МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: «**Розробка клієнтської частини системи управління бібліотекою з онлайн каталогом та резервуванням книг»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Бородій Ярослав Сергійович

Керівник: *викладач кафедри ІТАД,*
Мацевич Денис Володимирович

Рецензент: *розробник програмного забезпечення в ТОВ ОБНОВА-ЄВРОШОП, викладач кафедри інформаційних технологій та аналітики даних НаУОА*
Шведюк Володимир Володимирович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)

Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: Розробка клієнтської частини системи управління бібліотекою з онлайн каталогом та резервуванням книг.

Автор : Бородій Ярослав Сергійович.

Науковий керівник: викладач-стажист, Мацевич Денис Володимирович.

Захищена «.....»..... 20____ року.

Пояснювальна записка до кваліфікаційної роботи: 82 с., 76 рис., 2 табл., 2 додатки, 17 джерел.

Ключові слова: бібліотека, інформаційна система, клієнтська частина, онлайн-каталог, бронювання книг, веб-розробка, HTML, CSS, JavaScript, Vue.js, REST API, інтерфейс користувача, автоматизація, інформаційні об’єкти, UI/UX дизайн, тестування, профіль користувача, вподобання, резервування, фронтенд, архітектура застосунку, цифрова трансформація.

Короткий зміст праці:

У кваліфікаційній роботі розглядаються теоретичні та практичні аспекти створення клієнтської частини веб-застосунку для автоматизації бібліотечних процесів. У першому розділі проаналізовано сучасні функції бібліотек, визначено їх роль у суспільстві та проблематику ручного обліку, проведено огляд існуючих аналогів електронних бібліотек. У другому розділі описано інформаційні об'єкти, REST-архітектуру для API, принципи взаємодії фронтенду з сервером, структуру компонентів, архітектуру проєкту, а також використані інструменти веб-розробки

(HTML, CSS, JavaScript, Vue.js). У третьому розділі реалізовано основні функції інтерфейсу: реєстрацію, логін, пошук, перегляд каталогу, бронювання та вподобання книг, а також профіль користувача. Також описано процес ручного тестування функціоналу та адаптації системи до різних браузерів. У висновках узагальнено результати роботи, підтверджено ефективність розробленої клієнтської частини та її перспективність для впровадження в практику сучасних бібліотек.

SUMMARY

Keywords: library, information system, client-side, online catalog, book reservation, web development, HTML, CSS, JavaScript, Vue.js, REST API, user interface, automation, data objects, UI/UX design, testing, user profile, favorites, reservation, frontend, application architecture, digital transformation.

Summary of the work:

This qualification paper explores both theoretical and practical aspects of developing a client-side web application for automating library processes. The first chapter analyzes modern library functions, their societal role, the challenges of manual data handling, and a review of existing electronic library systems. The second chapter describes the system's data objects, the REST API architecture, principles of frontend-server interaction, component structure, application architecture, and web development tools (HTML, CSS, JavaScript, Vue.js). The third chapter implements key interface functions: registration, login, search, catalog viewing, book reservation and favorites, and the user profile. It also covers manual interface testing and system adaptation to different browsers. The conclusions summarize the project’s results and confirm the effectiveness and practical applicability of the developed client-side for modern libraries.

ЗМІСТ

ВСТУП

Сучасна бібліотека — це значно більше, ніж просто місце зберігання книг. Вона є багатофункціональним простором, що надає доступ до цінних ресурсів: від розвитку критичного мислення та збереження історичної спадщини спільнот — до підтримки підприємництва та інновацій. Чим активніше розвивається і зміцнюється роль бібліотек у суспільстві, тим більш обізнаною, відкритою до змін та стійкою стає сама спільнота.

Традиційна модель роботи бібліотеки передбачає значну кількість рутинних завдань. Бібліотекарі механічно ведуть облік літератури, оформляють видачу та повернення книг, реєструють користувачів, здійснюють пошук у каталогах і ведуть облік штрафів за прострочення. Такі процеси вимагають багато часу та ресурсів, а зростання обсягів фонду й кількості відвідувачів лише посилює ці виклики.

Оптимальним рішенням є впровадження інформаційної системи автоматизації бібліотечних процесів. Вона дозволяє централізовано управляти даними, забезпечує зручність, швидкість і безпеку обліку. Однак в Україні наразі бракує сучасних програмних продуктів, які б поєднували функціональність із комфортним та інтуїтивно зрозумілим інтерфейсом для користувачів. Тому розробка такої платформи є актуальним завданням, що відповідає сучасним вимогам цифрової трансформації бібліотечної сфери.

Мета дослідження – аналіз теоретичних аспектів застосування бази даних для ефективного керування процесами та розробка власної інформаційної системи для діяльності бібліотеки.

Задачі дослідження:

- аналіз предметної області й наявних аналогів;
- вибір оптимального рішення для створення системи;
- опис структури фронтенду;
- проєктування програмного та технічного функціоналу;
- тестування інтерфейсу.

Об‘єкт дослідження – персональна цифрова бібліотека як частина сучасного інформаційного середовища.

Предметом дослідження виступають методи та засоби проєктування й розробки інформаційної системи для онлайн бібліотеки.

Створення такого веб-сайту спрямоване на значне покращення взаємодії користувачів із ресурсами бібліотечного фонду, автоматизуючи велику частину рутинних справ і багатьох процесів, що виконувались у паперовому форматі, наприклад, облік, бронювання, а також збереження книг, які зацікавили користувача. Для того, щоб реалізувати відповідні процеси, застосовувались такі технології: фундаментальні інструменти при розробці фронтенд сторони системи – HTML, Vue.js, CSS та JavaScript.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Огляд предметної області

Бібліотека — це установа, яка забезпечує широкий доступ до інформаційних ресурсів та різноманітних послуг для всіх верств населення. Вона може функціонувати у різних форматах — державному, академічному, шкільному чи спеціалізованому, виконуючи роль важливого соціально-культурного осередку. Бібліотеки створюють сприятливі умови для навчання, досліджень і культурного розвитку, надаючи можливість кожному користувачеві не лише ознайомитися з літературою, а й здобути нові знання та навички. [1]

Сьогодні ці заклади активно розширюють спектр своїх функцій, спрямовуючи зусилля на підтримку місцевих громад. Вони організовують майстер-класи, творчі зустрічі, тренінги з пошуку роботи, консультації з користування електронними ресурсами та допомогу в оформленні офіційних документів. Окремі бібліотеки впроваджують мобільні сервіси, що дозволяють доставляти книги тим, хто не має можливості особисто відвідати заклад, тим самим роблячи інформацію ще більш доступною та наближеною до потреб суспільства. [2]

Процес діяльності бібліотеки визначається такими характеристиками: бібліотекар, працівник установи, має ряд завдань, які він може виконувати щоденно:

- Переглядати каталоги книг, інформацію про читачів;
- Додавати нову літературу;
- Записувати дату і час, коли видавалась книга користувачу і повернулась;

- Реєструвати кожного нового читача, і редагувати, за потреби, їхні дані;
- Переглядати, яка література не була повернена;
- Формувати звіти; наприклад, щоб швидко знаходити розташування конкретної книги в закладі, складати списки користувачів.

Зі сторони читача, то він може отримати доступ до перегляду інформації щодо установи, списку всіх ресурсів, що наразі доступні, і ті, які ще не повернуті, має можливість брати і повертати книги, також самостійно проводити пошук щодо наявності потрібної літератури, має право тимчасово користуватися матеріалами протягом 15 днів. [3]

Рис. 1.1. Вигляд публічної бібліотеки

Бібліотеки відіграють важливе значення як осередки знань й освітніх ресурсів, доцільно надаючи кожному читачу необхідний доступ до широкого спектра літератури для навчання та саморозвитку. Попри масштабність внеску, який роблять такі установи, вони стикаються з багатьма труднощами в управлінні інформацією, наприклад: облік книг, користувачів, обробка позик і повернень, а також формування звітів. Більшість перерахованих функцій, що проводяться, здебільшого вимагають застосування на практиці традиційних методів записів вручну, які врешті-решт можуть призвести до помилок, втрати різних даних чи пошкодження інформації, а це, у свою чергу, суттєво впливає на ефективність роботи бібліотеки.

Враховуючи сучасний розвиток технологій, цей проєкт стане вирішенням проблем, з якими стикаються такого типу установи. Завдяки веб-сайту стане можливим зручне управління інформацією, що суттєво покращить роботу бібліотеки та взаємодію користувачів з електронною платформою. Інтерфейс буде комфортним, візуально привабливим і зручним для навігації як для бібліотекарів, так і для читачів.

Якщо розглядати визначення В. А. Резніченка, то, за його трактуванням, електронна бібліотека представляє собою інтегровану інформаційну систему, що дозволяє накопичувати, зберігати й ефективно застосовувати різні колекції електронних повнотекстових, а також мультимедійних документів, які знаходяться у вільному доступі та зручному вигляді для користувачів. [4, с.5]

Додатково, можна очікувати, що такі веб-сайти бібліотек будуть виконувати функцію цифрових центрів підтримки та ресурсу для громадян, власне трансформуватимуться поступово в багатофункціональні системи. Створення даних онлайн-платформ потребує ретельно продуманого дизайну для того, щоб забезпечити зручний доступ до інформації та сервісів для широкої аудиторії. Саме це вимагає старанного

опрацювання фронтенд частини коду, того аспекту, який відповідальний за ефективну взаємодію користувача із системою; формує перше враження про бібліотеку, це про розробку інтуїтивно зрозумілого інтерфейсу з дотриманням правил коректного розташування елементів на сторінках, що сприятиме швидкому пошуку книг та зручному перегляду інформації тощо. [5]

1.2. Огляд наявних аналогів

Огляд інформаційних систем для бібліотек, які наразі є у вільному доступі, виступить визначним етапом з метою розробки новітніх рішень, адже таким чином надається змога оцінити, як вже застосовані підходи та технології можуть бути адаптовані для поліпшення управління бібліотечними ресурсами. Детальний розгляд подібних веб-сайтів дозволить не лише виявити їхні переваги та недоліки, що є важливим врахувати при розробці власної платформи, але й дасть можливість на глибшому рівні зрозуміти особливості структур, способи взаємодії з користувачами, і виявити, яких елементів не вистачає. Застосування даного методу допомагає не тільки уникнути повторення помилок, але й надає доступ щодо впровадження новітніх функцій, які зроблять систему більш ефективною та зручною.

Чтиво - https://chtyvo.org.ua [6]

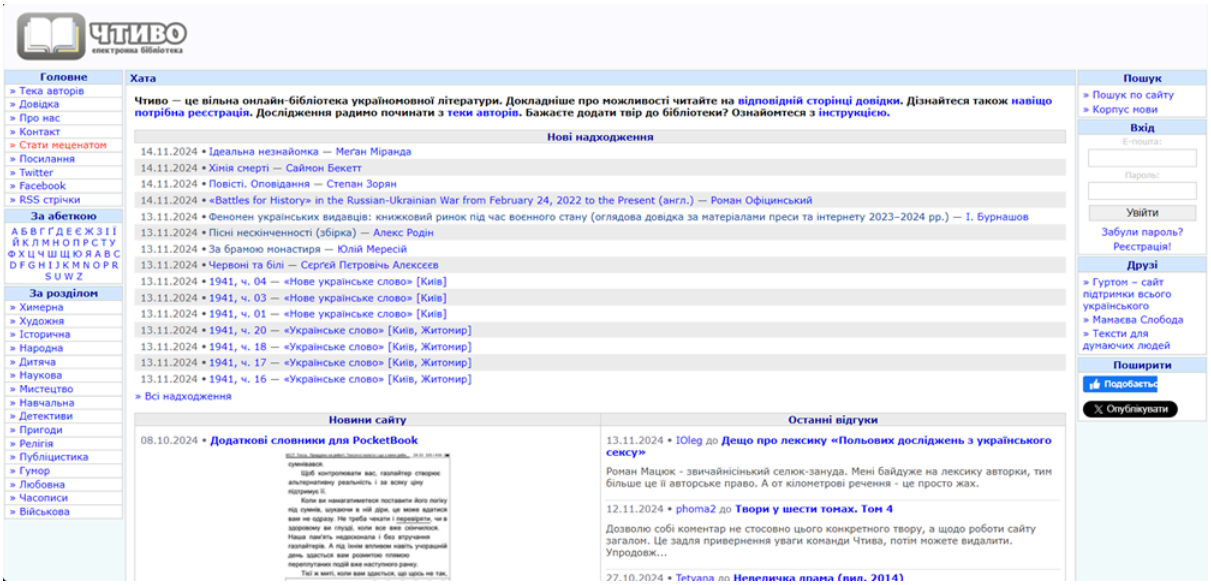


Рис. 1.2. Вигляд головної сторінки сайту №1

Переваги інформаційної системи: наявність великого вибору ресурсів, які зручно структуровані, та необхідного функціоналу, наприклад, пошук по сайту, заповнення форми для реєстрації, і досить важливо, є також відгуки користувачів.

Недоліки інформаційної системи: занадто перевантажений інтерфейс значною кількістю тексту, який поданий надто малим шрифтом; незручне розташування найважливіших елементів; відсутність зображення обкладинок книг; немає можливості забронювати книгу; застарілий дизайн системи.

KNIIGOGO - https://knigogo.top [7]

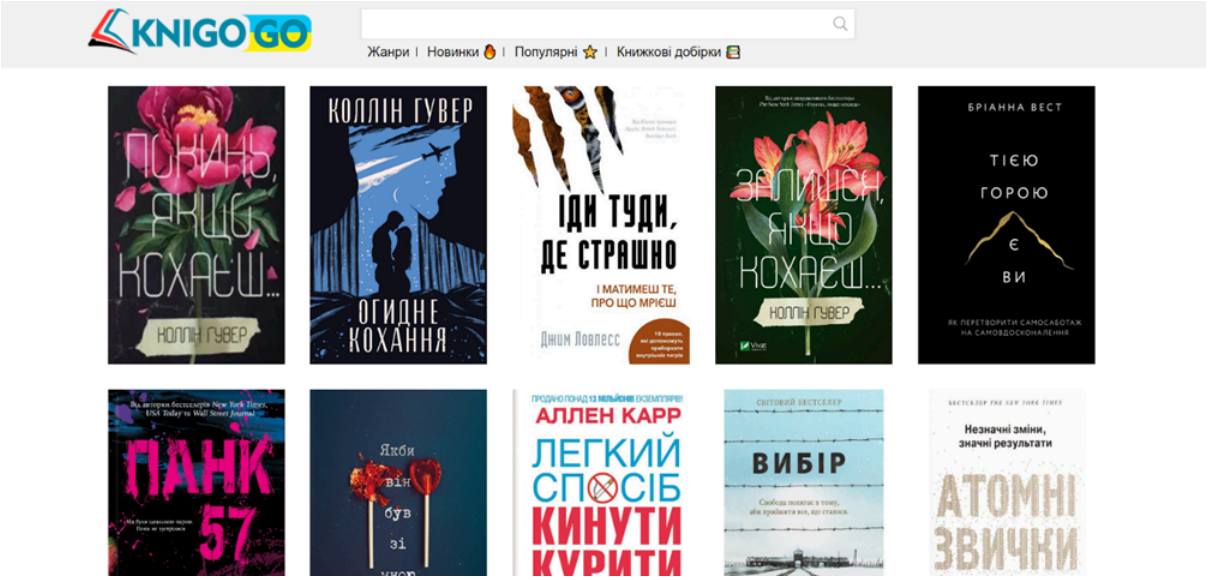


Рис. 1.3. Вигляд головної сторінки сайту №2

Переваги інформаційної системи: комфортний, простий та інтуїтивно зрозумілий інтерфейс, оскільки на першій сторінці розташовані великого розміру зображення обкладинок книг, тому користувачі можуть одразу перейти до каталогу, що є одним з найважливіших аспектів системи; наявний головний функціонал.

Недоліки інформаційної системи: відсутність можливості реєстрації власного кабінету та бронювання книг.

Libruk - https://libruk.com.ua [8]

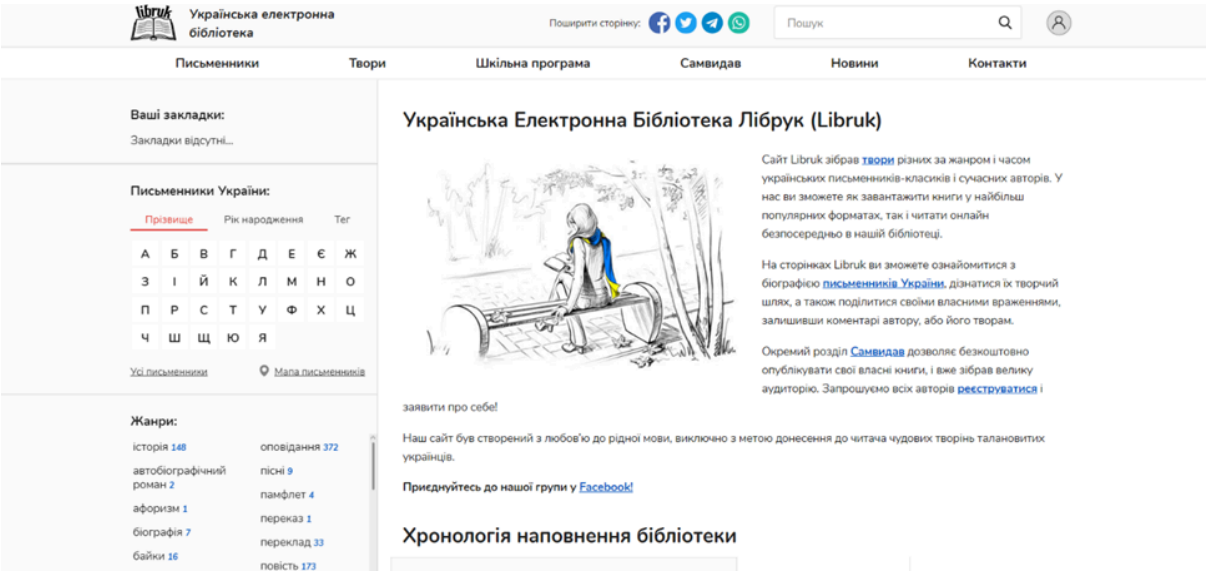


Рис. 1.4. Вигляд головної сторінки сайту №3

Переваги інформаційної системи: дизайн інтерфейсу виглядає інтуїтивно зрозумілим, простим, упорядкованим; доступна значна кількість різного матеріалу; є потрібний функціонал, зокрема можливість реєстрації особистого кабінету.

Недоліки інформаційної системи: відсутність змоги забронювати книги, а також немає рекомендації щодо книг, які є популярними; подекуди веб-сайт перевантажений текстом; недостатньо помітна панель пошуку.

Культура України - https://elib.nlu.org.ua [9]

Рис. 1.5. Вигляд головної сторінки сайту №4

Переваги інформаційної системи: наявний простий дизайн інтерфейсу з правильно упорядкованим необхідним функціоналом.

Недоліки інформаційної системи: одноманітна кольорова гамма, яка ускладнює навігацію на сайті; у каталозі розташовані дрібні зображення обкладинок книг, що не дає змоги швидко розпізнати їх; застарілий дизайн системи, а також відсутність можливості реєстрації та бронювання.

Національна бібліотка України імені В. І. Вернадського - http://www.nbuv.gov.ua [10]

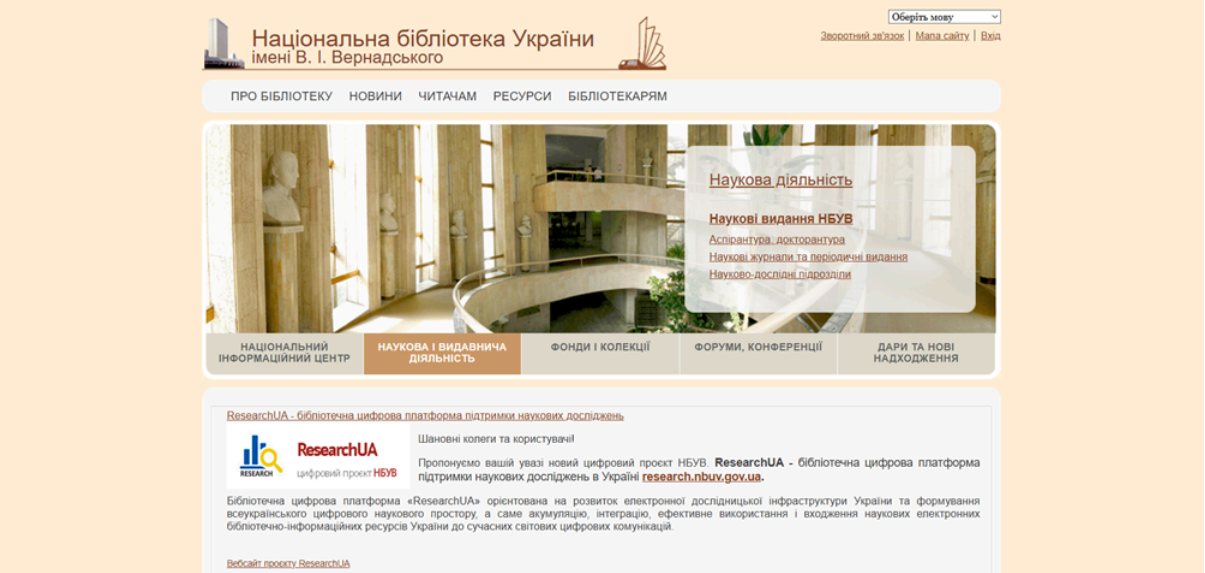


Рис. 1.6. Вигляд головної сторінки сайту №5

Переваги інформаційної системи: можливість реєстрації кабінету; розширений спектр послуг для кожного користувача зручне виділення підпунктів меню, які присвячені окремо читачам та бібліотекарям.

Недоліки інформаційної системи: перевантаженість текстом, а також дрібний шрифт, особливо на пунктах щодо контактної інформації та реєстрації; відсутність на головній сторінці панелі пошуку; одноманітна кольорова гамма; у деяких напрямках надто хаотичне розташування елементів, що сприяє ускладненню навігації по сайту і пошуку потрібних даних; книги не виступають як важливий аспект, який одразу помітний на платформі.

Отже, аналіз наявних інформаційних систем бібліотек допоміг визначити, що більшість сучасних рішень успішно автоматизують процеси каталогізації, обліку й пошуку ресурсів. Проте є випадки, в яких системи не мають комфортного сучасного дизайну інтерфейсу, не дають змоги забронювати книги або зареєструвати персональний кабінет на сайті, а також переповнені текстом дрібного шрифту. Це свідчить про актуальність створення власної інформаційної системи, яка врахує

специфіку роботи бібліотек, покращить доступ до ресурсів і підвищить якість обслуговування користувачів.

1.3. Формулювання завдання

Майбутня інформаційна система буде застосовуватись користувачами, які не є високо кваліфікованими у сфері комп'ютерних технологій, тому інтерфейс повинен бути простим та інтуїтивно зрозумілим. Перед розробкою платформи було необхідно окреслити перелік завдань, послідовне вирішення яких забезпечує досягнення поставленої мети:

- Підібрати ефективні засоби для розроблення платформи;
- Ознайомитись з процесами бекенд-розробки, зокрема особливості взаємодії з діючими API, принципи обробки запитів і передачі інформації між компонентами, а також організацію загальної роботи з базами даних та суміжними аспектами;
- Визначити архітектуру фронтенд-частини застосунку, що передбачає розробку структури клієнтського інтерфейсу. Цей етап охоплює модульне проєктування, зокрема головної сторінки, каталогу, особистого кабінету користувача та сторінок аутентифікації;
- Виокремити додаткові модулі, необхідні для реалізації функціональних можливостей застосунку. Зокрема, для встановлення й керування залежностями заплановано використати менеджер пакетів npm; для відображення сповіщень та повідомлень - бібліотеку sweetalert; для забезпечення захисту паролів користувачів - bcrypt; для додавання іконок – набір fontawesome;
- Визначити структуру розміщення функціоналу, передбачивши його логічний поділ між окремими файлами, модулями та компонентами, відповідно до принципів модульності та повторного використання коду;

- Реалізувати сторінки для аутентифікації користувачів, зокрема для входу до системи та реєстрації нового облікового запису. Після успішної реєстрації буде надаватись доступ до персонального кабінету і окремих функцій, доступних виключно авторизованим користувачам;
- Розробити пошук за книжками, що можна досягнути зі створенням API-запитів і фільтрів, які надаватимуть змогу, при введенні у пошуковий механізм ключових слів чи часткової назви, знаходити необхідну літературу;
- Створити сторінки із загальним списком книжок, з можливістю відкриття розширеної інформації про них, а також каталоги, які структуровані за темами, жанрами, і кожен має свою окрему сторінку з відповідним контентом;
- Розробити повноцінний модуль профіль користувача, завдяки якому можна буде отримати доступ до власного кабінету, де користувач може переглядати, редагувати або видалити особисту інформацію, бачити історію дій, список заброньованих книг, нараховані штрафи тощо.;
- Надати змогу користувачу залишати вподобання на книги, а також здійснювати бронювання літератури і скасування.

Висновок до розділу 1

Бібліотеки відіграють важливу роль як осередки знань, соціальної взаємодії та громадської підтримки. Однак, попри їхню значущість, існує нагальна потреба у впровадженні сучасних інформаційних систем, що дозволить підвищити ефективність надання послуг. Традиційні методи ведення обліку, зокрема ручне оформлення звітності, суттєво сповільнюють робочі процеси, підвищують ризик виникнення помилок і втрати даних.

Запровадження ретельно продуманого фронтенду веб-сайту суттєво покращить щоденну діяльність бібліотеки завдяки автоматизації рутинних завдань. Це, у свою чергу, дозволить зменшити навантаження на працівників, підвищить ефективність обслуговування користувачів та залучить ширшу аудиторію, сприяючи популяризації читання.

Створення зручної та інтуїтивно зрозумілої цифрової платорми забезпечить відвідувачам веб-сайту просту навігацію та позитивний досвід взаємодії. Завдяки продуманному інтерфейсу кожен зможе легко знайти необхідну інформацію та скористатись доступними послугами, що підвищить загальний рівень задоволеності від використання ресурсу.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Опис використаних інформаційних об’єктів системи

При розробці клієнтської частини майбутньої платформи важливо врахувати, з якими даними буде відбуватись взаємодія в системі. Для цього необхідно ознайомитись з різними об’єктами, що позначають та зберігають сукупність різних даних. Такі елементи є необхідними для коректного функціонування додатку, що забезпечить правильний обмін між клієнтською та серверною частинами, перегляд та редагування інформації.

Загальний огляд об’єктів, а саме обраних даних, які будуть використані:

Книги (books, copies) — один із найбільш важливих елементів системи, що містить відомості про назву, автора, жанр, рік видання, опис, кількість доступних примірників тощо. Такий процес включає перегляд інформації, додавання нових книг, редагування та видалення.

Користувачі (users) — об’єкт, що надає особисті дані тих осіб, що зареєструвались в систему. Елемент включає інформацію про ім’я, електронну пошту, роль, тобто читач або адміністратор, історію позик тощо. Головним аспектом є надання безпечного збереження та обробки даних користувачів.

Позика (loans) — елемент, який буде відображати те, що читач одержав книгу тимчасово; дату, коли передали літературу та отримали після завершення користування. Окрім того, буде висвітлюватись статус повернення.

Бронювання (reservations) — дозволяє користувачу забронювати книгу, якщо вона наразі не є доступною. Елемент, міститиме дані про дату резервування, очікувану дату отримання і статус виконання.

Вподобання (likes) — об’єкт, який зберігає інформацію про улюблені книги користувача. За допомогою таких даних можна буде регенерувати списки, відповідальні за особисті рекомендації.

Категорії розділення (publishers, authors) та жанри (genres) — допоміжні об’єкти, які класифікуватимуть книги за різними темами для зручного пошуку та фільтрації.

Повідомлення та сповіщення — елементи, які забезпечують інформування користувача про дату повернення матеріалів, підтвердження бронювання або зміну статусу позики.

Кожен такий об’єкт є взаємопов’язаний з іншими. За їх допомогою формуватимуться таблиці для збереження інформації, а самі дані надходитимуть на фронтенд у вигляді структур і використовуватимуться для відображення на сайті, редагування, пошуку, а також побудови інтерфейсу.

2.2. Взаємодія з діючим API

Створений API для програмного забезпечення, розробляється з метою взаємодії та роботи з клієнтською частиною, що надає можливість постійної та двосторонньої співпраці між ними.

Розглядаючи веб-розробку, дане рішення дозволить фронтенду запитувати бажану інформацію та отримувати відповідні дані, або виконувати певні дії в системі.

API надає розділення в програмному забезпеченні, тобто розподіляючи відповідальності між користувацьким інтерфейсом, що власне відповідає насамперед за досвід клієнтів та бекенд, який працює з логікою, даними та ефективною розробкою. [11]

Системи допускають застосування різних підходів, такі як:

REST – структура, яка найчастіше зустрічається в API. Її суть полягає у використанні найголовніших методів HTTP, що дозволяє

взаємодіяти з посиланнями бекенду. Ці API не мають стану та комунікують за допомогою стандартизованих форматів даних. [12]

SOAP – складніший та об'ємніший метод, що використовує тільки XML файли для взаємодії, є надійнішим та безпечнішим від REST, але й використовується в порівнянні з ним набагато менше, що, відповідно, зменшує його актуальність при розробці.

GraphQL – є альтернативою REST, який дозволяє клієнтам запитувати тільки ті дані, які їм необхідно. Цей протокол є хорошим інструментом, але потребує складнішого та довшого налаштування з боку бекенду.

WebSockets – використовується для взаємодії в реальному часі для комунікацій; корисний для застосунків з чатами та сповіщеннями.

У цьому проєкті було обрано архітектуру REST, що співпрацює через JSON за допомогою HTTP, який є стандартним підходом для сучасної розробки веб-додатків.

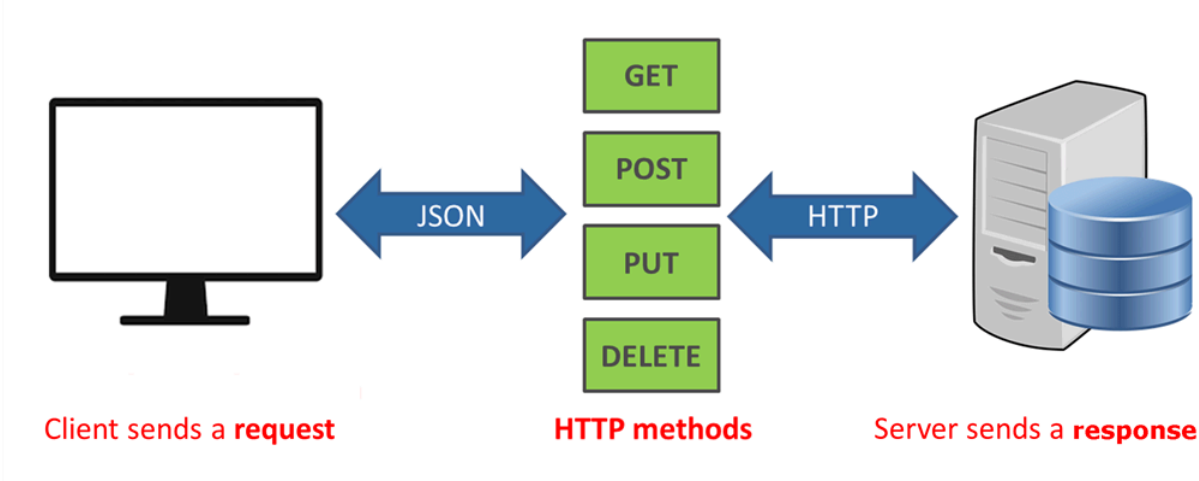


Рис. 2.1. Візуалізація роботи REST

HTTP стандарт ділиться на наступні основні методи: [13]

GET: отримання певного набору бажаних даних.

POST: відправка інформації для подальшого використання в бекенді.

PUT або PATCH: оновлення існуючих даних.

DELETE: видалення обраної інформації.

Кожен з таких типів був використаний у цьому проєкті. Далі буде наведено приклад з різних частин коду, що дадуть краще розуміння їх взаємодії.

Метод GET:

```
async fetchBooks() {
  try {
    const response = await axios.get("http://localhost:3000/api/books");
    this.books = response.data;
    console.log(this.books);
  } catch (error) {
    console.error("Error fetching books:", error);
    Swal.fire({
      icon: "error",
      title: "Помилка",
      text: "Не вдалося отримати список книг.",
    });
  }
},
```

Рис. 2.2. Виведення всіх книг

Тут можна побачити, як задається запит до серверу з уточненням ендпойнту та методу get, що дає можливість записати ці дані до змінної books, що будуть отримані з відповіді (response.data), та працювати надалі з нею.

Метод POST:

```
await axios.post(
  "http://localhost:3000/api/reservations",
  {
    bookId,
    status: "active",
    loandate: new Date().toISOString().split("T")[0],
    dueDate: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000)
      .toISOString()
      .split("T")[0],
  },
  {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  }
);
```

Рис. 2.3. Створення бронювання

Наступний код показує, як можна працювати з методом POST. На спеціальне посилання надсилається ідентифікатор книги, статус бронювання та дати, що необхідні для обробки в системі, також ці всі дані опрацьовуються на бекенді.

До того ж, тут було використано headers, що передають token користувача, через те, що ця функція є тільки обмежена для авторизованих осіб, тому його використання є необхідним та обов'язковим для правильного функціонування бронювань.

Метод PUT:

```
const response = await axios.put(
  "http://localhost:3000/api/user/update-profile",
  this.user,
  {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  }
);
```

Рис. 2.4. Оновлення даних профілю

Цей метод надсилає оновлені дані на сервер, тобто він оновлює змінену інформацію в профілі та використовує для цього PUT.

Метод DELETE:

```
try {
  await axios.delete(`http://localhost:3000/api/likes/${likeid}`, {
    headers: {
      Authorization: `Bearer ${localStorage.getItem("token")}`,
    },
  });
  this.likes = this.likes.filter((like) => like.likeid !== likeid);

  Swal.fire({
    icon: "success",
    title: "Видалено!",
    text: `"${likedBook.book_title}" видалено з вподобаних.`,
    confirmButtonColor: "#ff7e5f",
  });
```

Рис. 2.5. Прибирання вподобання з книги

На Рис.2.5 код зображає логіку при застосуванні методу DELETE, тому необхідно надіслати ідентифікатор книги прямо в маршруті для того, щоб бекенд міг віднайти та видалити обрану літературу з вподобаних, а користувача, для якого це необхідно зробити, визначають за допомогою

переданого токена. В кінці при успішному виконанні задається відповідь у вигляді сповіщення.

2.3. Опис структури до розробки фронтенду

2.3.1. Архітектура клієнтської частини

У сучасній веб-розробці архітектура додатку означає структурований підхід та організацію коду, компонентів, сервісів та логіку проєкту, який працює в браузері користувача. Вона відіграє ключову роль для забезпечення ефективності, підтримки та загальної роботи додатку.

Якісно реалізована архітектура надає можливість:

- Швидко знаходити та змінювати бажані функції, що, в свою чергу, надають підтримку;
- Зручне збільшення функціональних елементів застосунку, завдяки компонентній структурі;
- Легке повторне використання компонентів;
- Тестування є набагато швидшим та ефективнішим;
- Розділення відповідних функціональних елементів на логічні спільні блоки.

Цей застосунок використовує підхід, що надає можливість працювати з модульно-компонентною логікою. Головна ціль – розділення відповідальностей та повторне використання коду. Нижче наведена структура папок та деяких файлів, що були створені:

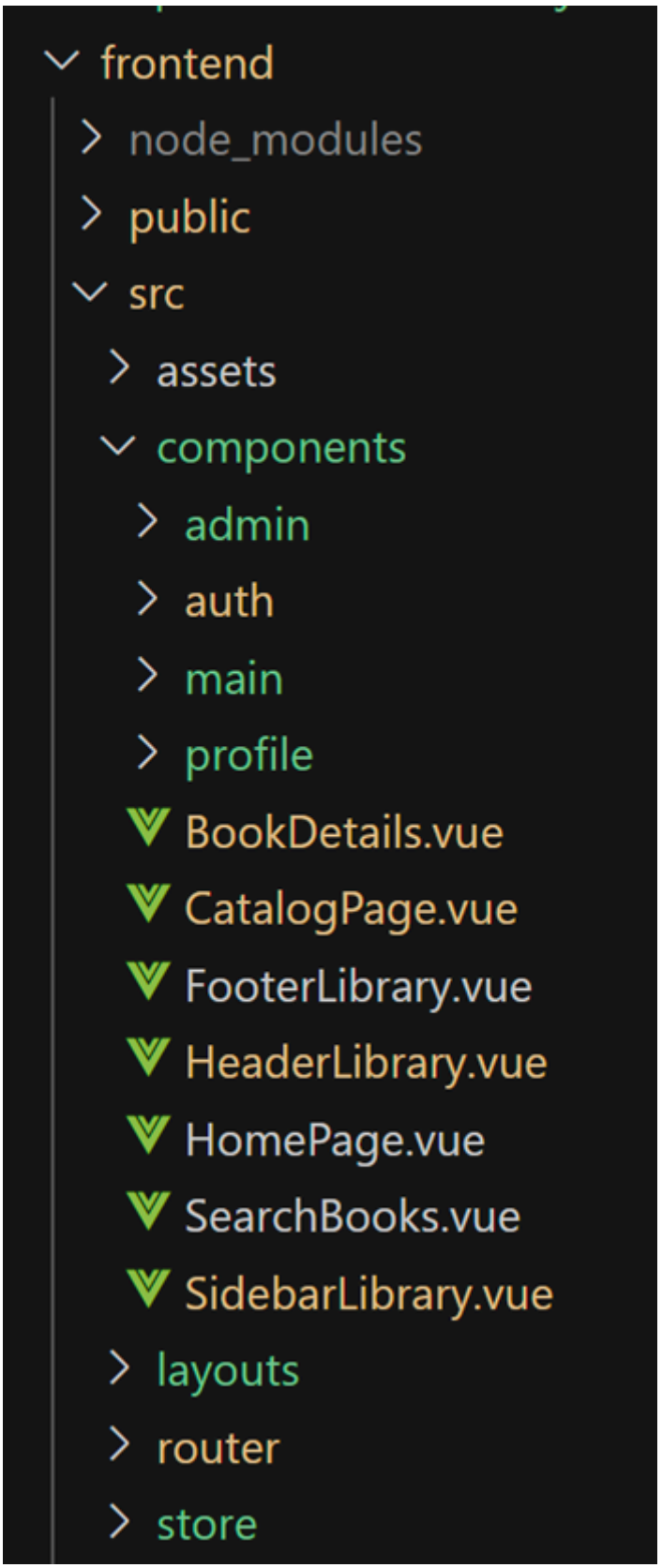


Рис. 2.6. Структура проєкту

Далі представлено коротку характеристику головних елементів цієї структури:

Node_modules – фіксує використані налаштування з node.

Public – папка, що зберігає статичні файли (index.html) та співпрацює напряму з браузером.

Assets – надає доступ до файлів, зазвичай фото для компонентів коду, що надає можливість використання їх на сторінці.

Components – загальна папка, що містить більшість коду проєкту та напряму зберігає UI файли.

Auth – надає доступ до компонентів, що відповідають за аутентифікацію користувача.

Main – зберігає файли, що використовуються для виведення інформації на основних сторінках.

Profile – описує частини компонентів, які утворюють сторінки для керування даними профілів користувачів.

Layouts – ця папка надає доступ до бажаної структури сторінок, які часто повторюються на різних маршрутах проєкту, надаючи різний вигляд та обрану схематичну розмітку. Це включає специфічні хедери, футери та бічні панелі, що обгортають контент, який міститься в components.

Router – зберігає маршрути, що розставляють межі між сторінками та дають можливість означити, які маршрути мають прив’язку до правильних компонентів з використанням layouts при потребі.

Store – папка, що задає глобальні змінні для конфігурації Vuex. Це, в свою чергу, включає дані для перевірки аутентифікації.

2.3.2. Налаштування модулів та допоміжних файлів

Під час створення сучасного проєкту, використовуючи професійну структуру розробки, неможливо оминути застосування модулів та інших утиліт, що дозволяють оптимізувати розробку інтерфейсу.

Фреймворки дають можливість розділяти різні відповідальності в таких файлах, надаватись спосіб конфігураційним видам буде окремо від файлів, що створюють веб-сайт, вони зазвичай використовуються для налаштування поведінки застосунку, процесу білдингу, залежності та інші функції.

Далі пропонується опис, що пояснює кожен з цих застосованих файлів та їх місце в системі.

Рис. 2.7. Вигляд package.json

Наданий файл є центром цього проєкту, пропонуючи можливість прописувати та вносити зміни до залежностей, коду та запусків.

Найважливіші частини включають:

- Задання назви проєкту та його деталей;
- Список залежностей в dependencies, що зв’язаний з node_modules та керує тим, які з розширень потрібно використовувати;
- Автоматичні scripts: для запуску, білду та тестування фронтенду.

package-lock.json – схожий файл до попередньо описаного, але включає конкретні встановлені залежності, контролюючи небажані конфлікти між версіями розширень.

babel.config.js – є конвертатором, що дає сучасному ES6+ коду можливість поєднання з старими версіями браузерів, є корисним, бо допомагає додатку працювати в різних середовищах.

.gitignore – розробляється для роботи з Git, що вказує, які файли потрібно проігнорувати при створенні комітах. Є необхідним для того, щоб не деплоїти вміст папок та файлів: node_modules, dist, env і т.д.

frontend > src > JS main.js > ...

```
1   import { createApp } from "vue";
2   import App from "./App.vue";
3   import router from "./router";
4   import store from "./store";
5
6   const app = createApp(App);
7
8   app.use(router);
9   app.use(store);
10  app.mount("#app");
11
```

Рис. 2.8. Вміст main.js

Показаний файл служить відправною точкою для цього серверу. Він ініціалізує та впроваджує загальні файли для відображення за допомогою DOM, підтримуючи розділення на бажані сторінки. Без цього файлу застосунок не можливо запустити, він з'єднує разом всі модулі, дозволяючи запуск інтерфейсу.

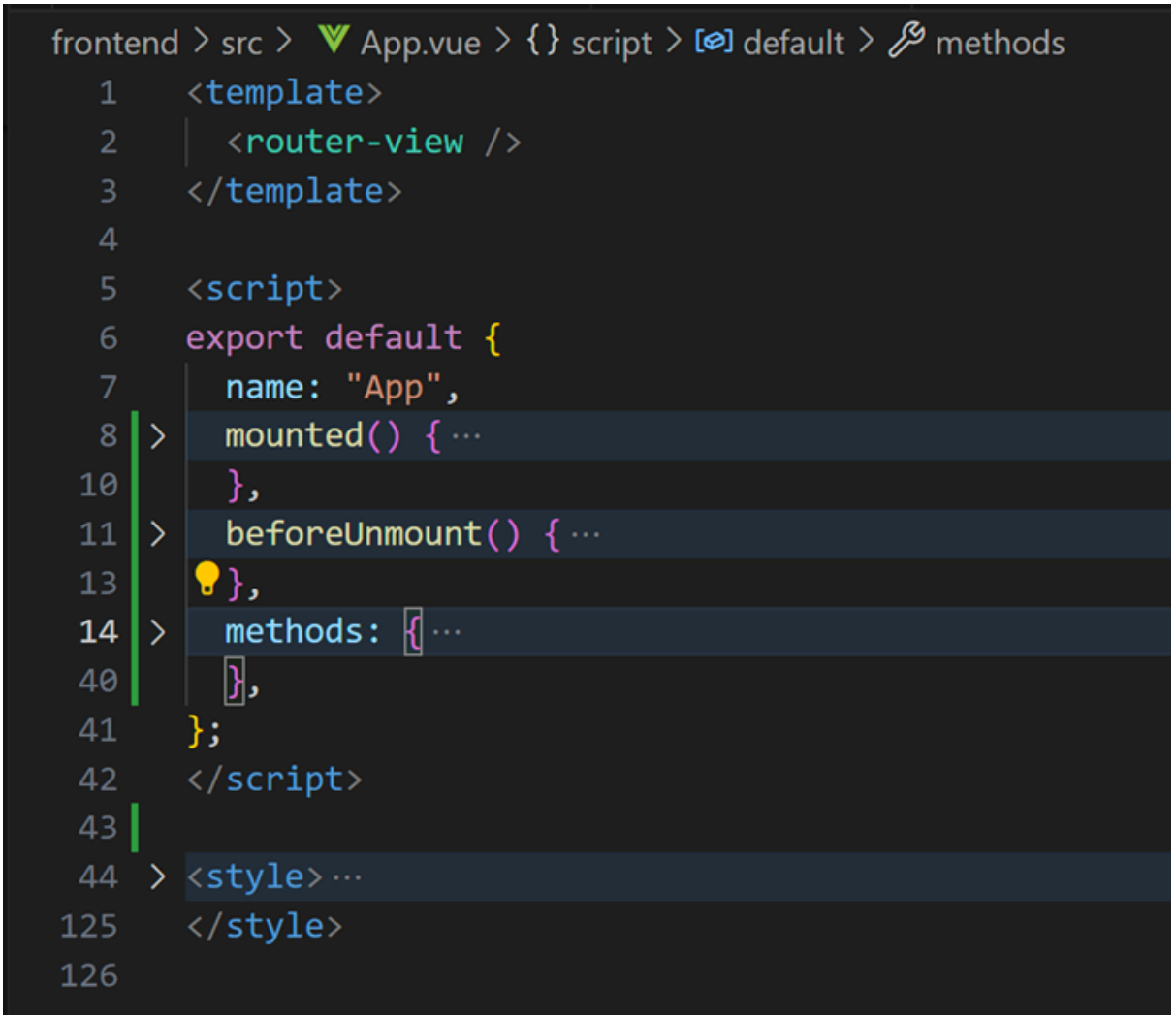


Рис. 2.9. Вміст App.vue

Наступний файл є кореневим компонентом застосунку. У такому випадку він є найнижчим в ієрархії компонентів та містить тільки загальні стилі та <router-view />, що відповідає за динамічне завантаження даних, в залежності від обраного маршруту, хедери і футери теж можуть в ньому зберігатись, але в цьому типі застосунку вони використані в layouts.

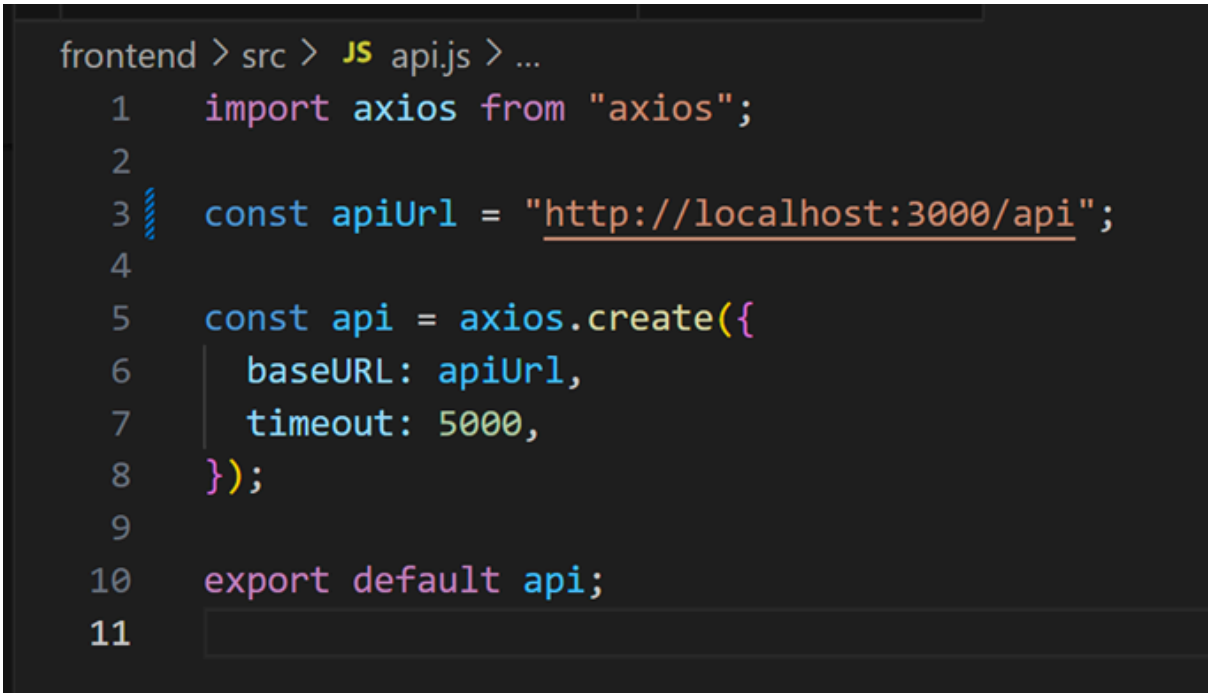


Рис. 2.10. api.js

Цей файл реалізується для того, щоб ініціалізувати загальні API запити, та містить імпортування axios, за допомогою якого ці запити здійснюються.

2.3.3. Розподілення функціоналу на компоненти

Один з найважливіших аспектів при створенні інтерфейсу є розділення на різні функціональні частини робочого коду, що власне спонукає до поділу користувацького інтерфейсу на менші, більш зручні для змін та коригування, блоки, які називаються компонентами.

Кожен з цих компонентів вміщує у собі власну структуру HTML, логіку коду JavaScript та стилі CSS, надаючи можливість впровадження покращень та змін до кожного з них, незалежних один від одного.

Ці розділені компоненти можуть бути, як простими кнопками або надписами, так і складними структурними сторінками, формами та макетними контейнерами.

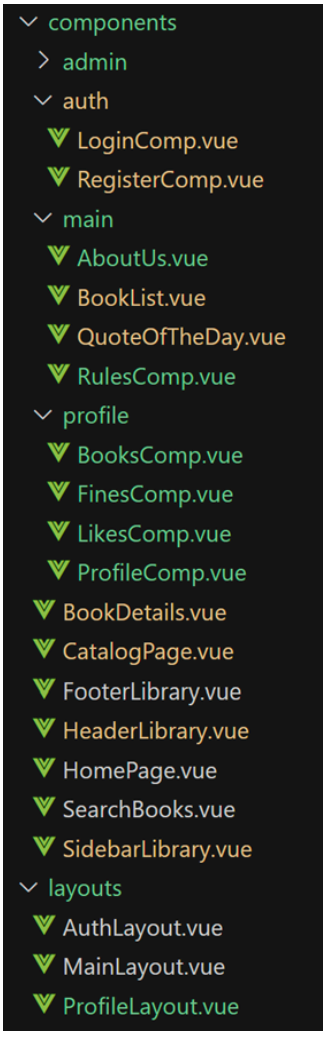


Рис. 2.11. Створені компоненти системи

Реалізовані частини коду можна описати таким чином:

- LoginComp.vue та RegisterComp.vue – відповідають за обробку аутентифікації користувачів, беручи макет з AuthLayout.vue.
- AboutUs.vue, BookList.vue, RulesComp.vue, BookDetails.vue, CatalogPage.vue – є загальними сторінками вебсайту, що виводять дані про всі книги, опис сайту та інші деталі, вони знаходяться в контейнері з структурою MainLayout.
- BooksComp.vue, FinesComp.vue, LikesComp.vue, ProfileComp.vue – відповідають за обробку даних в особистому профілі користувача, застосовуючи макет з ProfileComp.vue.

• QuoteOfTheDay.vue, HeaderLibrary.vue, FooterLibrary.vue, SidebarLibrary.vue – є допоміжними файлами, які необхідні для реалізації тільки частини сторінки. Деякі з них додатково використовуються для макетів, а інші одноразово тільки на одній сторінці.

Загалом обрана структура задає логічну, гарно організовану та гнучку систему, яка надає кожній обмеженій секції потрібні файли з можливістю розширень чи швидких внесень бажаних змін та доповнень. Даний підхід повністю відображає сучасні та найкращі методи з реалізації програмного забезпечення з точки зору користувача.

Висновок до розділу 2

Отже, щоб система мала зручний обмін даними між фронтендом та бекендом, було визначено головні інформаційні об’єкти такі, як: книги, користувачі, позики, бронювання, вподобання, жанри, повідомлення.

Для обміну даними використовувався REST API з методами HTTP, а саме: GET, POST, PUT, DELETE, таким чином можна було налаштувати доволі ефективну взаємодію клієнтської частини з сервером, і водночас, зберігати зрозумілу структуру запитів та відповідей у форматі JSON.

Архітектура фронтенду побудована на компонентному підході з чітким розподілом логіки на окремі модулі. Такий підхід забезпечує масштабованість системи, дозволяє ефективно обробляти зростаючі обсяги даних, додавати нові ресурси та повторно використовувати елементи інтерфейсу. Структура проєкту включає окремі папки для компонентів, маршрутизаторів, макетів сторінок, профілю користувача та логіки автентифікації.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Головними інструментами фронтенд розробки, які будуть використовуватись для побудови власної інформаційної системи, є: HTML, CSS, JavaScript. Кожен з цих інструментів відіграє важливу роль, так як вони відповідальні за визначення форми, стилізації та функціоналу для майбутнього вебсайту. [14]

Однією з ключових характеристик мови гіпертекстової розмітки HTML є її здатність формувати чітку та впорядковану структуру веб-сторінок шляхом розподілу контенту на логічні секції. Для цього використовуються теги, за допомогою яких задається загальний зміст документа — зокрема заголовки, списки, форми, зображення, абзаци та посилання. Особливо важливу роль відіграють семантичні теги, оскільки вони не лише покращують організацію контенту, а й полегшують навігацію для користувачів, зокрема для осіб з обмеженими можливостями. Окрім того, така мова розмітки надає доступ створеному вебсайту відображатись правильним чином на різних пристроях. Щодо головних переваг HTML, то можна виокремити наявний простий синтаксис, вона підтримується всіма основними веб-браузерами, а також, якщо застосовувати правильні теги й семантичну структуру, тоді буде покращуватись SEO інформаційної платформи, роблячи її більш видимою у пошукових системах.

CSS вважається таким інструментом, який відповідає за покращення візуальної привабливості системи. Абревіатура розшифровується як Cascading Style Sheets, що означає каскадні таблиці стилів, така мова забезпечує доцільну презентацію вебсторінок, власне опрацьовуючи такі аспекти як: налаштування шрифтів, кольорів, фонів, меж. CSS дозволяє керувати розташуванням, розмірами, а також

відступами між елементами на сторінці, допомагаючи створювати адаптивний дизайн для різних платформ, екранів.

Остання найбільш складна частина написання коду – використання динамічної мови JavaScript, за допомогою якої вебсайти стають інтерактивними та функціональними, наприклад, можна віднести до можливостей перевірку форми й оновлення змісту на сторінках. Спростити розробку системи дозволяє використання її фреймворку Vue.js.

Загалом дана мова описує такі процеси як: зміна елементів, обробка подій типу реакції на дії користувачів (кліки, рухи миші), виконання кількох операцій одночасно, таким чином підвищуючи продуктивність, і зрештою комунікація з сервером. Додатково, JavaScript сприяє забезпеченню зручності застосування сайту завдяки оновленням і переходами між сторінками, забезпечує стабільну роботу на різних пристроях, оскільки вона підтримується з усіма браузерами, а також дозволяє створювати додатки через легку інтеграцію з бібліотеками.

3.2. Реалізація функціоналу

Важливим компонентом ефективного вебсайту чи вебзастосунку є його якісно розроблений інтерфейс, власне це той аспект, завдяки якому формується початкове враження про ресурс. Якщо частина фронтенду системи буде опрацьована на високому рівні, тоді це сприятиме легкій навігації та забезпечить комфорт взаємодії. Під час реалізації функціоналу передбачена розробка такого інтерфейсу, що дозволить користувачам швидко орієнтуватися, розуміти структуру сторінки, фокусуватися на основному контенті та легко знаходити необхідні функції.[15]

Даних результатів можна досягти, застосовуючи сучасні принципи UI/UX дизайну, що включає: логічну побудову та візуальну ієрархію елементів, відповідність кольорової гами та типографіки. Недотримання таких аспектів, до прикладу, перевантаженість інформацією, поганий

контраст, невдале розміщення компонентів, – зазвичай спричиняє відмову від використання вебсайту. Таким чином, реалізація якісного інтерфейсу надасть можливість привернути увагу читачів та утримати її до моменту здійснення цільової дії, тобто конверсії.

3.2.1. Створення сторінок для автентифікації

Для виконання автентифікації було обрано підхід JSON Web Token (JWT) – це стандарт для створення веб-токенів, що можуть містити необов'язкові підписи або шифрування. Окрім того, вони передають дані у форматі JSON, які заявляють певні вимоги чи претензії, а також є ймовірність, що будуть підписані за допомогою приватного секрету або відкритого/приватного ключа для забезпечення їх цілісності.[16]

Створення автентифікації починається з підключення необхідних модулів.

```
1   const express = require("express");
2   const bcrypt = require("bcryptjs");
3   const jwt = require("jsonwebtoken");
4   const mysql = require("mysql2");
5
6   const app = express();
7   app.use(express.json());
```

Рис. 3.1. Підключення модулів

Після цього виконується звернення до ендпойнту, відповідального за реєстрацію користувача, де заповнюються поля, що відповідають структурі таблиці User.

```
18  app.post("/register", async (req, res) => {
19    const { username, password, email, firstName, lastName, phone } = req.body;
20
```

Рис. 3.2. Внесення полів

Однією з важливих перевірок при розробці платформ є така, що визначає, чи користувач вже реєструвався в системі через цю пошту або, чи логін з таким набором символів вже є створений. Це є дуже важливим етапом, що протидіє дублюванню та накопичуванню даних.

Тому в реєстрації важливо прописати перевірку, чи була вже пошта чи логін використані в системі.

```
22  db.query(
23    "SELECT * FROM Users WHERE Username = ? OR Email = ?",
24    [username, email],
25    async (err, results) => {
26      if (err) {
27        console.error("Помилка при перевірці існування користувача:", err);
28        return res.status(500).json({ message: "Помилка сервера" });
29      }
30
31      if (results.length > 0) {
32        console.log(
33          "Користувач із таким ім'ям користувача або електронною поштою вже існує."
34        );
35        return res.status(400).json({ message: "Користувач уже існує" });
36      }
```

Рис. 3.3. Пропис перевірки

Якщо такий користувач вже існує, то буде виведена помилка, яка нагадає про це людині, що входить в систему. За допомогою спеціального модуля в БД зашифровується пароль для кращої надійності збереження даних.

Після цього задається вхід для користувачів платформи. Це відбувається завдяки 2 полям: логіну та паролю.

```
60  app.post("/login", (req, res) => {
61    const { username, password } = req.body;
```

Рис. 3.4. Вхід для користувачів

Далі в базі даних триває пошук користувача з такою ж інформацією, якщо відповідних не знайдено або якщо на сервері стався збій у роботі, тоді виводяться помилки.

```
db.query(
  "SELECT * FROM Users WHERE Username = ?",
  [username],
  async (err, results) => {
    if (err) {
      console.error("Помилка при пошуку користувача:", err);
      return res.status(500).json({ message: "Помилка сервера" });
    }

    if (results.length === 0) {
      console.log("Користувача не знайдено:", username);
      return res.status(400).json({ message: "Користувача не знайдено" });
    }
```

Рис. 3.5. Пошук та показ помилок

Для перевірки тестування автентифікації використовуватиметься Postman.

Найпершим кроком буде введено кілька користувачів у системі.

Для цього потрібно вказати назву запущеного сервера, задати тип POST, у полі Body обрати raw – JSON та вписати дані реєстрації.

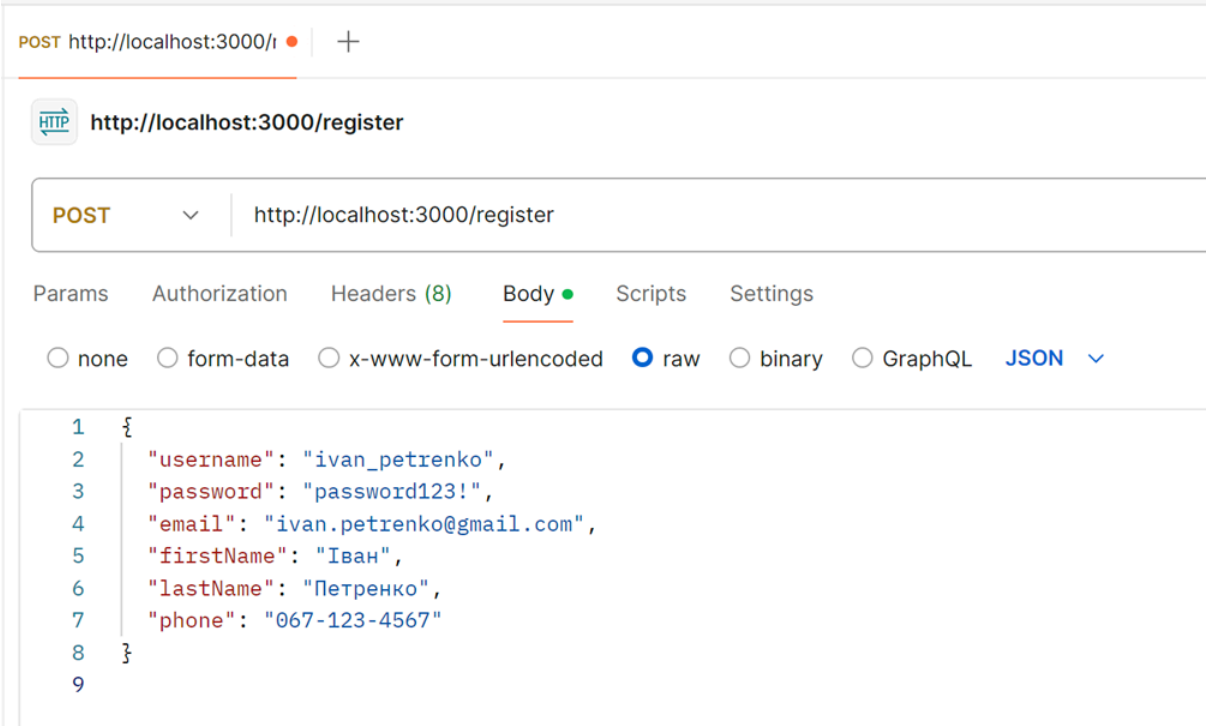


Рис. 3.6. Введення користувачів

Результат реєстрації виводиться одночасно знизу в Postman та в запущеному сервері.

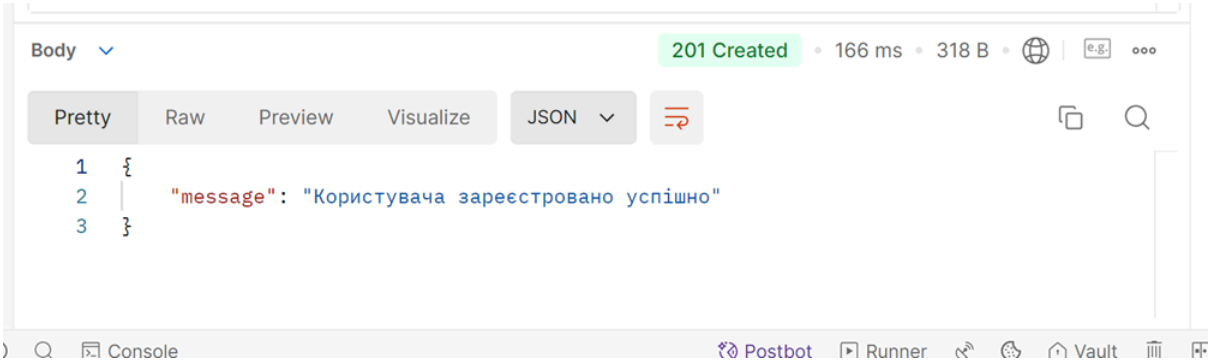


Рис. 3.7. Результат реєстрації

Для перевірки входу в систему використовується аналогічний підхід, як для реєстрації, але з новою назвою ендпойнту та дещо іншим JSON вмістом.

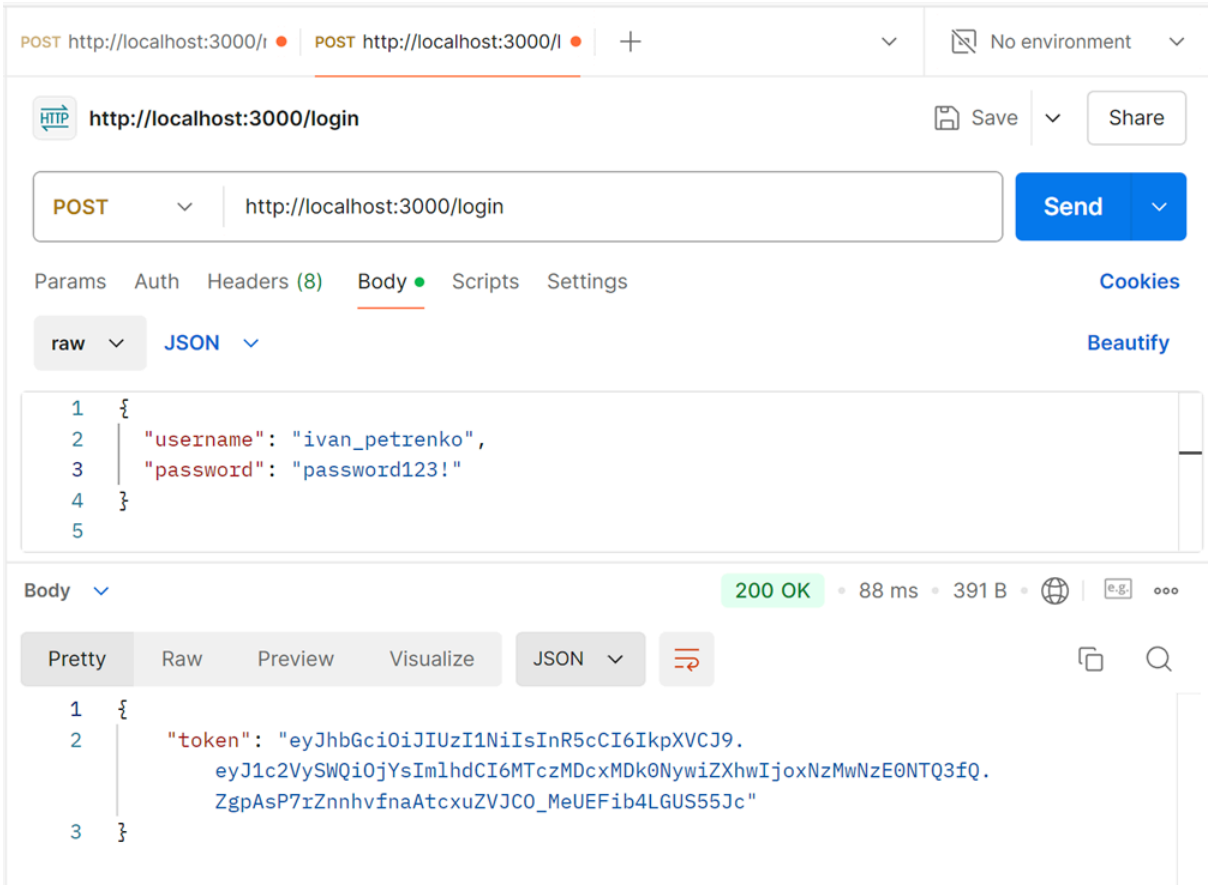


Рис. 3.8. Перевірка входу

З рисунку 3.8 з результатами перевірки можна побачити, що все працює коректно та згідно реалізованого коду з формуванням токену.

Після цього додатково створюються початкові сторінки, які допоможуть змоделювати роботу майбутніх користувачів системи.

Сторінка реєстрації виглядає наступним чином:

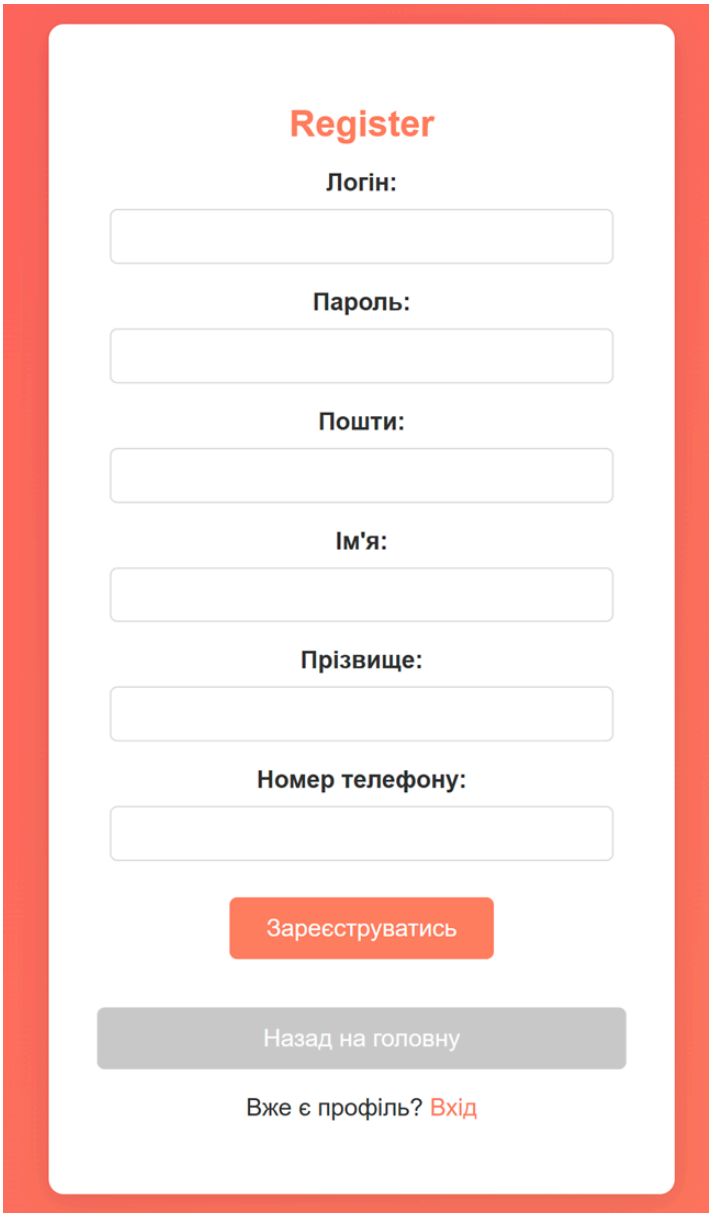


Рис. 3.9. Інтерфейс реєстрації

Нижче представлено інтерфейс сторінки авторизації.

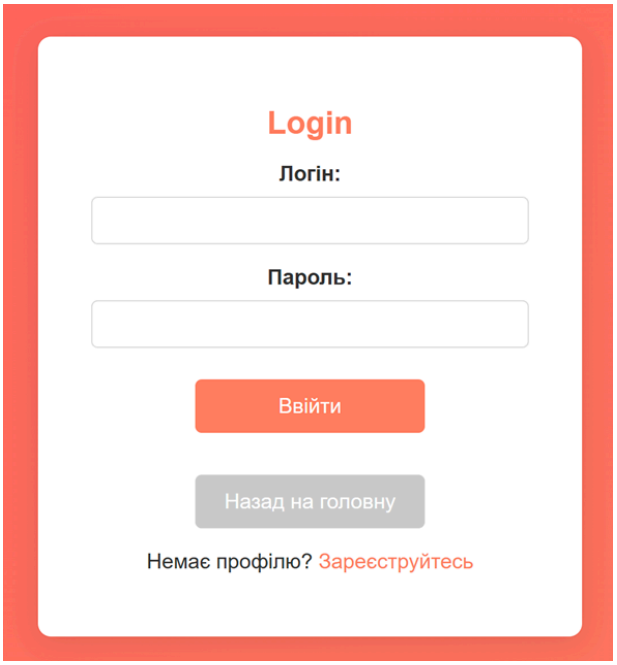


Рис. 3.10. Інтерфейс логіну

Таким чином, була реалізована автентифікація, яка надалі активно використовуватиметься для впровадження функціоналу та допоможе в розвитку проєкту для повноцінної системи.

3.2.2. Реалізація пошуку

Для реалізації пошуку використовується компонентний підхід, що був використаний у створенні всього вебсайту.

Першим кроком під час реалізації є створення коду серверної частини. Далі за сформованим books/search та записаним посиланням формується обробка цих даних на стороні клієнтського серверу. Для того, щоб його запустити, необхідно ввести початкову інформацію до поля з пошуком.

Після цього йде реалізація фронтенд частини, що надасть структуру цим елементам. У цьому коді реалізується створення елементів, що беруть участь під час пошуку. Поле для введення, що приймає значення для обробки результатів та виведення списку книг у вигляді сформованих

посилань, ведуть на детальну сторінку книги та обробки помилки з відображенням повідомлення “Нічого не знайдено”.

```
<template>
  <div class="search-header" ref="searchHeader">
    <input
      v-model="query"
      type="text"
      placeholder="Пошук книг..."
      @input="handleSearch"
      @focus="showResults = true"
      class="search-input"
    />
    <div v-if="showResults && books.length" class="search-results">
      <ul>
        <li v-for="book in books" :key="book.bookid">
          <a :href="`/book/${book.bookid}`">{{ book.title }}</a>
        </li>
      </ul>
    </div>
    <div v-else-if="showResults && !books.length" class="no-results">
      <p>Нічого не знайдено</p>
    </div>
  </div>
</template>
```

Рис. 3.11. Формування інтерфейсу

Останнім кроком є додавання стилів до цих елементів.

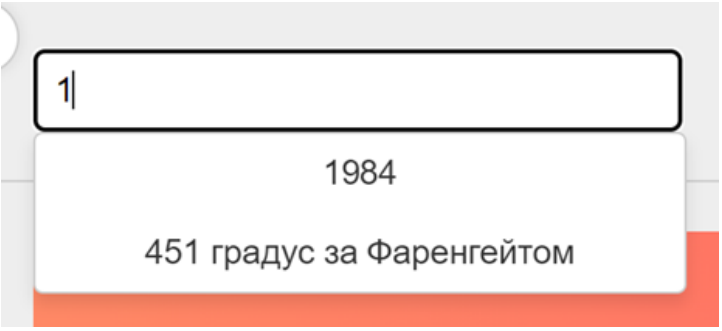
Для проведення оптимізації буде внесено деякі зміни до коду, наприклад, це використання handleSearch() через дебаунсинг (debounce), щоб не викликати кожного разу запит до сервера, а очікувати певний час, коли користувач завершить написання ключових слів. [17]

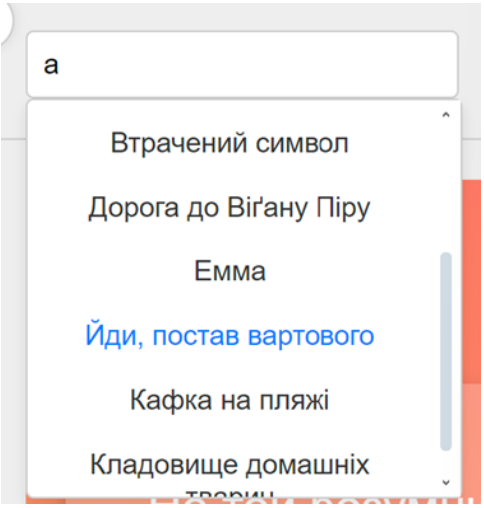
```
this.debounceTimeout = setTimeout(async () => {
  if (this.query.trim().length === 0) {
    this.books = [];
    this.showResults = false;
    return;
  }

  if (this.cache[this.query]) {
    this.books = this.cache[this.query];
    this.showResults = this.books.length > 0;
    return;
  }
```

Рис. 3.12. Внесення дебаунсингу

Також додатково було прописано використання кешу, що буде перевірятись, щоб не запитувати сервер ті ж запити.

Після запуску роботи серверів проводиться тестування реалізованого пошуку.





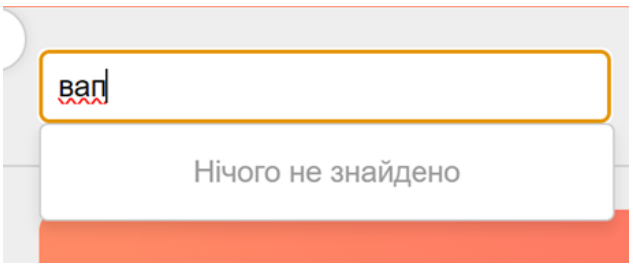


Рис. 3.13-15. Тестування роботи пошуку

3.2.3. Створення сторінок для перегляду книги та каталогу

Наступним етапом є проєктування основних сторінок, які часто використовуються в системі, через те, що перегляд основної інформації про книгу є головною причиною при виборі для читання.

Щоб правильно створити цю структуру, необхідно притримуватись такої, що вже була задана раніше. Першим кроком є побудова порожніх файлів-компонентів у правильних папках зберігання та винесення їх у файлі router:

```
19  import CatalogPage from "@/components/CatalogPage.vue";
20  import BookDetails from "@/components/BookDetails.vue";
21
22  const routes = [
23    {
24      path: "/",
25      component: MainLayout,
26      children: [
27        {
28          path: "/catalog",
29          name: "Catalog",
30          component: CatalogPage,
31          meta: { title: "Каталог - Reread" },
32        },
33        {
34          path: "/book/:id",
35          name: "BookDetails",
36          component: BookDetails,
37          meta: { title: "Книга - Reread" },
38        },
```

Рис. 3.16. Налаштування маршрутів нових сторінок

Звідси можна помітити, що ці сторінки є дочірними від створеного батьківського компоненту MainLayout.

Це зроблено, через те що нові сторінки будуть огортатись всередину нього та належать структурі цього лейауту. Дане рішення покращує та спрощує користування сайту, зменшуючи постійне дублювання коду, надаючи користувачам доступ до бічної та верхньої панелі при застосуванні цих сторінок.

Далі побудова сторінок доволі схожа, у Vue елементі є 3 частини: template, script, styles, що визначають структуру, код та стилі, які описуються в компоненті.

Для виведення особистої сторінки необхідно скористатись функцією, що буде відображати дані з серверу та далі переносити їх на HTML- структуру.

```
async fetchBook() {
  this.loading = true;
  this.error = null;
  const bookId = this.$route.params.id;

  try {
    const response = await api.get(`/books/id/${bookId}`);
    this.book = response.data;
  } catch (err) {
    this.error = "Не вдалося завантажити дані про книгу.";
    console.error(err);
  } finally {
    this.loading = false;
  }
},
```

```
<div class="book-info">
  <h1>{{ book.book_title }}</h1>
  <p class="author">
    <strong>Автор:</strong> {{ book.author_firstname }}
    {{ book.author_lastname }}
  </p>
  <p><strong>Жанр:</strong> {{ book.genre_name }}</p>
  <p><strong>Рік:</strong> {{ book.year }}</p>
  <p><strong>ISBN:</strong> {{ book.isbn }}</p>
  <p class="description">{{ book.book_description }}</p>
```

Рис. 3.17-18. Обробка та виведення даних на сторінку

Для того, щоб переглянути створені сторінки, необхідно перейти за правильними маршрутами. Для особистої сторінки – /book/:id, де id – це власний ідентифікатор книги, а для каталогу /catalog.

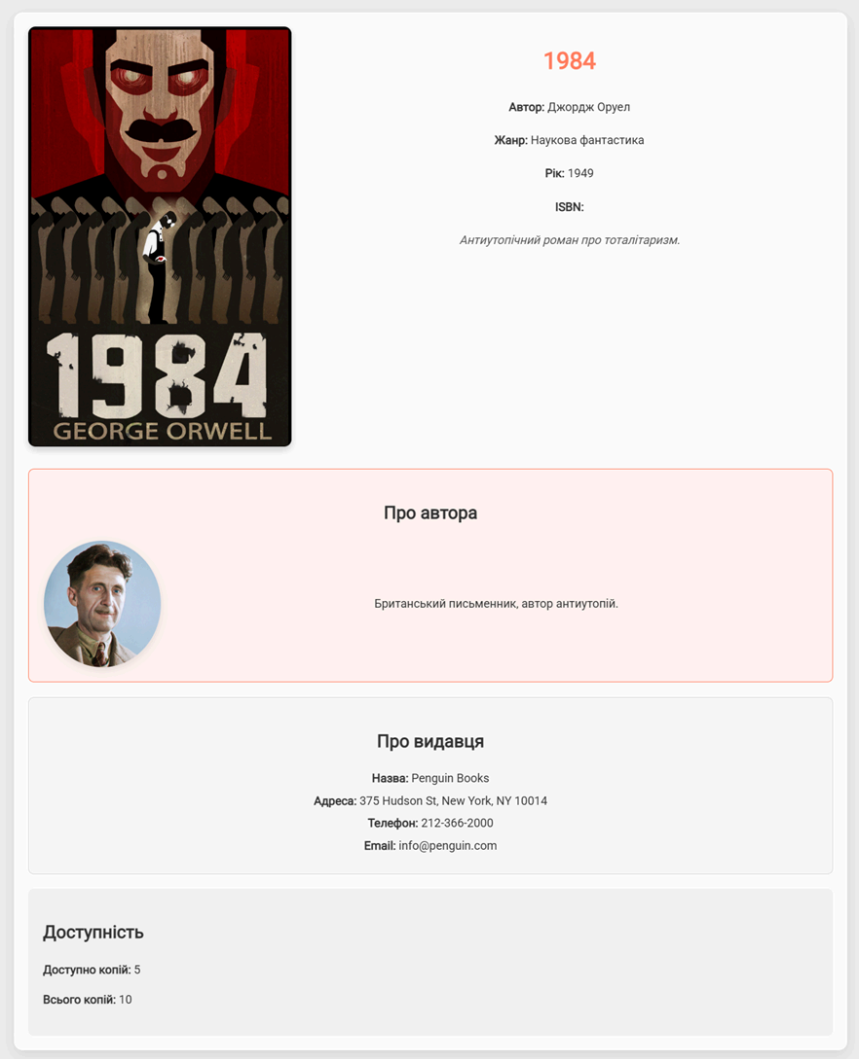


Рис. 3.19-20. Виведення даних книги з id 1 та третього каталогу

3.2.4. Модуль профілю користувача

Для того, щоб організувати дані читачів у системі, необхідно вивести їх на окремі сторінки, що допоможуть сховати цю інформацію від загального бачення та відображати її тільки в особистому кабінеті для поточного користувача.

Для організації цього процесу створюється окрема папка, в якій розміщуються необхідні файли, призначені для виведення потрібних даних у відповідних компонентах.

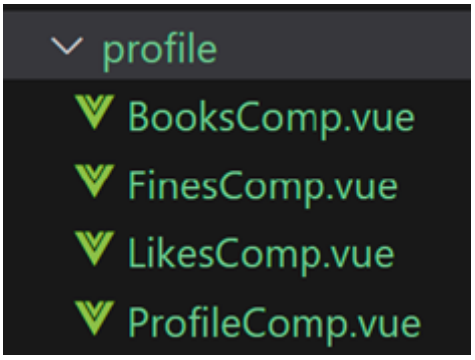


Рис. 3.21. Створені компоненти профілю

Далі в router необхідно задати правильні маршрути, що будуть задіяні для переходу між цими сторінками та прив’язуватись до бажаних компонентів. Додатково, вказується особиста назва кожного компоненту.

Також можна побачити, що ці компоненти додатково огорнуті в ProfileLayout, що дозволить успішно перемикатись між цими сторінками.

```
{
  path: "/profile",
  component: ProfileLayout,
  children: [
    {…
    },
    {
      path: "info",
      component: ProfileComp,
      meta: { title: "Профіль - Reread" },
    },
    {
      path: "likes",
      component: LikesComp,
      meta: { title: "Уподобання - Reread" },
    },
    {
      path: "books_shelf",
      component: BooksComp,
      meta: { title: "Моя книжкова полиця - Reread" },
    },
    {
      path: "fines",
      component: FinesComp,
      meta: { title: "Штрафи - Reread" },
    },
    { path: "", redirect: "/profile/info" },
  ],
  meta: { requiresAuth: true },
},
```

Рис. 3.22. Маршрути сторінок створених для профілю

Далі до кожного прописується відповідний код, що виконуватиме різний функціонал додатку. Після чого необхідно задати деталі до ProfileComp, вивівши в нього всі дані профілю з можливістю їх редагування.

Для реалізації цього завдання необхідно вивести кожне поле у вигляді окремого елемента input. Це здійснюється шляхом підключення до маршруту бекенду /api/user/me, який за допомогою персонального токена надає доступ до даних конкретного користувача. Отримана інформація автоматично підставляється у відповідні поля форми.

```vue
<form @submit.prevent="updateProfile">
  <div class="form-group">
    <label for="username">Логін:</label>
    <input
      v-model="user.username"
      type="text"
      id="username"
      disabled
      placeholder="Логін (не можна змінити)"
    />
  </div>
```

```
async fetchUserProfile() {
  try {
    const token = localStorage.getItem("token");
    const response = await axios.get("http://localhost:3000/api/user/me", {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    this.user = response.data;
  } catch (error) {
    console.error("Помилка при отриманні профілю", error);
  }
},
```

Рис. 3.23-24. Виведення даних профілю на сторінку

Останнім кроком є налаштування оновлення, що задається завдяки /api/user/update-profile, що теж за допомогою токену визначає користувача, який змінює дані та зберігає їх до бази даних.

```
async updateProfile() {
  try {
    const token = localStorage.getItem("token");
    const response = await axios.put(
      "http://localhost:3000/api/user/update-profile",
      this.user,
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );
    console.log(response);
```

Рис. 3.25. Налаштування оновлення профілю

3.2.5. Функціонал для вподобань та бронювання книг

Для реалізації цього функціоналу використовуються два компоненти: LikesComp і BooksComp.

Наповнення template повторює схожі компоненти, де було проведено виведення даних про книги, проте розбіжності приходять при реалізації елементу script. Тут для вподобань та бронювань використовуються окремі маршрути, що показують тільки ті книги, які належать до цих груп.

```
async fetchLikes() {
  this.loading = true;
  this.error = null;
  try {
    const token = localStorage.getItem("token");
    const response = await axios.get("http://localhost:3000/api/likes", {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    this.likes = response.data;
  } catch (err) {
    this.error = err.response?.data?.message || "Помилка при завантаженні";
  } finally {
    this.loading = false;
  }
},
```

```
async fetchReservations() {
  this.loading = true;
  this.error = null;
  try {
    const token = localStorage.getItem("token");
    const response = await axios.get(
      "http://localhost:3000/api/reservations",
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );
    this.reservations = response.data;
  } catch (err) {
    this.error = err.response?.data?.message || "Помилка при завантаженні";
  } finally {
    this.loading = false;
  }
},
```

Рис. 3.26-27. Виведення вподобаних та забронюваних книг

Далі розбіжності продовжуються тим, що в компоненті з вподобаннями знаходиться можливість видалення лайку, якщо книга була помилково доданою або при зміні думки користувача та перенесення книги на сторінку з бронюваннями. Та функція реалізовується завдяки відправленню інших маршрутів, що використовують більш інтерактивні методи delete та post.

```
async removeLike(likeid) {
  const likedBook = this.likes.find((like) => like.likeid === likeid);
  const confirm = await Swal.fire({···
  });

  if (confirm.isConfirmed) {
    try {
      await axios.delete(`http://localhost:3000/api/likes/${likeid}`, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem("token")}`,
        },
      });
      this.likes = this.likes.filter((like) => like.likeid !== likeid);
```

```
121     async createReservation(bookId) {
122       const selectedBook = this.likes.find((like) => like.bookid === bookId);
123
124 >     const result = await Swal.fire({ ···
133       });
134
135       if (result.isConfirmed) {
136         try {
137           const token = localStorage.getItem("token");
138           await axios.post(
139             "http://localhost:3000/api/reservations",
140             {
141               bookId,
142               status: "active",
143               loandate: new Date().toISOString().split("T")[0],
144               dueDate: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000)
145                 .toISOString()
146                 .split("T")[0],
147             },
148             {
149               headers: {
150                 Authorization: `Bearer ${token}`,
151               },
152             }
153           );
```

Рис. 3.28-29. Створення функцій removeLike та createReservation

Ці дві функції є описані в компоненти вподобань, тоді як сторінка бронювань містить вже раніше розроблений fetchReservations, за допомогою якого виводяться книги, які користувач додав до бронювань та cancelReservation, що надає можливість повної відміни з додатковим візуальним редагуванням елементу.

```
async cancelReservation(reservationId, copyId) {
  try {
    const token = localStorage.getItem("token");
    const response = await axios.patch(
      `http://localhost:3000/api/reservations/${reservationId}`,
      { copyId },
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );

    if (response.status === 200) {
      this.reservations = this.reservations.map((reservation) =>
        reservation.reservationid === reservationId
          ? { ...reservation, status: "cancelled" }
          : reservation
      );
      Swal.fire( ···
      );
```

Рис. 3.30. Функція cancelReservation

Вигляд реалізованих функцій буде показаний у наступних підрозділах та додатках.

3.3. Керівництво користувача

Для того, щоб локально переконатись в роботі додатку, необхідно пройти кілька кроків.

Один із способів взаємодії забезпечує доступ до всього коду вебсайту та налаштування програмних застосунків. Необхідно завантажити Node та PostgreSQL, щоб коректно проводити запуски. Після успішного завантаження та перевірки їх роботи, можна перейти до наступного кроку.

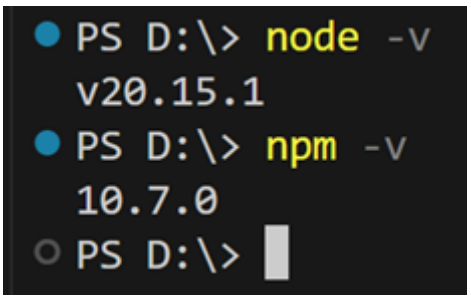


Рис. 3.31. Перевірка роботи Node

Тут необхідно перейти до терміналу та запустити команди для роботи відповідних серверів, але спочатку потрібно налаштувати залежності, що застосовуються в додатку. Загалом, під час розробки вони автоматично вказуються у файлі package.json.

```
10  "dependencies": {
11    "@splidejs/vue-splide": "^0.6.12",
12    "axios": "^1.7.9",
13    "chart.js": "^4.4.9",
14    "core-js": "^3.8.3",
15    "jwt-decode": "^4.0.0",
16    "recharts": "^2.15.2",
17    "sweetalert2": "^11.17.2",
18    "swiper": "^11.2.1",
19    "vue": "^3.2.13",
20    "vue-chartjs": "^5.3.2",
21    "vue-router": "^4.5.0",
22    "vuex": "^4.1.0"
23  },
```

Рис. 3.32. Залежності фронтенду

А для того, щоб запустити їх у середовищі або користувачем, що не розробляв поступово вебсайт, потрібно ввести команду npm install, що встановить розширення, які були описані вище.

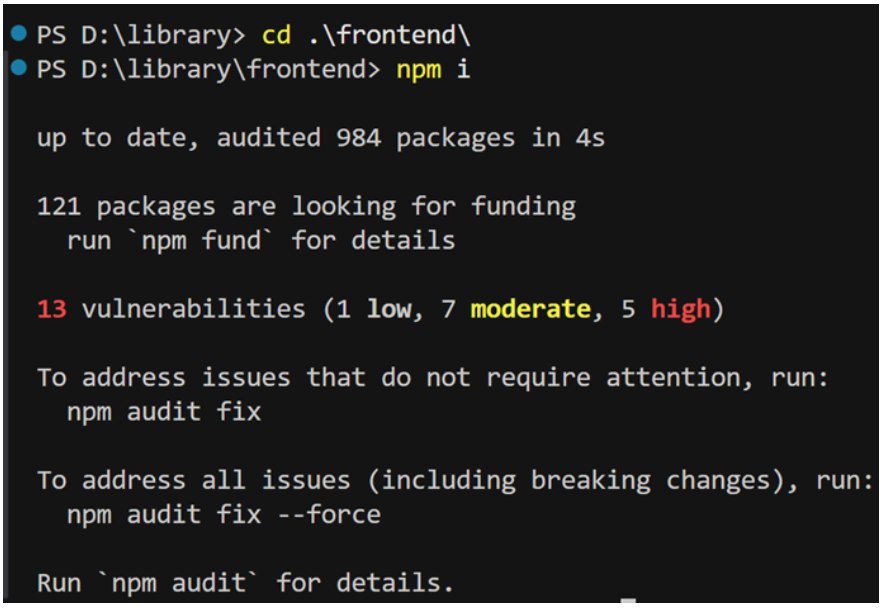


Рис. 3.33. Встановлення залежностей

Ця команда додасть нову папку node_modules, яка зберігає дані розширення, але їх необхідно використовувати тільки локально. При деплою вони не надсилаються та не застосовуються.

Останнім кроком для запуску є введення команди npm run serve, що успішно запустить клієнтську частину додатку.

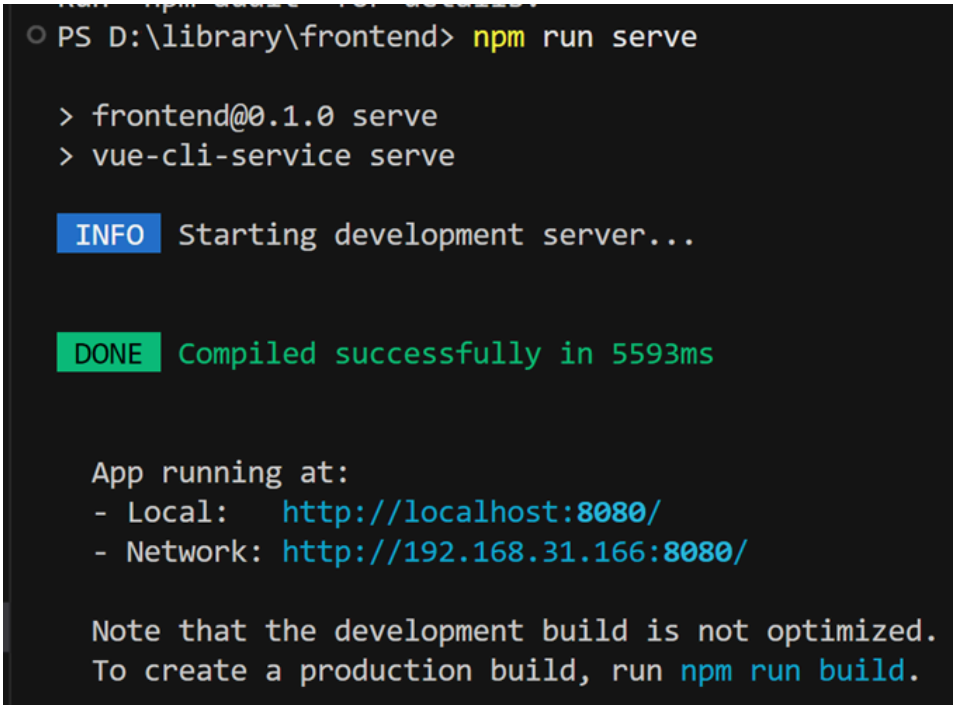


Рис. 3.34. Запущений сервер фронтенду

Для того, щоб організувати повноцінну роботу сайту, необхідно зробити такі ж кроки для інсталяції залежності в папці backend, але командою запуску слугуватиме npm run start.

3.4. Тестування інтерфейсу

Проведення тестування реалізованих сторінок клієнтської частини сайту виконується, щоб перевірити чи правильно застосовується кожен з компонентів та описана в них логіка. Для цього буде використаний ручний метод.

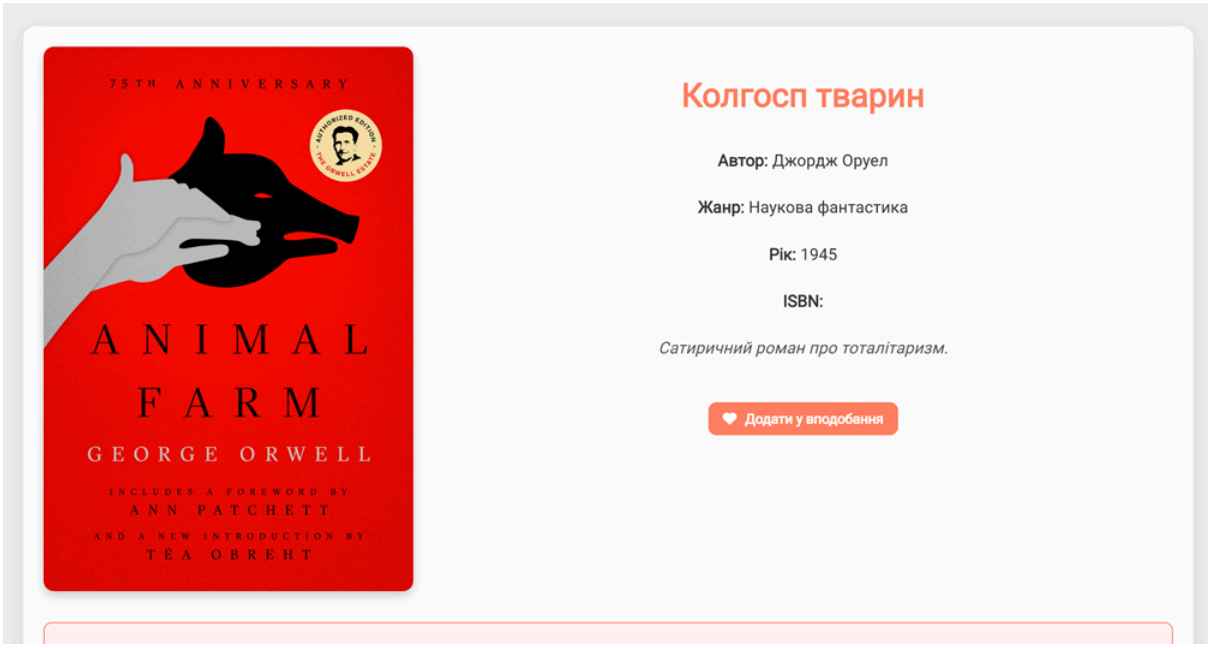
Спочатку перевірятиметься виведення книг та їх уподобання для користувачів системи. Щоб це виконати потрібно авторизуватись в профілі, далі буде застосовано тест-кейси, які необхідно перевірити.

Табл.3.1. Перевірка роботи сторінки книги

№	Назва тестування	Дії користувача	Очікуваний результат	Статус тесту
1	Вхід до профілю	Внесення логіну та паролю	Якщо користувач заблокований, то доступ заборонений	Пройдено
			Успішний вхід, якщо користувач не заблокований	Пройдено
2	Перегляд деталей	Перехід на сторінку книги	Якщо користувач авторизований, то доступна кнопка вподобання	Пройдено

			Якщо користувач не пройшов авторизацію, кнопка вподобання не виводиться	Пройдено
3	Вподобання	Натискання кнопки вподобань	Якщо книги немає у вподобаних, відображається повідомлення про успіх	Пройдено
			Якщо книга присутня, сповіщення про те, що книга вже додана	Пройдено

Цей процес буде налаштований наступним чином: якщо користувач успішно автентифікований, тоді вподобання є доступним та демонструється сповіщення, що все пройдено правильно, інші результати тестування показані в додатках.



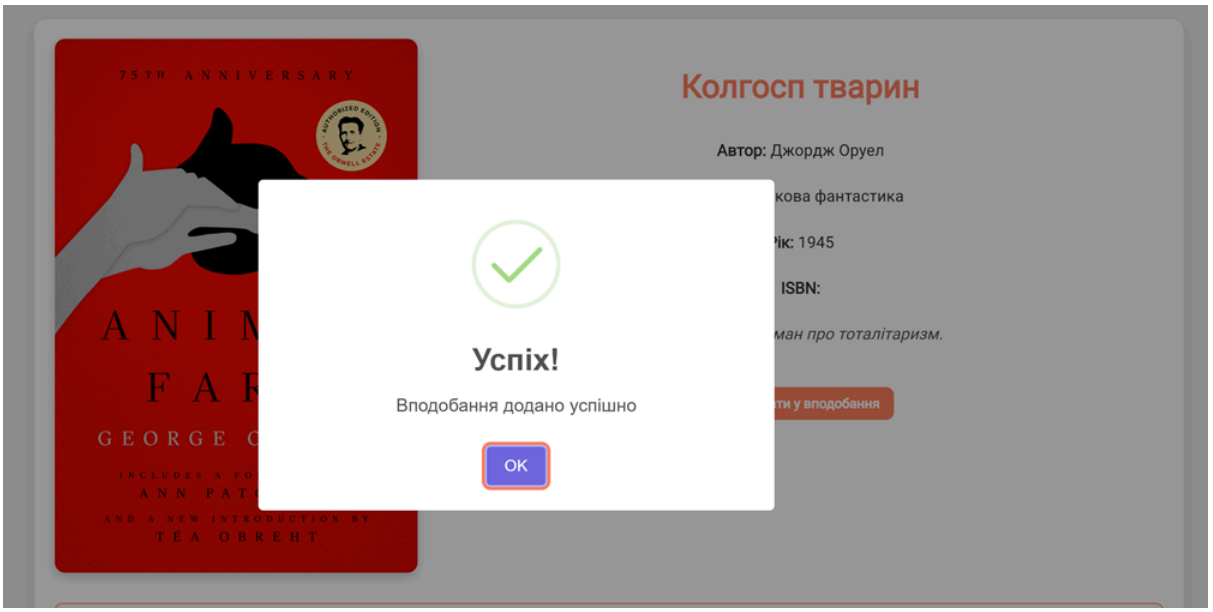


Рис. 3.35-36. Додавання вподобаних книг на сторінці

Далі необхідно перевірити роботу особистого профілю користувача.

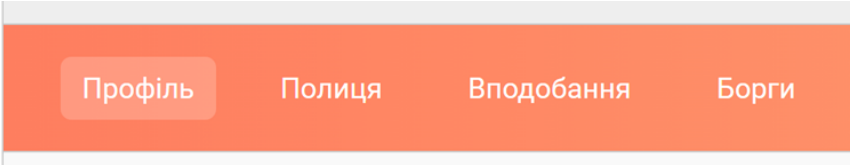
Табл.3.2. Перевірка профілю користувача

№	Назва тестування	Дії користувача	Очікуваний результат	Статус тесту

1	Вхід до профілю	Внесення логіну та паролю	Якщо користувач заблокований, то доступ заборонений	Пройдено
			Успішний вхід, якщо користувач не заблокований	Пройдено
2	Перегляд сторінок	Перехід між сторінками профілю	Система правильно відображає вміст кожної сторінки, вивівши необхідні дані для користувача	Пройдено
3	Видалення вподобання	Натискання кнопки видалення вподобань	Система сповіщає повідомленням, чи користувач впевнений у своєму рішенні, і при підтвердженні успішно видаляє книгу з вкладки вподобаних	Пройдено
4	Бронювання книги	Натискання кнопки забронювати серед вподобаних книг	Система сповіщає повідомленням, чи користувач підтверджує бронювання, і при підтвердженні успішно додає книгу до вкладки Полиця	Пройдено

5	Сортування сторінки Полиця	Перемикання між статусами бронювань	При перемиканні між необхідними статусами, система правильно змінює результати сортувань	Пройдено
6	Скасування бронювання	Натискання кнопки скасувати бронювання	Система сповіщає повідомленням, чи користувач підтверджує скасування бронювання, і після згоди змінює колір бронювання на червоний	Пройдено

Інтерфейс відповідно до цих тест-кейсів матиме наступний вигляд.



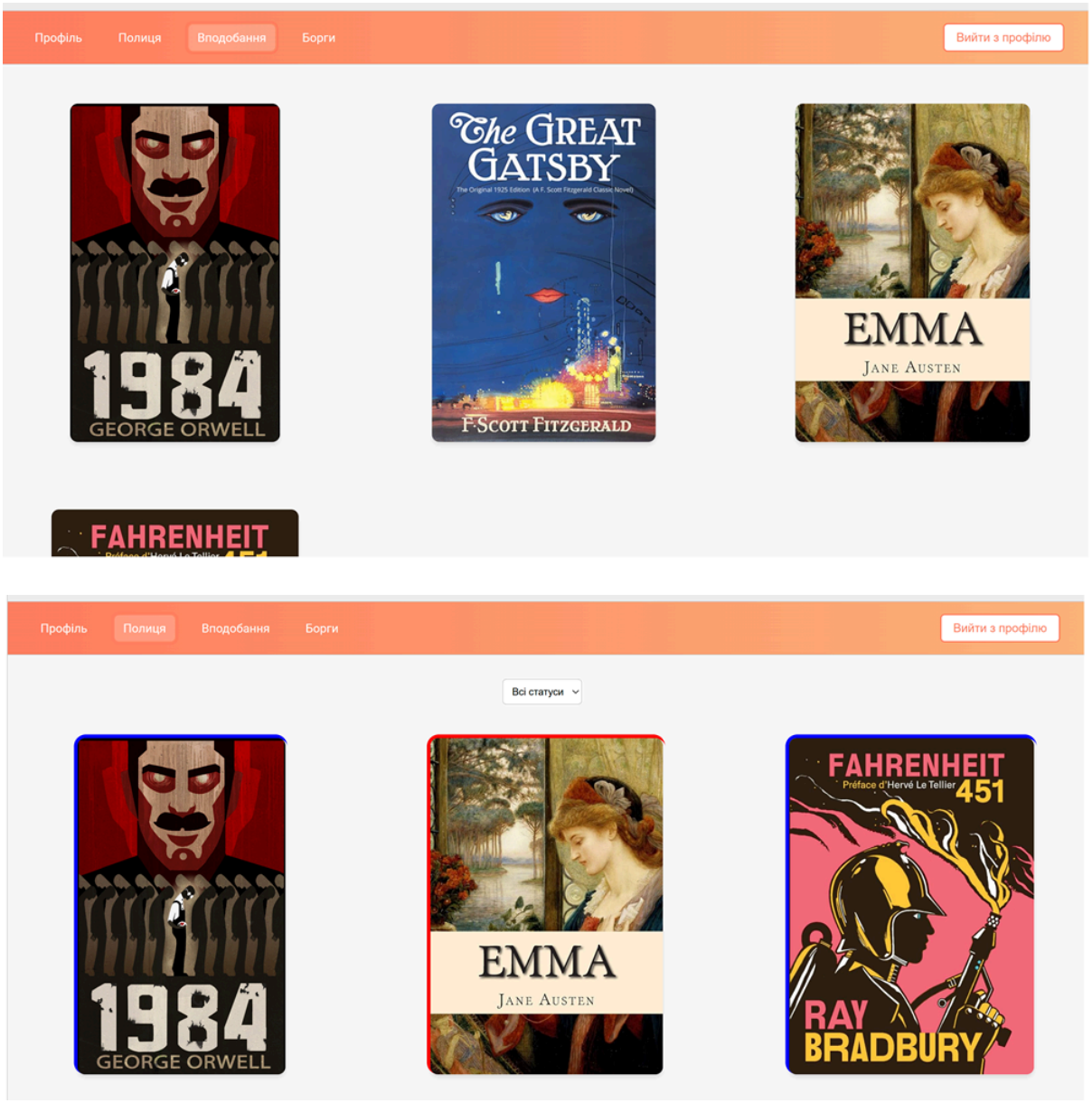


Рис. 3.37-39. Виведення сторінок профілю

Решту сторінок та інтерактивних деталей наведено в додатках.

Додатково, виведення сторінок було перевірено на доступних браузерах: Opera, Microsoft Edge, Google Chrome, Firefox.

Висновок до розділу 3

На останньому етапі розробки інформаційної системи було досягнуто таких результатів:

- Реалізовано реєстрацію та вхід, які перевіряють через базу даних, таким чином, щоб було надано безпечне збереження паролів та водночас не повторювались користувачі;
- Створені інтерфейси, що синхронізовані із серверною частиною;
- Реалізовано функції: пошук книг, перегляд каталогу й матеріалів, залишення вподобань та бронювання літератури;
- Розроблено профіль користувача;
- Проведено тестування інтерфейсів, включаючи етап перевірки роботи на різних браузерах.

Дана система працює коректно щодо всіх реалізованих функцій, включаючи аспекти обробки помилок, повідомлень, реакцій на дії користувача. Вебсайт демонструє стабільну роботу на різних платформах, що робить його придатним як для навчальних цілей, так і для подальшого розвитку. Фронтенд побудовано таким чином, що він легко масштабується, дозволяє додавати нові функції та інтегруватися з іншими пристроями.

ВИСНОВКИ

У результаті проведеного дослідження щодо розробки програмного забезпечення, призначеного для користування електронною бібліотекою, було досягнуто головної мети - створити інформаційну систему з інтуїтивно зрозумілим дизайном та широким функціональним набором. Для цього було поступово виконано ряд завдань:

- Детально проаналізовано предметну область і, з ціллю логічної побудови платформи, попередньо було виокремлено ряд задач для поетапного розв’язання;
- Оглянуто існуючі рішення на ринку та визначено їхні переваги й недоліки, для включення кращих аспектів до власної розробки і уникання тих чинників, що ускладнюють взаємодію;
- Описано інформаційні об’єкти системи, завдяки чому можна було забезпечити правильний обмін даними між клієнтською та серверною частинами, а також перегляд і редагування інформації;
- Реалізовано взаємодію з діючим API для того, щоб надати доступ до серверної частини, визначено архітектурні рішення для клієнтської частини. Систему побудовано на основі компонентної структури з використанням засобів: JavaScript, CSS і HTML;
- Розроблено функціонал: автентифікації користувачів із застосуванням JWT, уподобання та бронювання книг;
- Реалізовано сторінки для пошуку літератури, що працює на базі компонентного підходу, та для перегляду книг, а також каталогу;
- Створено інтерфейси для реєстрації й логіну;
- Розроблено профіль користувача, де можна редагувати особисту інформацію;
- Проведено тестування інтерфейсів, під час якого було підтверджено коректність розробки, включаючи успішну перевірку на різних браузерах.

У цьому проєкті власним внеском є детальне проєктування та реалізація фронтенд частини інформаційної системи, власне акцентуючи увагу на простому, сучасному, зрозумілому інтерфейсі, а також забезпечуючи інтерактивність й зручність взаємодії з системою. Дані результати можуть бути використані для подальшого розвитку подібних вебзастосунків у навчальних, культурних та освітніх установах, сприяючи підвищенню доступності цифрових ресурсів.

Отже, створення інформаційної системи для бібліотеки дозволить автоматизувати управління книгами, користувачами, що відповідно суттєво підвищить ефективність її функціонування. Головними можливостями такої платформи виступає зменшення ризику появи будь-яких помилок, спрощення керування наявними ресурсами, полегшення виконання адміністративних процесів та забезпечення надійного зберігання інформації. Використання вебсайту істотно підвищить зручність доступу до бібліотечних послуг, оскільки користувачі матимуть змогу оперативно знаходити необхідні книги та оформлювати їх отримання, що сприятиме економії часу та зменшенню загальних витрат. Більш того, впровадження інформаційної системи дозволить суттєво знизити навантаження на персонал установи й, відповідно, підвищити якість обслуговування відвідувачів. Важливою складовою є продуманий фронтенд, адже завдяки зручному та інтуїтивно зрозумілому інтерфейсу майбутній застосунок стане доступним для широкого кола користувачів, що, у свою чергу, наблизить бібліотеку до досягнення її основної мети — популяризації читання та підвищення зацікавленості суспільства до літератури.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Library Technology Guides: Key Resources in Library Automation. [Електронний ресурс]. – Режим доступу: https://librarytechnology.org/ (дата звернення: 01.05.2025).

2. Importance of Libraries: 10 Reasons – Bibliolifestyle. [Електронний ресурс]. – Режим доступу: https://bibliolifestyle.com/importance-of-libraries-10-reasons/ (дата звернення: 01.05.2025).

3. What Libraries Do – I Love Libraries. [Електронний ресурс]. – Режим доступу: https://ilovelibraries.org/what-libraries-do/ (дата звернення: 01.05.2025).

4. Резніченко В.А., Захарова О.В., Захарова Е.Г. Електронні бібліотеки: інформаційні ресурси та сервіси. [Електронний ресурс] // Проблеми програмування. – 2005 – С. 5. – Режим доступу: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/1374/05-%20Резнко%233.pdf?sequence=1 (дата звернення: 01.05.2025).

5. Frontend і Backend: дві сторони веброзробки – Voll. [Електронний ресурс]. – Режим доступу: https://voll.com.ua/uk/blog/frontend-i-bekend-dve-storony-odnoj-medali (дата звернення: 01.05.2025).

6. Електронна бібліотека – Чтиво. [Електронний ресурс]. – Режим доступу: https://chtyvo.org.ua (дата звернення: 01.05.2025).

7. Онлайн бібліотека книг українською – Knigogo. [Електронний ресурс]. – Режим доступу: https://knigogo.top (дата звернення: 01.05.2025).

8. Українська електронна бібліотека – Libruk. [Електронний ресурс]. – Режим доступу: https://libruk.com.ua (дата звернення: 01.05.2025).

9. Електронна бібліотека – НЮУ. [Електронний ресурс]. – Режим доступу: https://elib.nlu.org.ua (дата звернення: 01.05.2025).

10. Національна бібліотека України імені В. І. Вернадського. [Електронний ресурс]. – Режим доступу: http://www.nbuv.gov.ua (дата звернення: 01.05.2025).

11. Wat zijn API’s? – ITpedia. [Електронний ресурс]. – Режим доступу: https://uk.itpedia.nl/2018/11/02/wat-zijn-apis-application-programming-interface/ (дата звернення: 01.05.2025).

12. What Is REST API, SOAP, GraphQL, WebSockets and RPC? – Fineproxy. [Електронний ресурс]. – Режим доступу: https://fineproxy.org/what-is-rest-api-soap-graphql-websockets-and-rpc/ (дата звернення: 01.05.2025).

13. HTTP Methods Explained – Apidog. [Електронний ресурс]. – Режим доступу: https://apidog.com/blog/http-methods/ (дата звернення: 01.05.2025).

14. The Importance of HTML, CSS and JavaScript in Web Development – Moldstud. [Електронний ресурс]. – Режим доступу: https://moldstud.com/articles/p-the-importance-of-html-css-and-javascript-in-web-development (дата звернення: 01.05.2025).

15. Web Components. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_components (дата звернення: 01.05.2025).

16. Authentication Tokens: JSON Web Tokens Explained – QATestLab. [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/authentication-tokens/ (дата звернення: 01.05.2025).

17. Implementing Debouncing in Vue. [Електронний ресурс]. – Режим доступу: https://dev.to/jacobandrewsky/implementing-debouncing-in-vue-22jb (дата звернення: 01.05.2025).

ДОДАТКИ

Додаток А. Код файлу router

```
import { createRouter, createWebHashHistory } from "vue-router";

import store from "@/store";

import MainLayout from "@/layouts/MainLayout.vue";

import AuthLayout from "@/layouts/AuthLayout.vue";

import ProfileLayout from "@/layouts/ProfileLayout.vue";



import AboutUs from "@/components/main/AboutUs.vue";

import RulesComp from "@/components/main/RulesComp.vue";

import HomePage from "@/components/HomePage.vue";

import CatalogPage from "@/components/CatalogPage.vue";

import BookDetails from "@/components/BookDetails.vue";

import LoginPage from "@/components/auth/LoginComp.vue";

import RegisterPage from "@/components/auth/RegisterComp.vue";

import ProfileComp from "@/components/profile/ProfileComp.vue";

import LikesComp from "@/components/profile/LikesComp.vue";

import BooksComp from "@/components/profile/BooksComp.vue";

import FinesComp from "@/components/profile/FinesComp.vue";
```

Продовження додатку А

```
import AdminPanel from "@/components/admin/AdminPanel.vue";


const routes = [
  {
    path: "/",
    component: MainLayout,
    children: [
      {
      path: "",
      name: "Home",
      component: HomePage,
      meta: { title: "Головна - Reread" },
      },
      {
      path: "about",
      name: "AboutUs",
       component: AboutUs,
```

Продовження додатку А

```
    meta: { title: "Про нас - Reread" },

    },

    {

    path: "rules",

    name: "Rules",

    component: RulesComp,

    meta: { title: "Правила - Reread" },

    },

    {

    path: "/catalog",

    name: "Catalog",

    component: CatalogPage,

    meta: { title: "Каталог - Reread" },

    },

    {

    path: "/book/:id",

    name: "BookDetails",
```

Продовження додатку А

```
component: BookDetails,
meta: { title: "Книга - Reread" },
 },
 {
path: "/profile",
component: ProfileLayout,
children: [
      {
      path: "admin",
      component: AdminPanel,
      meta: {
      requiresAuth: true,
      requiresAdmin: true,
      title: "Адмін - Reread",
      },
      children: [
      {
```

Продовження додатку А

```
            path: "users",
                  component:                    ()                    =>
import("@/components/admin/UsersPanel.vue"),
                  meta: { title: "Користувачі" },
            },
            {
                  path: "books",
                  component:                    ()                    =>
import("@/components/admin/BooksPanel.vue"),
                  meta: { title: "Книги" },
            },
            {
                   path: "reservation",
                  component: () =>

import("@/components/admin/ReservationPanel.vue"),
                  meta: { title: "Резервування" },
            },
            {
```

Продовження додатку А

```
            path: "statistics",
              component: () =>

import("@/components/admin/StatisticsPanel.vue"),
                  meta: { title: "Статистика" },
            },
            ],
            },
            {
            path: "info",
            component: ProfileComp,
             meta: { title: "Профіль - Reread" },
            },
            {
            path: "likes",
            component: LikesComp,
            meta: { title: "Уподобання - Reread" },
            },
```

Продовження додатку А

```
    {
          path: "books_shelf",
          component: BooksComp,
          meta: { title: "Моя книжкова полиця - Reread" },
          },
          {
          path: "fines",
          component: FinesComp,
          meta: { title: "Штрафи - Reread" },
          },
          { path: "", redirect: "/profile/info" },
    ],
    meta: { requiresAuth: true },
     },
  ],
},
```

Продовження додатку А

```
{
    path: "/auth",
    component: AuthLayout,
    children: [
      {
      path: "login",
      name: "Login",
      component: LoginPage,
      meta: { title: "Вхід - Reread" },
      },
      {
      path: "register",
      name: "Register",
      component: RegisterPage,
      meta: { title: "Реєстрація - Reread" },
      },
    ],
```

Продовження додатку А

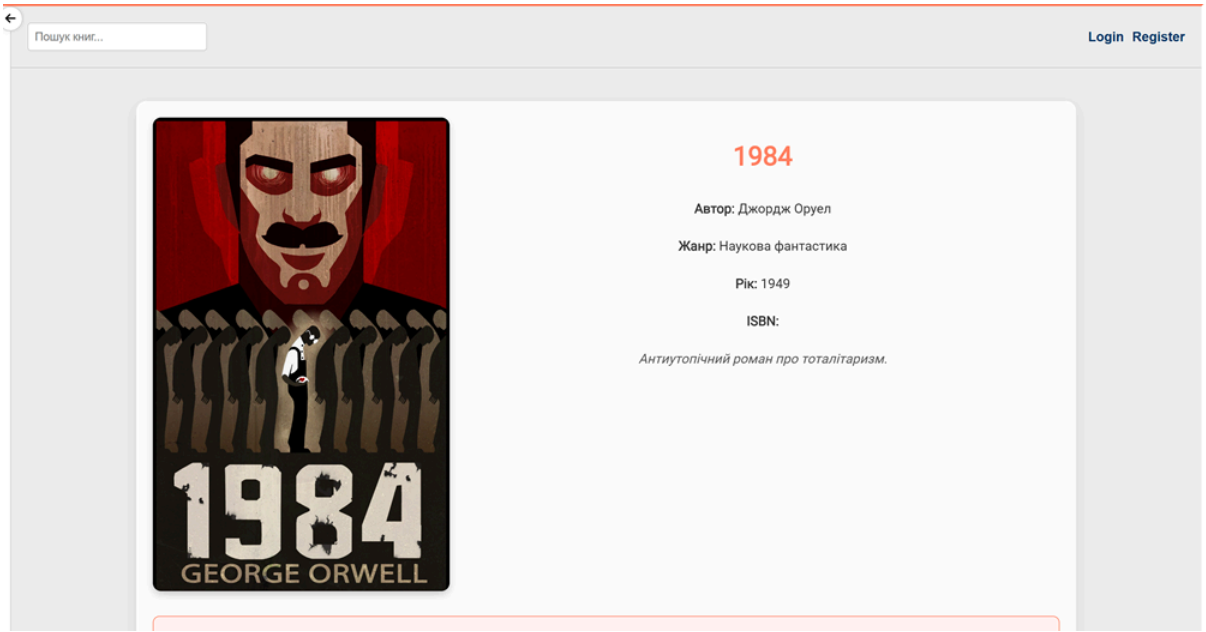
```
  },
];


const router = createRouter({
  history: createWebHashHistory(),
  routes,
});


router.beforeEach((to, from, next) => {
  const isAuthenticated = store.state.isAuthenticated;
  const isAdmin = store.state.isAdmin;


  if (to.meta.requiresAuth && !isAuthenticated) {
    next({ path: "/auth/login" });
  } else if (to.meta.requiresAdmin && !isAdmin) {
    next({ path: "/" });
  } else {
```

Продовження додатку А

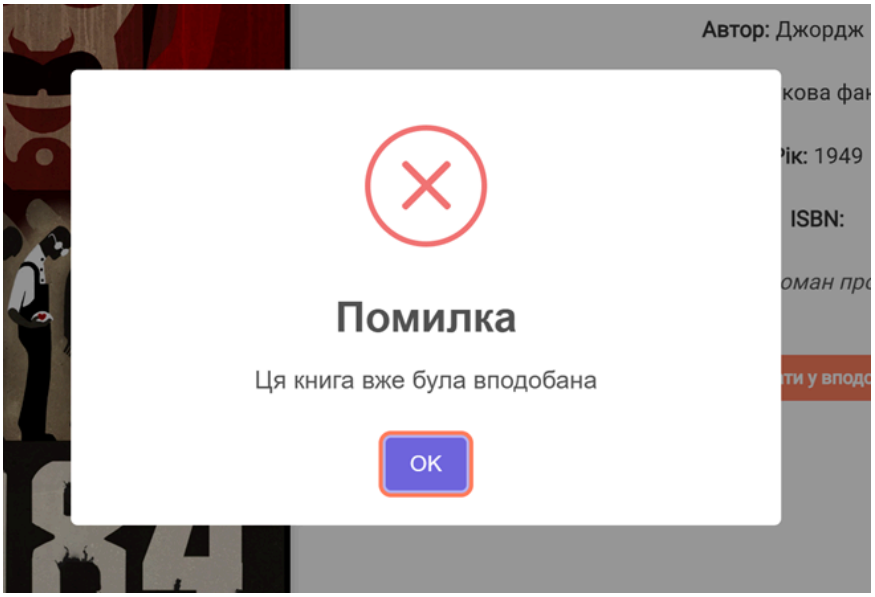
```
    next();
  }
});


router.afterEach((to) => {
  document.title = to.meta.title || "Default Title";
});


export default router;
```

Додаток Б. Знімки екрана процесу тестування створеного інтерфейсу

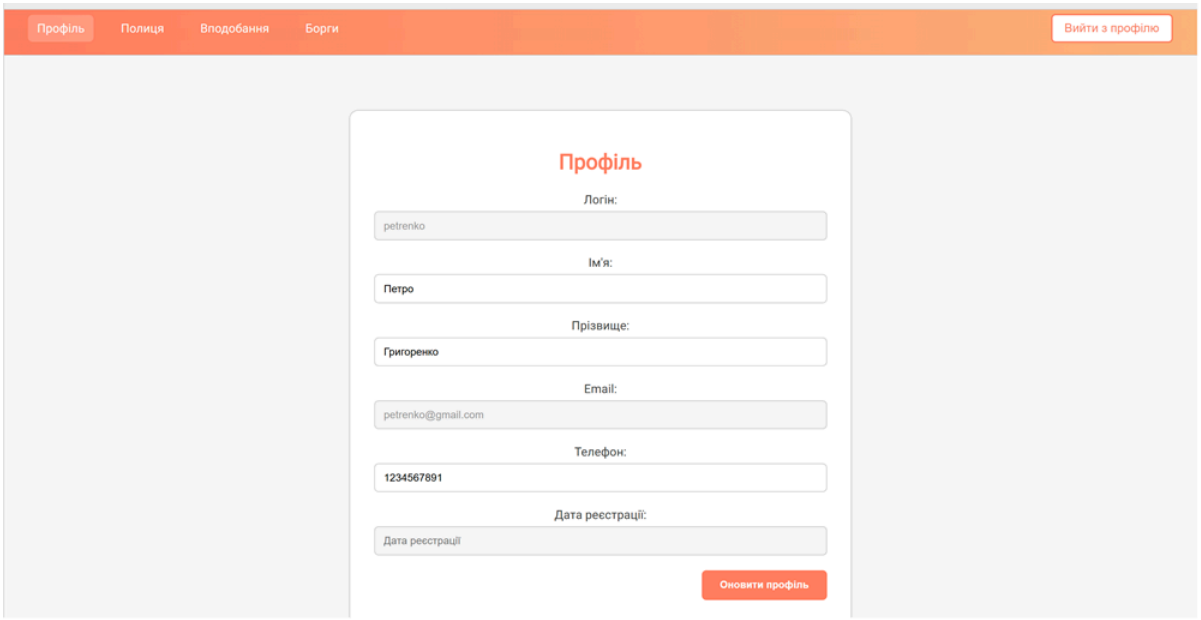
Користувач не залогінений:



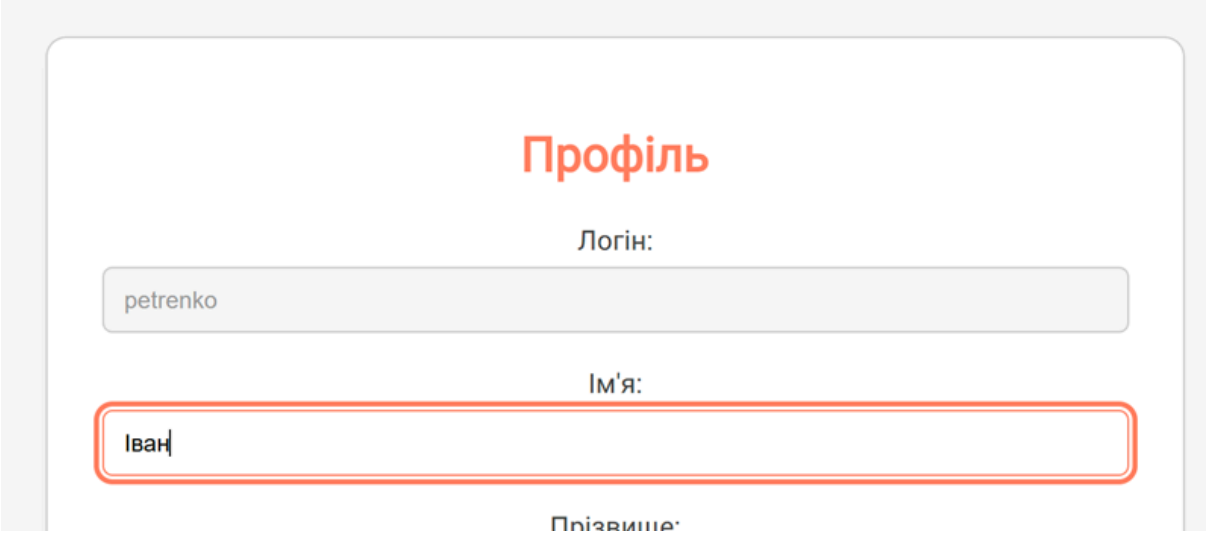
Помилка при повторному додаванні:

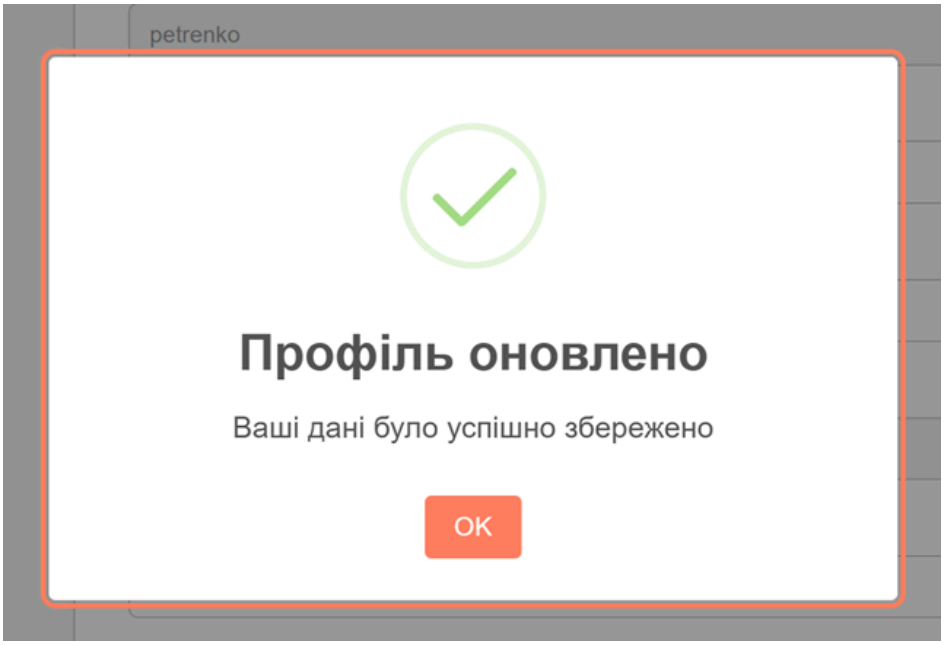


Дані профілю:

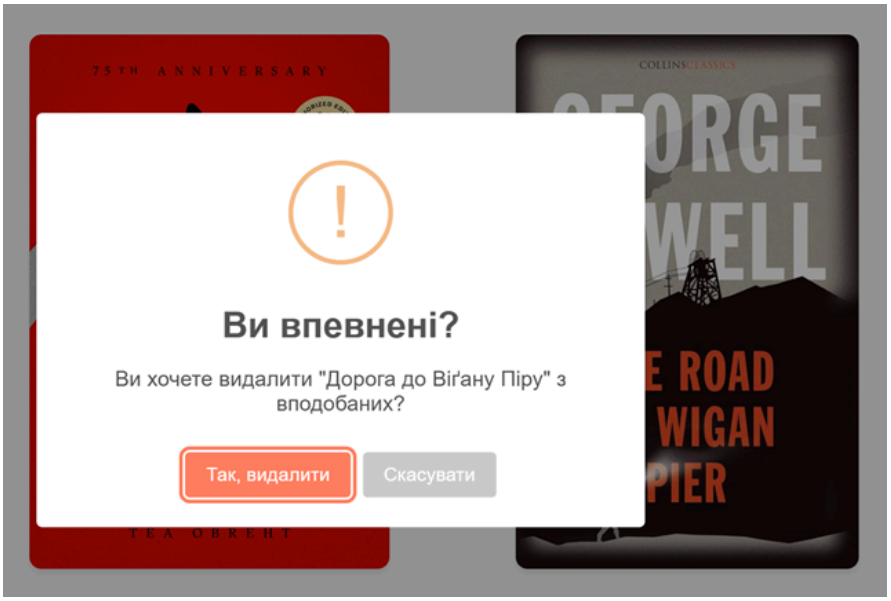


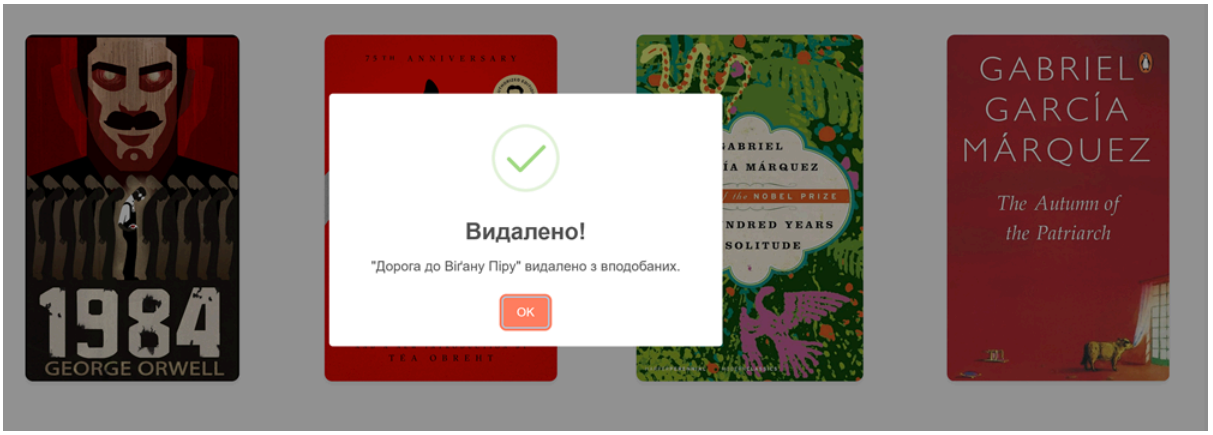
Оновлення профілю:



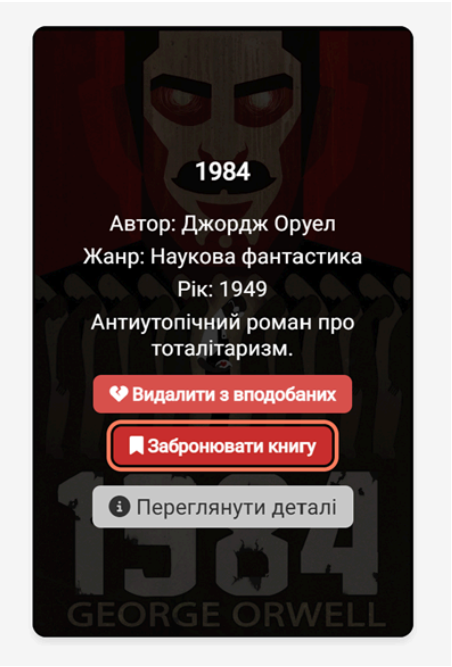


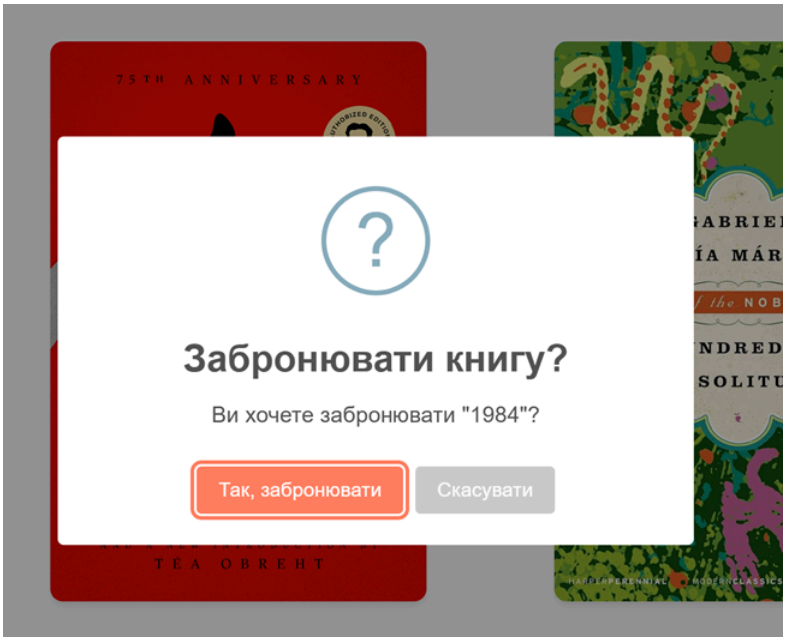
Видалення з вподобань:

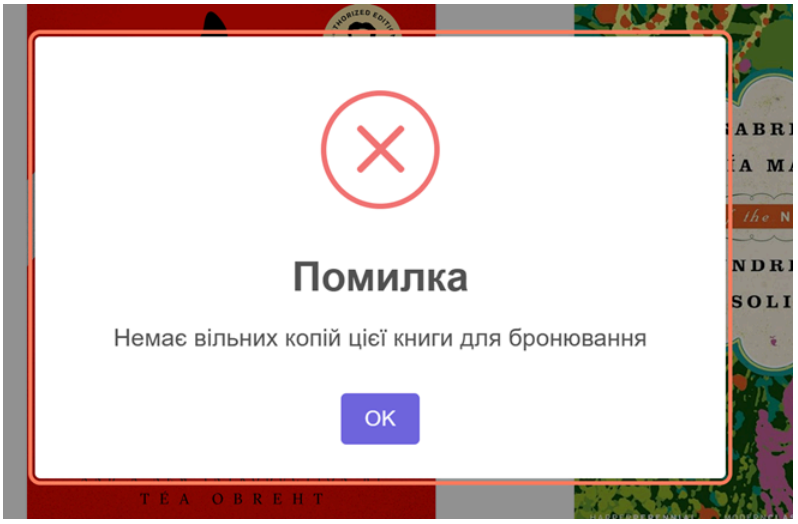


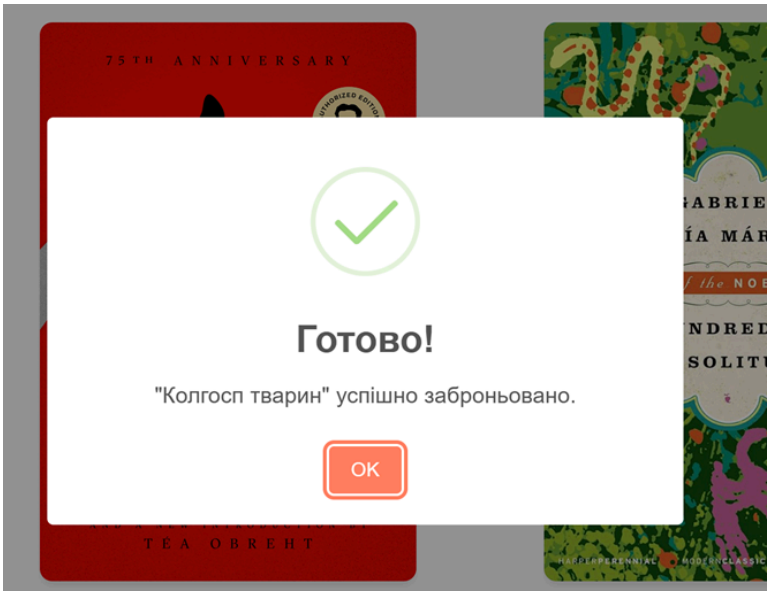


Бронювання книги: успішне та помилкове:

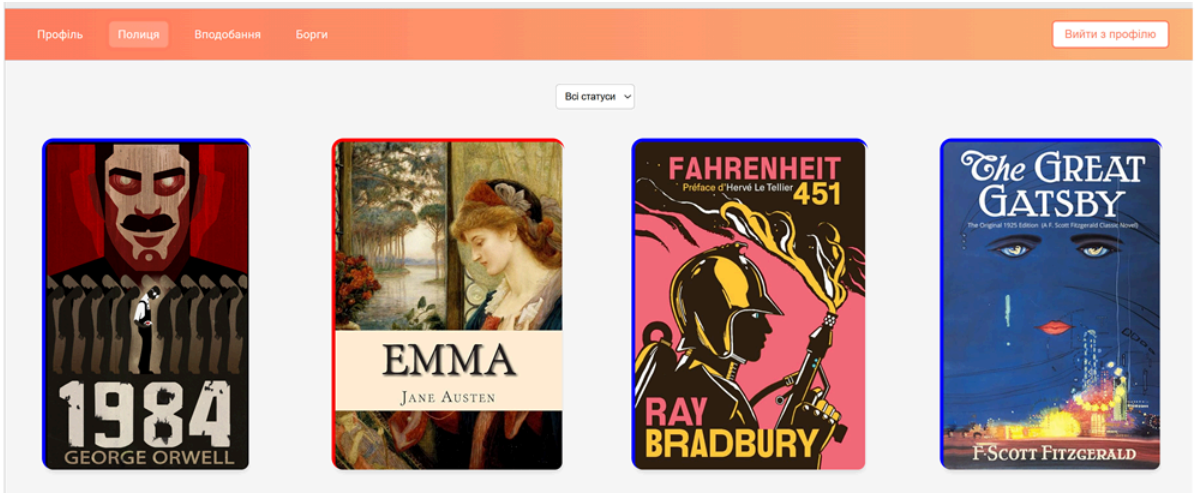


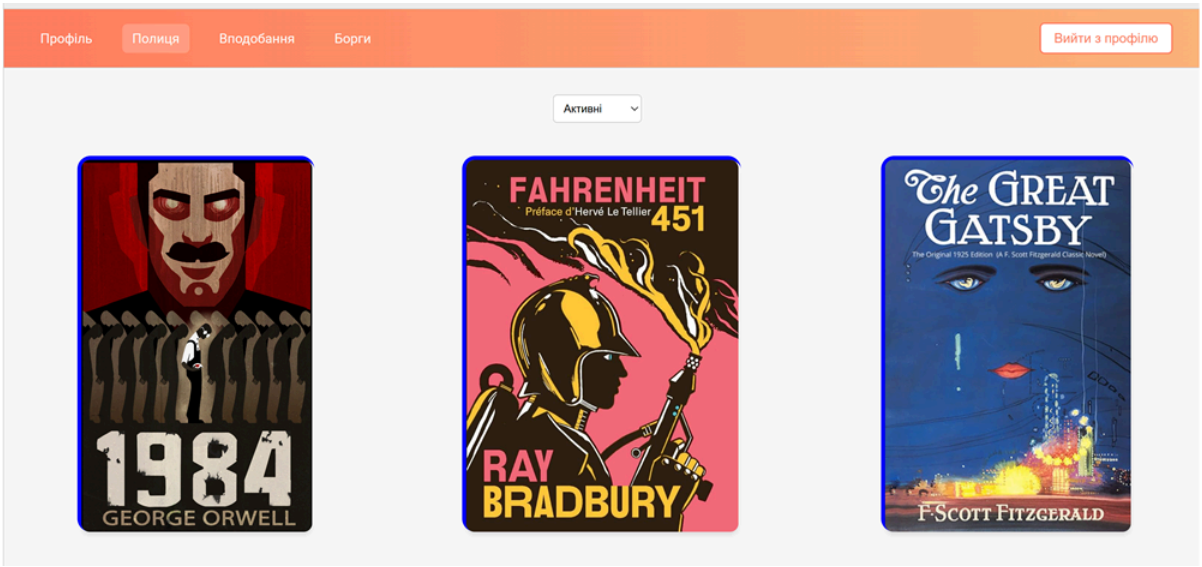

Забронювати книгу?
Ви хочете забронювати "1984"?
Так, забронювати
Скасувати

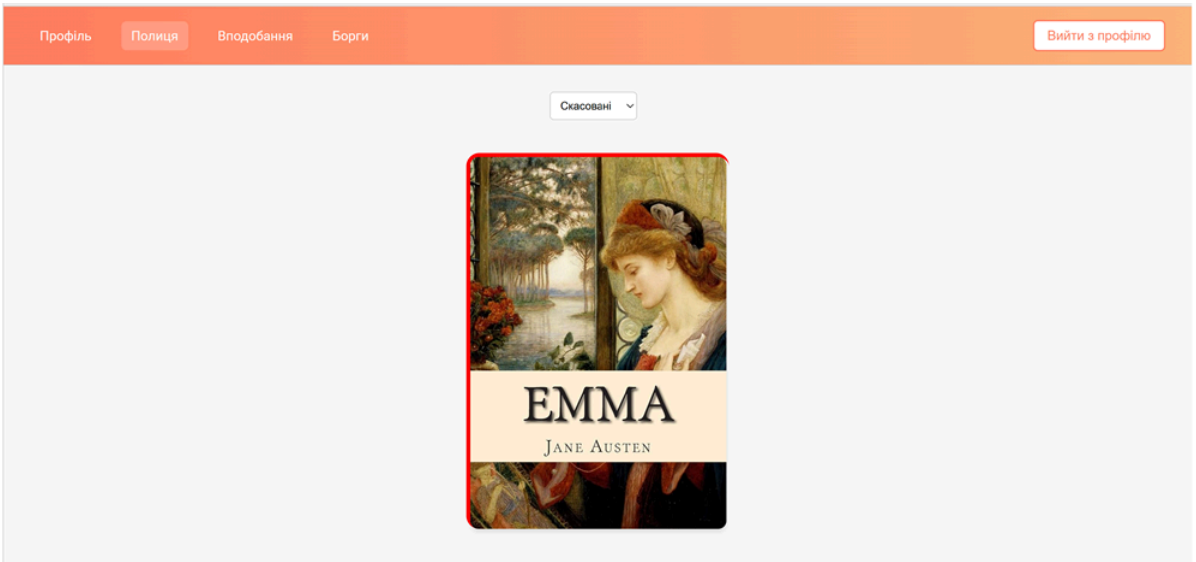

Помилка
Немає вільних копій цієї книги для бронювання
OK

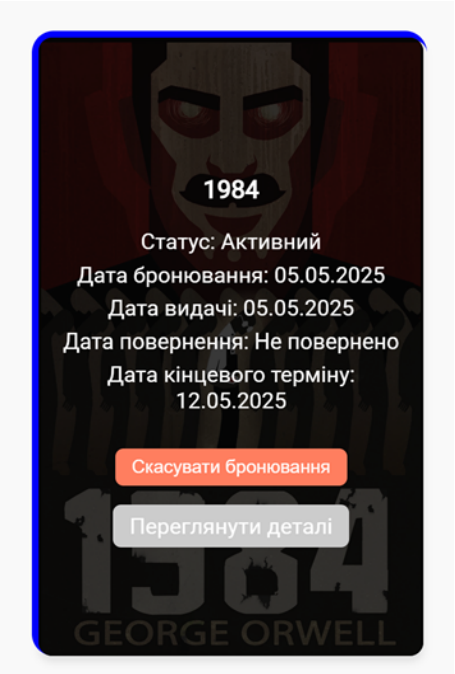

Готово!
"Колгосп тварин" успішно заброньовано.
OK

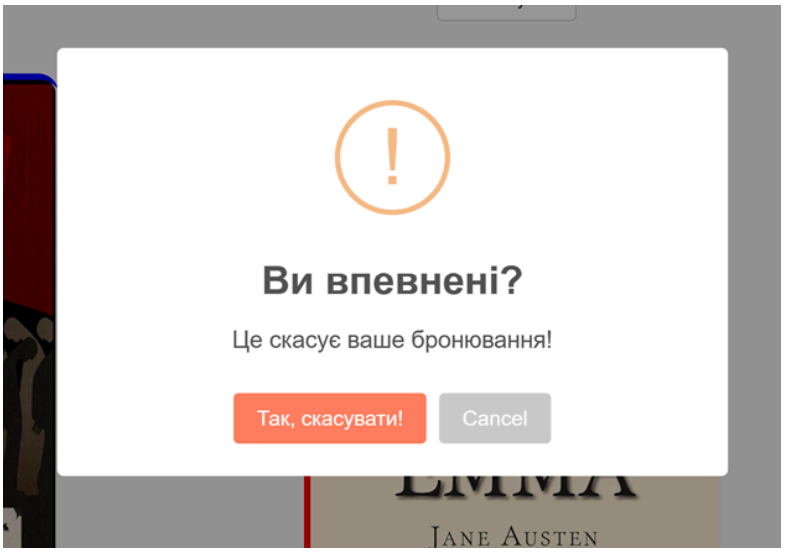
Тестування сторінки бронювань:

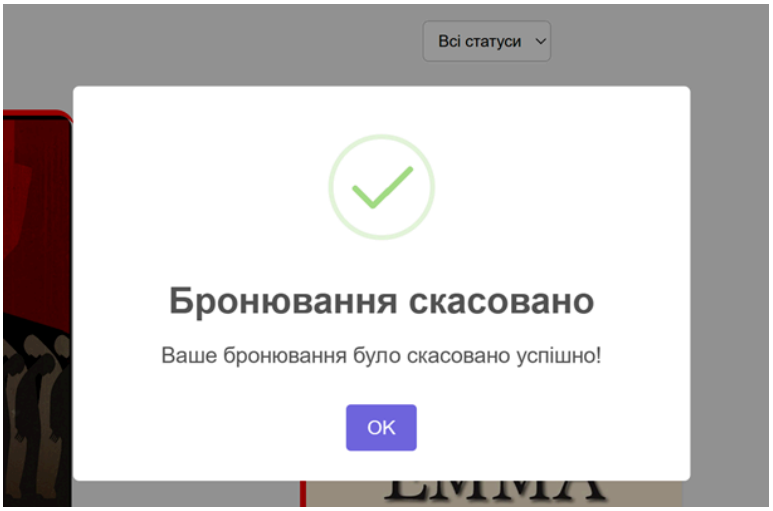


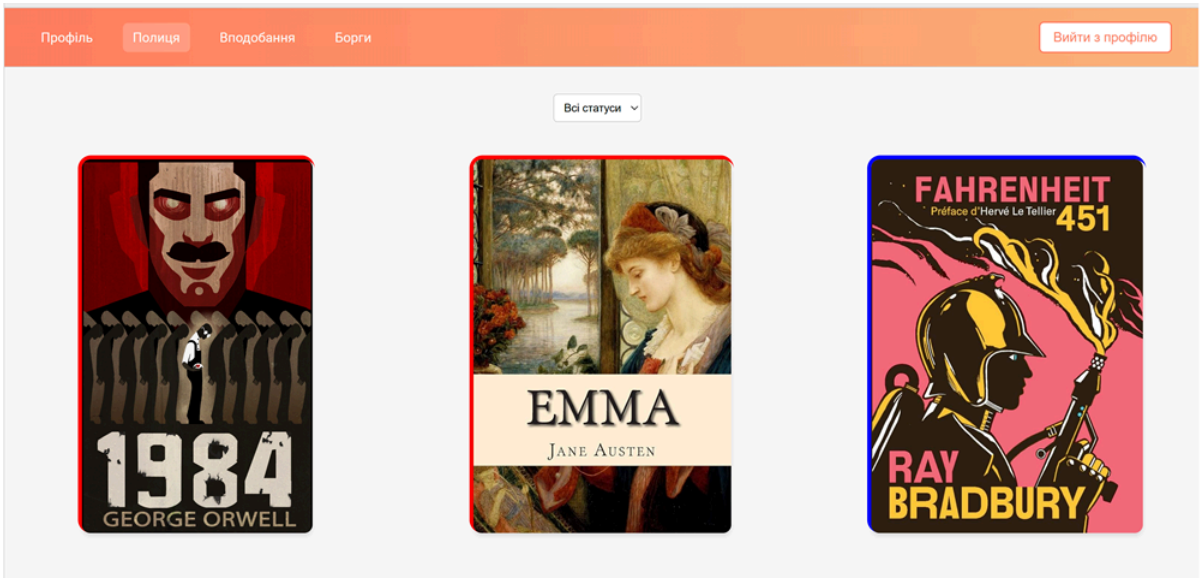





1984
Статус: Активний
Дата бронювання: 05.05.2025
Дата видачі: 05.05.2025
Дата повернення: Не повернено
Дата кінцевого терміну: 12.05.2025
Скасувати бронювання
Переглянути деталі
1984
GEORGE ORWELL


Ви впевнені?
Це скасує ваше бронювання!
Так, скасувати!
Cancel
EMMA
JANE AUSTEN


Всі статуси
Бронювання скасовано
Ваше бронювання було скасовано успішно!
OK
EMMA



Борги користувача: