціонально залежать лише від первинного ключа, що підтверджує відповідність третій нормальній формі.

Таким чином, спроєктована база даних має чітку реляційну структуру з визначеними первинними та зовнішніми ключами, відповідає вимогам третьої нормальної форми і є достатньою для реалізації всіх функціональних вимог, визначених у підрозділі 1.4.

2.3 Проєктування REST API серверної частини

REST API є важливим інтерфейсом, що забезпечує обмін даними між клієнтською частиною вебдодатку та серверною базою даних. Проєктування API передбачає визначення набору ендпоїнтів, їхніх методів, параметрів запитів, формату відповідей та кодів стану HTTP. Послідовна архітектура API забезпечує легкість розроблення клієнтської частини, простоту тестування та можливість масштабування системи [24].

Основою REST API є концепція ресурсів, кожен з яких ідентифікується унікальною URL-адресою. У контексті книжкового інтернет-магазину ресурсами є книги, жанри, автори та замовлення. Кожен ресурс може отримуватись, створюватись або оновлюватись за допомогою стандартних HTTP-методів: GET для читання, POST для створення, PUT або PATCH для оновлення, DELETE для видалення [24]. За дизайном REST API вибирається підхід, коли URL відображає структуру даних, а не дії, які з ними виконуються. Наприклад, замість URL-адреси /api/getBooks використовується /api/books з методом GET.

Для вебдодатку книжкового інтернет-магазину спроєктовано мінімальний набір ендпоїнтів, достатній для реалізації функціональних вимог FR-01 – FR-08. У таблиці 2.3 наведено специфікацію REST API з переліком всіх ендпоїнтів, їхніх методів, параметрів та очікуваних відповідей.

Таблиця 2.3 – Специфікація REST API вебдодатку

Метод	Ендпоінт	Параметри запиту	Опис	Статус відповіді
GET	/api/books	genre_id (опціонально), search (опціонально)	Отримання списку книг з можливістю фільтрації за жанром та пошуку за текстом	200
GET	/api/books/{id}	–	Отримання детальної інформації про окрему книгу	200, 404
GET	/api/genres	–	Отримання списку всіх жанрів	200
POST	/api/orders	JSON: first_name, last_name, email, phone, delivery_address, items	Збереження замовлення в базу даних	201, 400

Ендпоінт GET /api/books є ключовим для реалізації функціональних вимог FR-01 і FR-02. Цей ендпоінт повертає список книг у форматі JSON, де кожна книга містить поля: ідентифікатор, назву, автора, жанр, описання, ціну, рік видання та URL обкладинки. Клієнтська частина може використовувати параметри запиту genre_id для фільтрації за жанром та search для текстового пошуку за назвою або автором. Реалізація фільтрації на рівні ендпоїнту дозволяє зменшити обсяг переданих даних і покращити продуктивність, оскільки сервер надсилає клієнту лише релевантні записи [24].

Формат JSON-відповіді для ендпоїнту GET /api/books структурований як масив об'єктів книг. Кожен об'єкт містить всю необхідну інформацію для відображення у каталогу та детальній сторінці. Цей підхід забезпечує узгодженість даних на фронтенді – інформація про одну книгу завжди мала один і той самий формат незалежно від того, чи була вона отримана зі списку, чи запрошена окремо через ендпоінт GET /api/books/{id}.

Ендпоінт POST /api/orders призначений для збереження замовлення у базу даних (вимога FR-08). Клієнт надсилає JSON-об'єкт, що містить персональні дані покупця (ім'я, прізвище, електронна пошта, телефон, адресу доставки) та массив позицій замовлення. Кожна позиція містить ідентифікатор книги, кількість примірників, ціну та назву на момент замовлення. Сервер валідує отримані дані, перевіряє наявність обов'язкових полів, перевіряє формат електронної пошти та

телефону, а потім зберігає замовлення у таблицю order. Статус 201 Created інформує клієнта про успішне створення ресурсу, а статус 400 Bad Request – про помилку у форматі запиту [24].

На рисунку 2.3 показано схему взаємодії між Angular-клієнтом та PHP-серверним API. Клієнт ініціює асинхронний HTTP-запит до ендпоїнту, сервер обробляє запит, звертаючись до бази даних MySQL, та повертає JSON-відповідь. Клієнт отримує дані, парсить їх і оновляє представлення на екрані користувача.

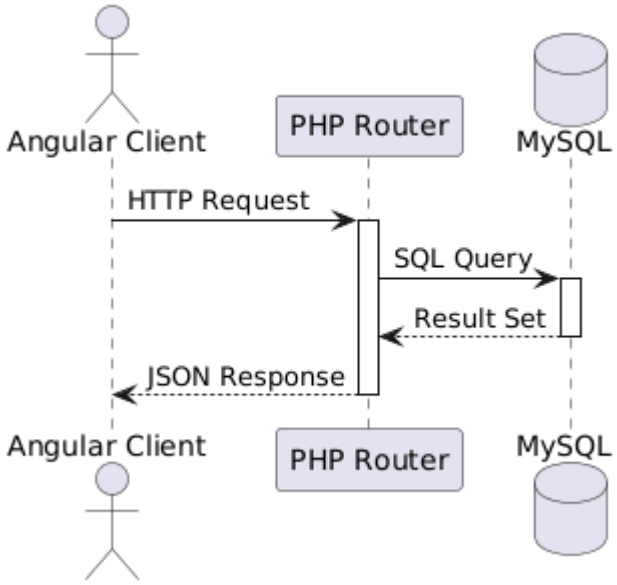

Рисунок 2.3 – Взаємодія клієнта та серверного API

Під час проєктування API важливо врахувати питання безпеки та оптимізації. REST API повинен захищатися від типових атак: SQL-ін'єкцій, XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery). У PHP-коді це забезпечується використанням підготовлених запитів (prepared statements) замість конкатенації рядків SQL, екрануванням вихідних даних та перевіркою заголовка Origin для запобігання CSRF-атакам [24]. Для оцінки коректності обробки даних API використовується метрика успішних відповідей, що розраховується за формулою, запропонованою у підрозділі 1.3 (формула 1.3).

Мінімальність проєктованого API полягає в тому, що він не включає операцій оновлення або видалення ресурсів, оскільки система не передбачає

адміністраторського інтерфейсу для керування каталогом. Дані каталогу вносяться безпосередньо до бази даних під час розгортання системи. Це рішення істотно спрощує реалізацію та тестування серверної частини та відповідає поточним вимогам до функціонального обсягу системи.

Таким чином, спроєктований REST API є лаконічним, але повним набором ендпоїнтів, що задовольняє всі функціональні вимоги вебдодатку та забезпечує чітку, типізовану взаємодію між клієнтською та серверною частинами. Архітектура API дотримується принципів REST-стилю та забезпечує надійну основу для розробки наступних компонентів системи.

2.4 Проєктування інтерфейсу та архітектури клієнтської частини на Angular

Проєктування клієнтської частини вебдодатку на Angular передбачає визначення архітектури компонентів, структури модулів, управління станом та інтеграцію з REST API. Angular як повноцінний фреймворк надає чітку організаційну структуру, що дозволяє розробляти масштабовані та легкі у супроводі застосунки [18]. Клієнтська частина книжкового інтернет-магазину організується навколо кількох ключових компонентів: перегляд каталогу, фільтрація книг, деталі окремої книги та кошик покупця.

Архітектура Angular-застосунку базується на компонентній моделі, де кожен компонент інкапсулює власний шаблон (HTML), логіку (TypeScript) та стилі (CSS). Починаючи з версії Angular 17, рекомендованим підходом є використання standalone-компонентів, які не потребують обов'язкового оформлення у NgModule [12]. Це спрощує структуру проєкту та робить його більш гнучким. Крім компонентів, система використовує сервіси для інкапсуляції бізнес-логіки: BookService взаємодіє з REST API для отримання даних про книги, CartService управляє станом кошика, а GenreService керує списком жанрів для фільтрації [15].

На рисунку 2.4 показано архітектурну діаграму клієнтської частини вебдодатку з основними компонентами та їхніми залежностями.

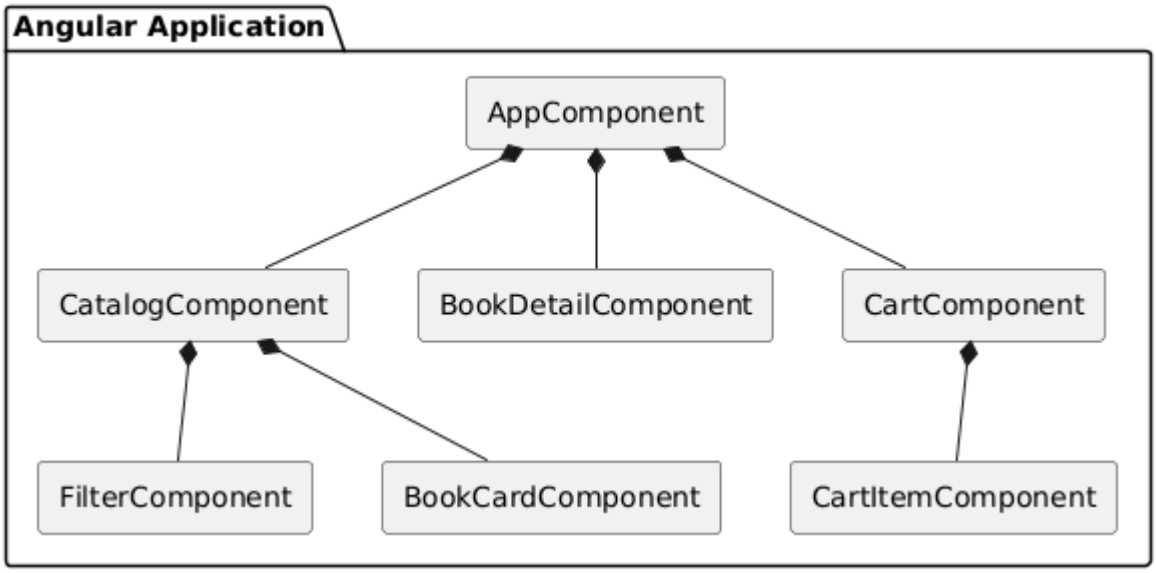

Рисунок 2.4 – Архітектура компонентів Angular-застосунку

Управління станом у застосунку реалізується за допомогою сигналів (Signals), нової реактивної примітиву, введеної в Angular 16 та стабілізованої в версії 17 [11]. Сигнали замінюють традиційний механізм виявлення змін на основі Zone.js та забезпечують більш дрібнозернисте реактивне обновлення інтерфейсу. CartService використовує сигнали для зберігання списку товарів, розрахунку загальної суми замовлення та синхронізації даних з localStorage для збереження кошика між сесіями (вимога FR-05).

Новий синтаксис управління потоком у шаблонах Angular 17 включає вбудовані структурні директиви @if, @for та @switch, які замінюють старіші *ngIf та *ngFor [11]. Цей синтаксис забезпечує кращу продуктивність та читаємість шаблонів. Наприклад, цикл для відображення списку книг реалізується як @for (book of books(); track book.id), де функція track оптимізує оновлення DOM, переслідуючи посилання на унікальний ідентифікатор кожної книги.

Для оформлення інтерфейсу використовується Angular Material, офіційна UI-бібліотека компонентів, що реалізує специфікацію Material Design компанії

Google [20]. Angular Material надає готові компоненти: MatCard для відображення книг у каталогу, MatButton для інтерактивних елементів, MatInput для полів введення, MatSelect для вибору жанру та MatIcon для іконок. Використання готових компонентів забезпечує консистентність дизайну, доступність та адаптивність інтерфейсу на різних розмірах екранів.

Маршрутизація у вебдодатку реалізується за допомогою Angular Router, вбудованого модуля для навігації між сторінками [12]. Застосунок містить три основні маршрути: / для каталогу книг, /books/:id для деталей окремої книги та /cart для сторінки кошика. Лінива загрузка маршрутів (lazy loading) не застосовується у цьому проєкті через його мінімальні розміри, проте архітектура передбачає можливість її додавання у майбутньому. У таблиці 2.4 наведено перелік основних компонентів застосунку з описом їхніх відповідальностей та взаємодії з API.

Таблиця 2.4 – Основні компоненти Angular-застосунку

Компонент	Відповідальність	Методи / Властивості	Взаємодія з API
AppComponent	Кореневий компонент, навігація	Header з посиланнями, маршрутизація	–
CatalogComponent	Виведення каталогу книг	books(), selectedGenre()	GET /api/books
FilterComponent	Фільтрація та пошук	applyFilter(), clearFilters()	–
BookCardComponent	Карта окремої книги	book input, addToCart()	–
BookDetailComponent	Детальна сторінка книги	book signal, quantity input	GET /api/books/{id}
CartComponent	Керування кошиком	cartItems(), totalPrice(), checkout()	POST /api/orders
BookService	Запити до API	getBooks(), getBookById(), getGenres()	REST API
CartService	Управління станом кошика	addItem(), removeItem(), clearCart(), save()	localStorage

Взаємодія з REST API реалізується через HttpClient, вбудований сервіс Angular для виконання HTTP-запитів [15]. BookService ін'єктується в компоненти через механізм впровадження залежностей (Dependency Injection), що дозволяє легко тестувати логіку компонентів, замінюючи реальний сервіс на

мок-версію [13]. Асинхронні операції обробляються за допомогою RxJS Observable та toSignal(), який перетворює потік даних у сигнал для реактивного оновлення шаблону.

Адаптивний дизайн інтерфейсу реалізується за допомогою медіа-запитів CSS та Flexbox. Каталог книг відображається як сітка, що автоматично адаптується до розміру екрана: на мобільних пристроях (до 600 px) – одна колонка, на планшетах (600-960 px) – дві колонки, на десктопах (понад 960 px) – три колонки. Це забезпечує вимогу нефункціональної вимоги щодо адаптивності на екранах від 320 до 1920 пікселів.

Таким чином, клієнтська частина вебдодатку спроєктована як модульна, компонентна SPA на сучасній версії Angular 17 з використанням сигналів для реактивного управління станом, Material Design для послідовного оформлення інтерфейсу та адаптивного макету для підтримки різних розмірів пристроїв. Архітектура дотримується принципів SOLID та забезпечує чистоту коду, легкість тестування та можливість подальшого розширення функціональності.

Проєктування вебдодатку книжкового інтернет-магазину охопило всі ключові аспекти розробки: від технічного завдання, що визначає цілі та вимоги, до детального проєктування бази даних, REST API та архітектури клієнтської частини. Технічне завдання, сформульоване відповідно до ДСТУ 3918-99, встановлює чітку межу роботи та критерії приймання системи. База даних спроєктована у третій нормальній формі з врахуванням посилальної цілісності та майбутнього масштабування. REST API забезпечує чистий, мінімальний інтерфейс для обміну даними, а архітектура клієнтської частини на Angular використовує сучасні можливості фреймворку для побудови інтерактивного, адаптивного інтерфейсу. Узгодженість між всіма рівнями системи та послідовність дизайну-архітектури забезпечують готовність до переходу до етапу реалізації.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБДОДАТКУ КНИЖКОВОГО ІНТЕРНЕТ-МАГАЗИНУ

3.1 Реалізація серверної частини та середовища розгортання на основі NestJS і Docker Compose

Реалізація серверної частини вебдодатку охоплює три взаємопов'язані складові: налаштування середовища розгортання на основі Docker Compose, ініціалізацію бази даних MySQL та розробку REST API засобами фреймворку NestJS. Усі три складові описуються єдиним файлом docker-compose.yml, що є центральним артефактом інфраструктури проєкту.

Середовище розгортання побудовано на основі Docker Compose і включає чотири сервіси: db (MySQL 8.0), backend (NestJS), frontend (Angular + Nginx) та adminer (вебінтерфейс адміністрування бази даних). Кожен сервіс ізольований у власному контейнері з чітко визначеними залежностями – зокрема, сервіси backend і adminer оголошені як залежні від db через директиву depends_on, що гарантує правильний порядок запуску. Усі сервіси об'єднані в спільну Docker-мережу, що дозволяє їм звертатись один до одного за іменами сервісів як за DNS-іменами: наприклад, NestJS звертається до MySQL за хостом db. Конфігурація сервісів наведена у таблиці 3.1.

Таблиця 3.1 – Конфігурація сервісів Docker Compose

Сервіс	Образ / Dockerfile	Порт хоста	Порт контейнера	Призначення
db	mysql:8.0	3306	3306	База даних MySQL
backend	./backend/Dockerfile	3000	3000	NestJS REST API
frontend	./frontend/Dockerfile	80	80	Angular + Nginx
adminer	adminer	8080	8080	Адміністрування БД

Дані MySQL зберігаються у іменованому Docker-волюмі db_data, що монтується до директорії /var/lib/mysql всередині контейнера. Це забезпечує

збереження даних між перезапусками контейнерів: навіть після виконання команди docker compose down і повторного docker compose up всі записи таблиць залишаються незмінними [12].

База даних ініціалізується автоматично при першому запуску контейнера db. MySQL-образ підтримує механізм автоматичного виконання SQL-скриптів із директорії /docker-entrypoint-initdb.d – саме туди монтується локальна папка ./database, що містить файл init.sql. Цей скрипт створює чотири таблиці (author, genre, book, order) та наповнює їх тестовими даними: п'ять жанрів, п'ять авторів і десять книжкових позицій. Структура бази даних відповідає ER-діаграмі, розробленій на етапі проєктування: між таблицями book і author, а також book і genre встановлено зв'язки типу «один до багатьох» через зовнішні ключі author_id і genre_id. Таблиця order зберігає склад замовлення у полі типу JSON, що дозволяє фіксувати назву та ціну книги на момент замовлення незалежно від подальших змін у каталозі.

Серверна частина реалізована засобами фреймворку NestJS на платформі Node.js 20. NestJS організовує код у модулі – кожен модуль відповідає за окремий ресурс системи. У проєкті визначено три функціональні модулі: BooksModule, GenresModule та OrdersModule. Кожен модуль містить три складові: сутність (entity), сервіс (service) та контролер (controller). Така структура відповідає принципу єдиної відповідальності та забезпечує чітке розмежування між рівнем даних, бізнес-логікою та HTTP-інтерфейсом [22].

Взаємодія з базою даних реалізована через TypeORM – об'єктно-реляційний відображувач, що дозволяє описувати таблиці у вигляді TypeScript-класів із декораторами. Наприклад, сутність Book визначає поля title, price, description, year, cover_url та зв'язки @ManyToOne з сутностями Author і Genre. TypeORM автоматично транслює операції над об'єктами у відповідні SQL-запити, що унеможливлює SQL-ін'єкції та спрощує роботу з реляційними даними [23]. Конфігурація підключення до бази даних зчитується з переменних середовища (DB_HOST, DB_PORT, DB_USER, DB_PASSWORD, DB_NAME),

що передаються через секцію environment у docker-compose.yml – це відповідає практиці зберігання конфігурації поза кодом [34].

Контролер BooksController обробляє два маршрути: GET /api/books для отримання списку книг та GET /api/books/:id для отримання деталей окремої книги. Метод findAll підтримує два необов'язкові параметри запиту: genre_id для фільтрації за жанром і search для текстового пошуку за назвою або іменем автора. Фільтрація реалізована через QueryBuilder бібліотеки TypeORM із застосуванням умов WHERE та оператора LIKE, що дозволяє виконувати часткове співпадіння рядків. Сервіс GenresService реалізує єдиний метод findAll, що повертає повний список жанрів для формування фільтра на клієнті. Контролер OrdersController обробляє POST /api/orders: перевіряє наявність обов'язкових полів (first_name, last_name, email, delivery_address, items) і у разі їхньої відсутності повертає відповідь зі статусом 400 Bad Request; при успішній валідації зберігає замовлення і повертає статус 201 Created. Загальну архітектуру модулів серверної частини проілюстровано на рисунку 3.1.

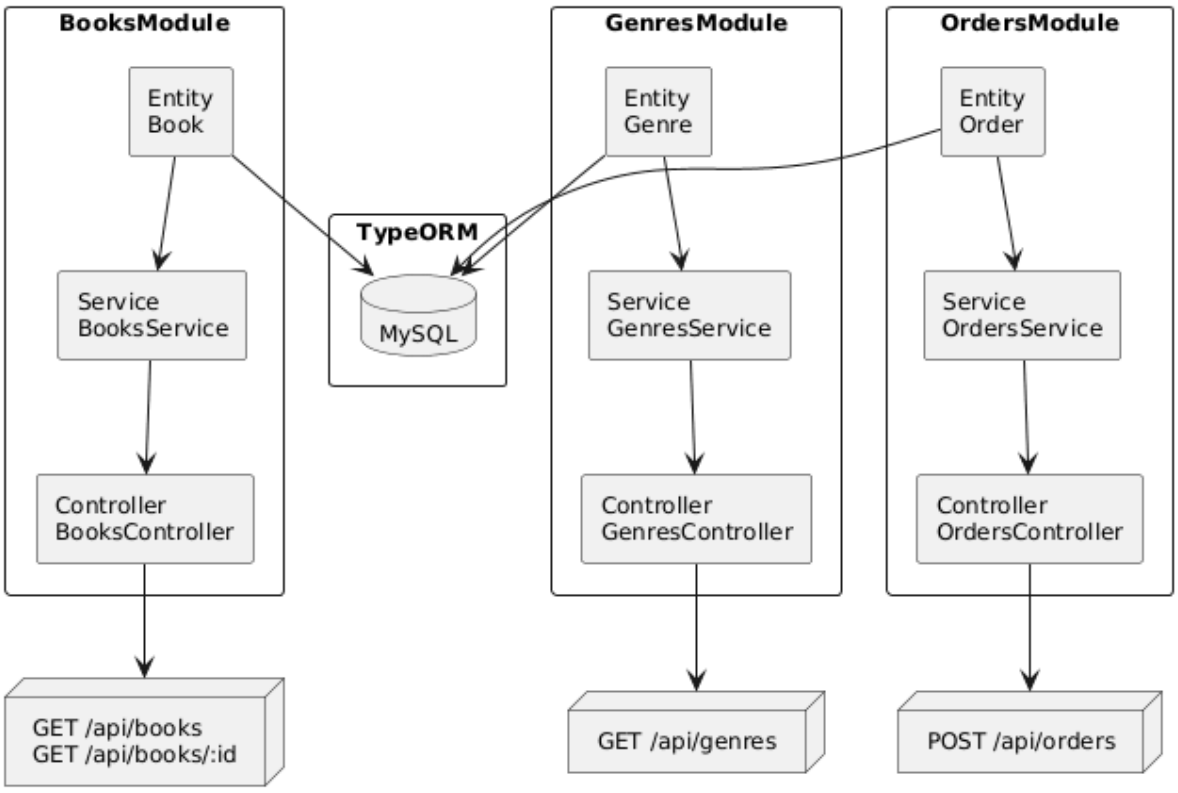

Рисунок 3.1 – Модульна структура серверної частини NestJS

Міжсервісна взаємодія між Angular-клієнтом і NestJS-бекендом забезпечується налаштуванням CORS у файлі main.ts: виклик app.enableCors({ origin: '*' }) дозволяє браузеру виконувати запити до API з будь-якого походження, що є необхідним у локальному середовищі розгортання. У продуктивному середовищі значення origin слід обмежити конкретним доменом [32]. Nginx у контейнері frontend налаштований на проксіювання запитів із префіксом /api до сервісу backend:3000, що дозволяє клієнтській частині звертатись до API через відносні URL без явного зазначення хоста і порту бекенду.

Збірка серверної частини виконується у два етапи, описані у Dockerfile: спочатку встановлюються залежності (npm install), потім виконується компіляція TypeScript у JavaScript (npm run build), і фінальний контейнер запускає скомпільований код командою node dist/main. Такий підхід забезпечує передбачуване середовище виконання незалежно від операційної системи розробника [44]. Результат успішного запуску всіх контейнерів та доступність API перевіряється через інтерфейс Adminer, що відкривається у браузері за адресою http://localhost:8080, як показано на рисунку 3.2.

Рисунок 3.2 – Інтерфейс Adminer з підключенням до бази даних bookshop

Таким чином, серверна частина вебдодатку реалізована як повністю контейнеризована система з чіткою модульною структурою, автоматичною ініціалізацією бази даних і типобезпечним доступом до даних через TypeORM. Усі налаштування середовища винесені в змінні оточення, а взаємодія між сервісами здійснюється виключно через визначені мережеві інтерфейси Docker Compose.

3.2 Реалізація клієнтської частини засобами Angular

Клієнтська частина вебдодатку реалізована як односторінковий застосунок (SPA) на базі Angular 19 з використанням підходу standalone-компонентів, що є рекомендованою практикою починаючи з Angular 17 [40]. Застосунок складається з п'яти компонентів, трьох сервісів і двох моделей даних, організованих у чітку файлову структуру всередині директорії src/app. Збірка для продуктивного середовища виконується командою ng build, після чого статичні файли розміщуються під управлінням Nginx у Docker-контейнері.

Точкою входу застосунку є файл main.ts, що викликає функцію bootstrapApplication з кореневим компонентом AppComponent та об'єктом конфігурації appConfig. Конфігурація, визначена у app.config.ts, реєструє три провайдери: provideRouter(routes) для клієнтської маршрутизації, provideHttpClient() для виконання HTTP-запитів до REST API та provideAnimations() для підтримки анімацій Angular Material [30]. Маршрутизація визначена у файлі app.routes.ts і містить три маршрути: кореневий / відображає CatalogComponent, маршрут /books/:id – BookDetailComponent, маршрут /cart – CartComponent.

Кореневий компонент AppComponent є мінімальним: він імпортує NavbarComponent і RouterOutlet та містить лише два елементи у шаблоні – навігаційну панель і точку підстановки маршрутів. NavbarComponent реалізує верхню панель на основі MatToolbar з назвою магазину, що є посиланням на

головну сторінку, та іконкою кошика з бейджем, що відображає кількість товарів у кошику. Значення бейджа прив'язане до обчислюваного сигналу cart.count() сервісу CartService і оновлюється реактивно без ручного управління підписками.

Центральним екраном застосунку є CatalogComponent, що реалізує функціональні вимоги FR-01 і FR-02. Компонент ініціалізує два сигнали: books для зберігання списку отриманих книг та genres для списку жанрів фільтра. При ініціалізації (ngOnInit) виконуються два паралельні запити до API: отримання списку жанрів і завантаження початкового каталогу. Фільтрація реалізована на рівні сервера: при зміні значення поля пошуку або вибраного жанру викликається метод loadBooks(), що формує новий HTTP-запит із відповідними параметрами genre_id і search. Завдяки цьому клієнт отримує лише релевантні дані, а не фільтрує весь каталог локально. Індикатор завантаження на основі MatProgressSpinner відображається під час очікування відповіді від сервера через сигнал loading. Список книг відображається у вигляді адаптивної сітки CSS Grid з автоматичним розрахунком кількості колонок залежно від ширини екрана. Зовнішній вигляд сторінки каталогу наведено на рисунку 3.3.

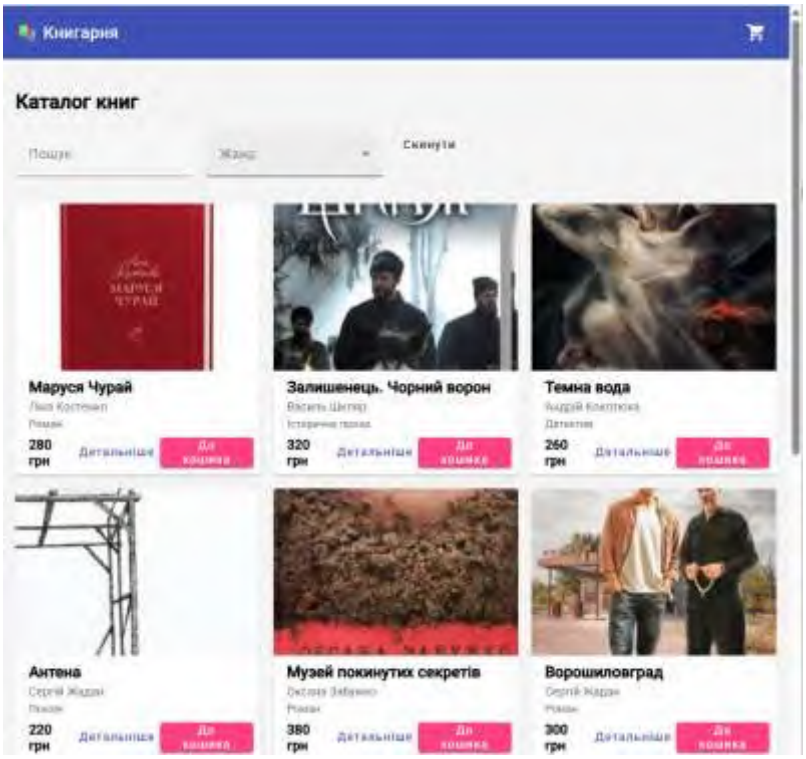

Рисунок 3.3 – Сторінка каталогу книг

Кожна книга у каталозі відображається через BookCardComponent, що отримує об'єкт книги через вхідний параметр @Input() book. Компонент побудований на основі MatCard і містить зображення обкладинки, назву, ім'я автора, жанр, ціну та дві кнопки: «Детальніше» для переходу на сторінку книги і «До кошика» для негайного додавання без переходу. При натисканні «До кошика» компонент викликає метод cart.addItem() з кількістю 1, що є типовою поведінкою для e-commerce каталогів [24].

BookDetailComponent реалізує функціональні вимоги FR-03 і FR-04. При ініціалізації компонент зчитує параметр id з активного маршруту та виконує запит GET /api/books/:id. Отримані дані зберігаються у сигналі book. Шаблон відображає повну інформацію про книгу: обкладинку, назву, автора, жанр, рік видання, опис і ціну. Додатково реалізований лічильник кількості примірників із кнопками збільшення і зменшення, при цьому значення не може бути меншим за 1. Кнопка «До кошика» передає у CartService поточну кількість разом із ідентифікатором, назвою та ціною книги. Сторінку деталей книги наведено на рисунку 3.4.

Рисунок 3.4 – Сторінка деталей книги

Управління станом кошика централізовано у сервісі CartService, що ін'єктується у будь-який компонент через механізм DI [20]. Сервіс використовує три реактивні примітиви: сигнал items для зберігання масиву позицій, обчислюваний сигнал total для автоматичного підрахунку загальної суми та обчислюваний сигнал count для кількості одиниць товару. При кожній зміні масиву items сервіс зберігає його серіалізований стан у localStorage браузера, що забезпечує виконання вимоги FR-05 – збереження кошика між сесіями. При повторному відкритті застосунку метод loadFromStorage(), що викликається безпосередньо в ініціалізаторі сигналу, відновлює попередній стан кошика.

Сторінку кошика з формою замовлення наведено на рисунку 3.5.

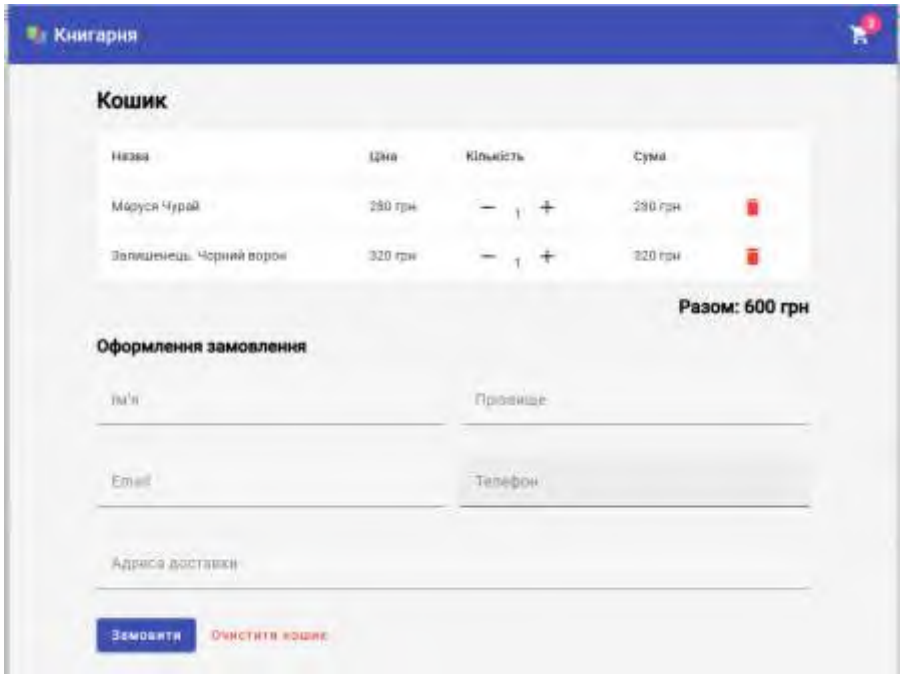

Рисунок 3.5 – Сторінка кошика з формою оформлення замовлення

CartComponent реалізує функціональні вимоги FR-06, FR-07 і FR-08. Список товарів відображається у вигляді таблиці MatTable з колонками: назва, ціна за одиницю, кількість із кнопками зміни, сума по позиції та кнопка видалення. Під таблицею розміщено форму оформлення замовлення з полями імені, прізвища, електронної пошти, телефону та адреси доставки, реалізовану через двостороннє зв'язування [(ngModel)]. Перед відправленням виконується

клієнтська валідація обов'язкових полів; у разі помилки відображається повідомлення через MatSnackBar. При успішному збереженні замовлення на сервері кошик очищається і форма скидається до початкового стану.

Взаємодія з REST API інкапсульована у двох сервісах. BookService реалізує три методи: getBooks() з необов'язковими параметрами фільтрації, getBook(id) для отримання окремої книги та getGenres() для списку жанрів. OrderService реалізує єдиний метод createOrder(), що виконує POST /api/orders. Обидва сервіси використовують HttpClient і повертають Observable, що відповідає реактивній моделі Angular [40]. Усі HTTP-запити використовують відносні URL із префіксом /api, що проксіюються Nginx до контейнера backend, завдяки чому клієнтський код не містить жодних явних посилань на адресу або порт серверної частини.

Збірка клієнтської частини для продуктивного середовища виконується у Dockerfile командою ng build, результатом якої є оптимізований набір статичних файлів у директорії dist/. Ці файли копіюються до образу Nginx, що обслуговує їх як статичний контент. Конфігурація Nginx містить директиву try_files $uri $uri/ /index.html, що є обов'язковою для коректної роботи клієнтської маршрутизації SPA: без неї пряме звернення до URL на кшталт /books/3 або /cart повертало б помилку 404 від веб-сервера.

3.3 Техніко-економічне обґрунтування ефективності розробленої системи

Техніко-економічне обґрунтування є обов'язковим етапом оцінки доцільності розробки та впровадження програмного забезпечення. Оцінка ефективності ІТ-проєктів здійснюється за трьома етапами: оцінка витрат на інформаційні технології, оцінка вигод від впровадження та розрахунок економічної ефективності [4]. У даному підрозділі застосовується фінансовий

підхід на основі показників прямих витрат на розробку, повернення інвестицій (ROI) та чистої теперішньої вартості (NPV).

Розроблений вебдодаток є власною розробкою без придбання комерційного програмного забезпечення: весь стек – Angular, NestJS, MySQL, Docker – є відкритим програмним забезпеченням з відкритим кодом і не потребує ліцензійних витрат. Технічне забезпечення також не закуповувалося – розробка виконувалася на наявному персональному комп'ютері з використанням VirtualBox і Alpine Linux. Отже, основною статтею витрат є оплата праці розробника. Трудомісткість розробки визначається на основі переліку виконаних робіт, наведеного у таблиці 3.2.

Таблиця 3.2 – Трудомісткість розробки вебдодатку книжкового інтернет-магазину

№	Вид роботи	Трудомісткість, год
1	Аналіз предметної області та проєктування	20
2	Налаштування середовища (Docker, Alpine, MySQL)	10
3	Розробка серверної частини (NestJS, TypeORM)	25
4	Розробка клієнтської частини (Angular, Angular Material)	35
5	Інтеграція, тестування та відлагодження	15
6	Документування (пояснювальна записка)	15
Разом		120

Витрати на оплату праці розраховуються за погодинною ставкою junior-розробника на ринку праці України у 2024-2025 роках, яка в середньому становить 200 грн/год [22]. Витрати на відрахування на єдиний соціальний внесок (ЄСВ) складають 22% від фонду оплати праці. Прямі витрати на розробку визначаються за адаптованою формулою [34]:

$$ВП = ВОП + ВВСЗ + ВІ, \quad (3.1)$$

де ВОП – витрати на оплату праці, грн;

ВВСЗ – витрати на відрахування на соціальні заходи, грн;

ВІ – інші прямі витрати (електроенергія, інтернет), грн.

Підставляючи числові значення: ВОП = 120 × 200 = 24000 грн; ВВСЗ = 24000 × 0,22 = 5280 грн; ВІ = 500 грн (витрати на електроенергію та інтернет за період розробки). Таким чином, ВП = 24000 + 5280 + 500 = 29780 грн. Щорічні витрати на утримання системи (ВУТР) оцінюються мінімально – оновлення залежностей та можливе розширення функціоналу потребуватимуть орієнтовно 20 год/рік, що становить 20 × 200 × 1,22 = 4880 грн/рік.

Впровадження книжкового інтернет-магазину забезпечує як прямі, так і якісні ефекти [4]. Прямий ефект полягає у можливості цілодобового прийому замовлень без залучення додаткового персоналу. Для малого книжкового магазину з середнім щомісячним оборотом у 50000 грн перехід на онлайн-продажі дозволяє збільшити виручку орієнтовно на 15-20% завдяки розширенню географії продажів та зручності для покупця [31]. При консервативній оцінці приріст виручки складатиме 7500 грн/міс або 90000 грн/рік. Якісні ефекти охоплюють: скорочення часу обробки замовлень завдяки автоматизації, підвищення точності обліку через зберігання замовлень у базі даних, покращення клієнтського досвіду через зручний інтерфейс і фільтрацію каталогу.

Для оцінки ефективності проєкту розраховуються показники ROI та NPV на горизонті три роки [34]. Середньорічний чистий прибуток від впровадження, враховуючи приріст виручки та щорічні витрати на утримання, складає: $AP = 90\,000 - 4\,880 = 85\,120$ грн/рік. Показник бухгалтерської рентабельності інвестицій розраховується за формулою [34]:

$$ROI = \frac{AP}{(INV_1 + INV_2)/2} \times 100\%, \quad (3.2)$$

де AP – середньорічний чистий прибуток, грн;

INV_1 – обсяг інвестицій на початок періоду (вартість розробки), грн;

INV_2 – залишкова вартість інвестицій на кінець розрахункового періоду (приймається рівною 0 після повної окупності).

Підставляючи: $ROI = 85120/(29780/2) \times 100 = 572\%$. Значення ROI суттєво перевищує середню ставку банківського депозиту (~15% річних станом на 2025 рік), що підтверджує економічну доцільність проєкту.

Чиста теперішня вартість розраховується за формулою при ставці дисконтування $i = 0{,}20$ (20% як орієнтир для IT-проєктів малого бізнесу в Україні) [4]:

$$NPV = \sum_{t=1}^{n} \frac{CF_t}{(1+i)^t} - INV_0, \tag{3.3}$$

де CF_t – грошовий потік за рік t, грн;

INV_0 – початкові інвестиції, грн;

i – ставка дисконтування;

n – горизонт розрахунку (3 роки).

Грошовий потік за кожен рік приймається рівним $CF = 85\,120$ грн. Розрахунок NPV наведено у таблиці 3.3.

Таблиця 3.3 – Розрахунок чистої теперішньої вартості проєкту

Рік (t)	Грошовий потік CF, грн	Коефіцієнт дисконтування $(1+0{,}20)^t$	Дисконтований CF, грн
1	85120	1,200	70933
2	85120	1,440	59111
3	85120	1,728	49259
Разом	255360		179303

Оскільки NPV = 179303 − 29780 = 149523 грн і NPV > 0, проєкт є економічно ефективним: за три роки він генерує 149523 грн чистого дисконтованого прибутку понад початкові інвестиції. Термін окупності, визначений як відношення початкових інвестицій до середньорічного грошового потоку, складає приблизно 29780 / 85120 ≈ 0,35 року, тобто менше чотирьох місяців від моменту запуску системи.

Таким чином, було описано практичну реалізацію вебдодатку книжкового інтернет-магазину: розгорнуто середовище на основі Docker Compose з чотирма контейнерами, реалізовано серверну частину на NestJS з TypeORM та базою даних MySQL, а також клієнтську SPA-частину на Angular 19 з використанням Angular Material, реактивних сигналів і клієнтської маршрутизації. Проведено техніко-економічне обґрунтування, за результатами якого встановлено, що загальні прямі витрати на розробку склали 29780 грн, показник ROI перевищує 500%, а чиста теперішня вартість за трирічний горизонт становить 149523 грн, що підтверджує високу економічну ефективність і доцільність розробленого програмного продукту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнута її початкова мета та вирішені усі поставлені завдання. Було проаналізовано наукову, методичну та нормативну літературу з теми розроблення вебдодатків для електронної комерції, зокрема щодо клієнт-серверної архітектури, сучасних JavaScript-фреймворків та підходів до контейнеризації програмного забезпечення.

Проведено аналіз предметної галузі книжкової електронної комерції та огляд існуючих рішень українського ринку – Yakaboo, «Книгарня Є», Bookovka та інших. Визначено типовий мінімальний функціональний набір книжкового інтернет-магазину: каталог із фільтрацією, картка товару, кошик і оформлення замовлення. Проведено порівняльний аналіз архітектурних підходів SPA та MPA, JavaScript-фреймворків (React, Angular, Vue), серверних фреймворків для Node.js (Express, Fastify, NestJS) та UI-бібліотек (Angular Material, Bootstrap). За результатами аналізу обґрунтовано вибір технологічного стеку: Angular 19 та Angular Material для клієнтської частини, NestJS з TypeORM для серверної частини, MySQL 8.x як СКБД та Docker Compose як середовище розгортання. Сформульовано вісім функціональних і п'ять нефункціональних вимог до системи, побудовано діаграму прецедентів та встановлено кількісні критерії приймання готового продукту.

Деталізовано архітектуру вебдодатку. Спроєктовано реляційну базу даних у третій нормальній формі, що охоплює чотири сутності: book, author, genre та order, з визначеними зв'язками типу «один до багатьох» та механізмом збереження складу замовлення у полі типу JSON. Розроблено специфікацію REST API з чотирма ендпоінтами: отримання каталогу книг із фільтрацією, отримання деталей книги, отримання списку жанрів та збереження замовлення. Спроєктовано компонентну архітектуру клієнтської частини на Angular із використанням реактивних сигналів для управління станом і адаптивної сітки CSS Grid для відображення каталогу.

Описано розробку та розгортання системи. Реалізовано середовище на основі Docker Compose з чотирма контейнерами: MySQL, NestJS, Angular+Nginx та Adminer. Серверна частина реалізована у вигляді трьох модулів NestJS – BooksModule, GenresModule та OrdersModule – з типобезпечним доступом до даних через TypeORM. Клієнтська частина реалізована як SPA з п'ятьма standalone-компонентами, трьома сервісами та збереженням стану кошика між сесіями через localStorage. Конфігурація Nginx забезпечує коректну клієнтську маршрутизацію та проксіювання запитів до API.

Техніко-економічне обґрунтування показало, що загальні прямі витрати на розробку склали 29780 грн за умови використання виключно відкритого програмного забезпечення без ліцензійних витрат. Показник рентабельності інвестицій ROI перевищує 500%, чиста теперішня вартість за трирічний горизонт становить 149523 грн, а термін окупності – менше чотирьох місяців від моменту запуску системи, що підтверджує високу економічну ефективність і доцільність розробленого програмного продукту.

Таким чином, поставлені завдання вирішені у повному обсязі. Напрямками подальших досліджень є розширення функціональності системи: додавання адміністраторської панелі для керування каталогом без прямого доступу до бази даних, впровадження системи автентифікації користувачів із можливістю перегляду історії замовлень, інтеграція з платіжними сервісами та службами доставки. Перспективним є також перенесення системи на хмарну інфраструктуру, впровадження серверного рендерингу засобами Angular Universal для покращення SEO-показників, а також розробка мобільного застосунку на базі існуючого REST API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. eCommerce-2026: світові інсайти для українського ритейлу. URL: https://turumburum.ua/blog/ecommerce-svitovi-insayti-dlya-ukrayinskogo-riteylu (дата звернення: 05.05.2026).

2. E-commerce 2026: рік українських брендів на глобальному ринку. URL: https://e-export.ukrposhta.ua/e-commerce-2026-rik-ukrayinskyh-brendiv-na-globalnomu-rynku/ (дата звернення: 05.05.2026).

3. Електронна комерція в Україні: правила для бізнесу. URL: https://biz.ligazakon.net/analitycs/226702_elektronna-komertsya-v-ukran-pravila-dlya-bznesu (дата звернення: 05.05.2026).

4. Аналіз ринку книжкового видавничого ринку України. URL: https://inventure.com.ua/uk/analytics/investments/analiz-rinku-knizhkovogo-vidavnichogo-rinku-ukrayini (дата звернення: 05.05.2026).

5. Українське книжковий ринок продовжує зростати: видавництва, ритейл, тренди. URL: https://hub.kyivstar.ua/articles/ukrayinskij-knizhkovij-rinok-prodovzhuye-zrostati-vidavnicztva-ritejl-trendi (дата звернення: 05.05.2026).

6. TOP-10 Ukrainian online bookstore 2025. URL: https://uba.top/en/ukrainian-online-bookstores-1/ (дата звернення: 05.05.2026).

7. Julianne Agu. Single-Page Application or Multi-Page Application for 2024? URL: https://medium.com/@julianneagu/single-page-application-or-multi-page-application-for-2024-5227d5ceb585 (дата звернення: 05.05.2026).

8. Jain V. A comparative analysis of single page applications (SPAS) and multi page applications (MPAS). *International Journal of Core Engineering & Management*. 2022. Vol. 7, no. 04. P. 271–277. URL: https://doi.org/10.5281/zenodo.14956673 (дата звернення: 05.05.2026).

9. Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best? URL: https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/ (дата звернення: 05.05.2026).

10. Angular vs React vs Vue: який фреймворк обрати? URL: https://www.golden-team.org/ua/blog/angular-vs-react-vs-vue-yakij-frejmvork-obrati-stattya-vid-golden-team (дата звернення: 05.05.2026).

11. Angular 17 New Features Explained: Signals, @defer & More. URL: https://www.ionicframeworks.com/2026/04/angular-17-new-features-explained.html

12. Angular Roadmap. URL: https://angular.dev/roadmap (дата звернення: 05.05.2026).

13. Інжектор в Angular: що це таке і як працює. URL: https://foxminded.ua/shcho-take-inzhektor-v-angular/ (дата звернення: 05.05.2026).

14. Розбираємо таємницю Angular Dependency Injection. URL: https://dou.ua/forums/topic/44768/ (дата звернення: 05.05.2026).

15. Dependency injection in Angular. URL: https://angular.dev/guide/di (дата звернення: 05.05.2026).

16. Exploring Angular Versions 17 to 20: A Developer's Perspective. URL: https://medium.com/@anishcp/exploring-angular-versions-17-to-20-a-developers-perspective-2bb14e938004 (дата звернення: 05.05.2026).

17. OpenServer Panel. URL: https://ospanel.io/ (дата звернення: 05.05.2026).

18. Angular CLI: офіційна документація. URL: https://angular.dev/tools/cli (дата звернення: 05.05.2026).

19. Angular Material UI Component Library. URL: https://material.angular.dev/ (дата звернення: 05.05.2026).

20. Angular "на пальцях". Швидкий старт і базові концепції фреймворку. URL: https://surl.li/qroate (дата звернення: 05.05.2026).

21. Найновіші технології веб-розробки 2026: як створюються сучасні сайти. URL: https://wezom.com.ua/ua/blog/naynovishi-tehnologiyi-veb-rozrobki-2025-yak-stvoryuyutsya-suchasni-sayti (дата звернення: 05.05.2026).

22. База даних MySQL для веб-розробки та серверних проектів. URL: https://brander.ua/technologies/mysql (дата звернення: 05.05.2026).

23. Що таке REST API та його можливості. URL: https://iptel.ua/blog/article/rest-api (дата звернення: 05.05.2026).

24. Що таке PHP? URL: https://site-konstruktor.com.ua/php (дата звернення: 05.05.2026).

26. Функціональні та нефункціональні вимоги до програмного забезпечення. URL: https://wezom.com.ua/ua/blog/funktsionalni-ta-nefunktsionalni-vimogi-do-programnogo-zabezpechennya (дата звернення: 05.05.2026).

27. Нефункціональні вимоги: приклади, типи, підходи. URL: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/ (дата звернення: 05.05.2026).

28. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 05.05.2026).

29. Діаграма прецедентів. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів (дата звернення: 05.05.2026).

30. Створення інтернет-магазину з нуля: 9 ключових етапів. URL: https://wezom.com.ua/ua/blog/etapy-razrabotki-internet-magazina (дата звернення: 05.05.2026).

31. ДСТУ ISO/IEC 12207:2016 Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ: УкрНДНЦ, 2016. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=71824 (дата звернення: 05.05.2026).

32. Технічне завдання. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Технічне_завдання (дата звернення: 05.05.2026).

33. Технічні вимоги щодо створення, впровадження та супроводження програмного забезпечення вебпорталу. URL: https://eef.org.ua/wp-content/uploads/2023/08/Dodatok_6_Tehnichni_vymogy_na_portal_dodatok_2.pdf (дата звернення: 05.05.2026).

35. Нормалізація баз даних. URL: https://uk.wikipedia.org/wiki/Нормалізація_баз_даних (дата звернення: 05.05.2026).

36. Про схеми баз даних. URL: https://foxminded.ua/skhemy-bazy-danyh/ (дата звернення: 05.05.2026).

37. Навіщо потрібні ключі в базах даних. URL: https://foxminded.ua/kliuchi-bazy-danykh/ (дата звернення: 05.05.2026).

38. Омельчук А., Булатецька Л., Булатецький В. Огляд поширених хмарних інструментів побудови ER-діаграм для вивчення баз даних. *Physics and educational technology*. 2022. № 1. С. 62–69. URL: https://doi.org/10.32782/pet-2022-1-8 (дата звернення: 05.05.2026).

39. Зв’язки в базах даних: що це і як вони працюють. URL: https://foxminded.ua/zviazok-u-bazi-danykh/ (дата звернення: 05.05.2026).

40. Hello, nest! A progressive Node.js framework for building efficient, reliable and scalable server-side applications. URL: https://nestjs.com/ (дата звернення: 05.05.2026).

41. TypeORM – ORM for TypeScript and JavaScript. URL: https://typeorm.io/ (дата звернення: 05.05.2026).

42. Web Platform Design Principles. W3C Working Group Note. 2023. URL: https://www.w3.org/TR/design-principles/ (дата звернення: 05.05.2026).