МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка Веб-застосунку для бібліотеки з інтеграцією Spring Boot та Next.js**

Виконав: студент 4 курсу, групи 6.1212-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
_____ М. А. Переверзов
(підпис, ініціали та прізвище)
Керівник к.ф.-м.н., доцент Г. П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
_____ _____П. О. Лютий_____

Запоріжжя
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)

Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Тетяна КРИТСЬКА
“_01_” _березня_2026_р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

___Переверзову Марку Андрійовичу___
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка вебзастосунку для бібліотеки з інтеграцією Spring Boot та Next.js

керівник роботи Коломоєць Генадій Павлович, к.ф.-м.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 23.12.2025 р. № 1830-с_____

2. Строк подання студентом кваліфікаційної роботи 02 червня 2026 р.__
3. Вихідні дані кваліфікаційної роботи бакалавра
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково – пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- аналіз сучасних технологій для створення клієнтської частини Веб застосунків;
- проєктування Веб-застосунку;
- програмна реалізація застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) _9_ слайдів презентації____

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області, огляд аналогів, формування вимог до системи	02.03.2026 – 15.03.2026	виконано
2	Вибір технологічного стеку та проєктування архітектури системи	16.03.2026 – 29.03.2026	виконано
3	Проєктування моделі даних, REST API та інтерфейсу користувача; участь у конференції «Молода наука – 2026»	30.03.2026 – 17.04.2026	виконано
4	Реалізація серверної частини: сутності, репозиторії, сервіси, контролери, налаштування безпеки	18.04.2026 – 10.05.2026	виконано
5	Реалізація клієнтської частини, інтеграція з REST API, розгортання у Docker та тестування	11.05.2026 – 26.05.2026	виконано
6	Усунення зауважень наукового керівника, доопрацювання функціоналу та документації	27.05.2026 – 07.06.2026	виконано
7	Оформлення кваліфікаційної роботи, презентація, доповідь, демонстрація ПЗ	08.06.2026	виконано
8	Попередній захист	09.06.2026	
9	Захист кваліфікаційної роботи	16.06.2026	

Студент _____ М.А. Переверзов
(підпис) (прізвище та ініціали)

Керівник роботим _____ Г.П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок - 89

Рисунків - 21

Таблиць - 9

Джерел - 36

Переверзов М. А. Розробка вебзастосунку для бібліотеки з інтеграцією Spring Boot та Next.js : кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення» / науковий керівник Г. П. Коломоєць. Запоріжжя : ЗНУ, 2026. 89 с.

Метою роботи є проєктування та реалізація повнофункціонального вебзастосунку для управління бібліотечним фондом і обслуговуванням читачів із рольовою моделлю доступу, інтеграцією серверної та клієнтської частин через REST API та захистом на основі JWT.

У роботі проаналізовано предметну область бібліотечних інформаційних систем та наявні рішення (Koha, OpenBiblio, Evergreen, LibraryWorld), обґрунтовано вибір технологічного стеку. Реалізацію виконано на Java 25 із застосуванням Spring Boot 4.0.6 для побудови REST API (Spring Security, Spring Data JPA, Bean Validation, Flyway, JWT); клієнтська частина — на Next.js 16 та React 19 з TypeScript, база даних — MySQL 8.4.

Вебзастосунок реалізує повний бібліотечний цикл: пошук і перегляд каталогу, бронювання, видачу й повернення видань, контроль прострочень та адміністрування каталогу й користувачів із розмежуванням доступу для ролей гостя, читача, бібліотекаря, адміністратора. Проведено тестування критичної бізнес-логіки (21 тест) та локальне розгортання через Docker Compose.

Практичне значення роботи полягає у створенні готового до використання вебзастосунку, придатного для впровадження в навчальних і малих публічних бібліотеках та подальшого розвитку.

Ключові слова: *вебзастосунок, бібліотечна система, Spring Boot, Next.js, REST API, JWT, MySQL, рольова модель доступу.*

SUMMARY

Pages – 89

Figures – 21

Tables – 9

Sources – 36

Pereverzov M. A. Development of a web application for a library with Spring Boot and Next.js integration: bachelor's qualification work in specialty 121 "Software Engineering" / scientific supervisor H. P. Kolomoiets. Zaporizhzhia: ZNU, 2026. 89 p.

The purpose of the work is to design and implement a full-featured web application for library fund management and reader services with a role-based access model, integration of server and client parts through a REST API, and JWT-based authentication.

The study analyses the subject domain of library information systems and existing solutions (Koha, OpenBiblio, Evergreen, LibraryWorld) and substantiates the choice of the technology stack. The implementation is built on Java 25 using Spring Boot 4.0.6 for the REST API (Spring Security, Spring Data JPA, Bean Validation, Flyway, JWT); the client part is on Next.js 16 and React 19 with TypeScript, and the database is MySQL 8.4.

The web application implements a complete library cycle: catalog search and browsing, reservations, issuing and returning items, overdue tracking, and catalog/user administration with role-based access for guest, reader, librarian, and administrator roles. Critical business logic was tested (21 tests), with local deployment via Docker Compose.

The practical value lies in a ready-to-use web application suitable for deployment in educational and small public libraries and as a basis for further development.

Keywords: *web application, library system, Spring Boot, Next.js, REST API, JWT, MySQL, role-based access control.*

ЗМІСТ

ВСТУП 11
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ 15
1.1 Призначення та типові функції бібліотечних систем 15
1.2 Огляд існуючих рішень 18
1.2.1 Koha 18
1.2.2 OpenBiblio 19
1.2.3 Evergreen 20
1.2.4 LibraryWorld 21
1.3 Аналіз технологій для реалізації 23
1.3.1 Spring Boot (серверна частина) 24
1.3.2 Next.js (клієнтська частина) 25
1.3.3 MySQL (база даних) 26
1.3.4 REST API та JWT 27
1.4 Формування вимог до розроблюваної системи 28
РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ 31
2.1 Архітектура застосунку 31
2.2 Модель даних 34
2.2.1 Концептуальна модель 34
2.2.2 Структура бази даних 36
2.3 Проектування REST API 38
2.4 Проектування інтерфейсу користувача 41
РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ 45
3.1 Реалізація серверної частини (Spring Boot) 45
3.1.1 Налаштування проекту 45
3.1.2 Реалізація шару сутностей (Entity) 48
3.1.3 Реалізація бізнес-логіки (сервіси) 53
3.1.4 Реалізація REST-контролерів 59
3.1.5 Ініціалізація бази даних 67
3.2 Реалізація клієнтської частини (Next.js) 71

3.2.1 Налаштування проекту та авторизації 71
3.2.2 Каталог книг 75
3.2.3 Адміністративна панель 77
3.3 Розгортання та тестування 78
3.3.1 Локальне розгортання 78
3.3.2 Тестування функціоналу 81
ВИСНОВКИ 85
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 87

ВСТУП

Сучасна бібліотека давно перестала бути лише сховищем друкованих видань. Облік фонду, пошук потрібної книги, видача та повернення — усе це сьогодні відбувається в цифровому середовищі, і читач сприймає це як належне. Водночас фонди зростають, навантаження на персонал збільшується, а робота з паперовими формулярами стає відверто неефективною. Через це автоматизація бібліотечних процесів перетворилася з бажаної опції на робочу необхідність. Стосується це не лише великих установ: навіть невеликій навчальній чи публічній бібліотеці потрібно вести каталог, бачити рух кожного примірника, опрацьовувати бронювання й видачі та чітко розмежовувати, хто з користувачів які дії може виконувати.

Проблема в тому, що готові продукти на ринку рідко влаштовують таку бібліотеку повністю. Одні з них — громіздкі промислові системи, розраховані на великі мережі, які складно розгортати й супроводжувати. Інші давно застаріли — і за набором функцій, і за технологіями. Звідси й випливає завдання роботи: потрібен сучасний вебзастосунок, який нескладно впровадити, який повністю покриває бібліотечний цикл, має зрозумілий інтерфейс і при цьому залишається відкритим для доопрацювання.

Актуальність роботи має два боки — практичний і методичний. З практичного погляду бібліотеки справді потребують доступного інструмента, що закриває весь цикл обслуговування. З методичного — сам застосунок побудовано на стеку, який нині є фактичним стандартом промислової розробки: серверну частину на Spring Boot і клієнтську на Next.js рознесено та з'єднано через REST API, автентифікація тримається на JWT, а схема бази даних розвивається через версіоновані міграції. Для кваліфікаційної роботи зі спеціальності «Інженерія програмного забезпечення» це показовий і доречний набір рішень.

Мета роботи — спроєктувати та реалізувати повнофункціональний вебзастосунок для управління бібліотечним фондом і обслуговуванням

читачів, який реалізує повний бібліотечний цикл (каталог, бронювання, видача, повернення, адміністрування) з рольовим розмежуванням доступу на основі клієнт-серверної архітектури з інтеграцією Spring Boot і Next.js через REST API.

Для досягнення поставленої мети сформульовано такі **задачі**:

1. Проаналізувати предметну область бібліотечних інформаційних систем, визначити типові функції та сценарії використання.
2. Виконати огляд та порівняльний аналіз наявних рішень для автоматизації бібліотек.
3. Обґрунтувати вибір технологій серверної та клієнтської частин, бази даних і засобів інтеграції.
4. Сформувати функціональні та нефункціональні вимоги до системи.
5. Спроєктувати архітектуру застосунку, модель даних і програмний інтерфейс REST API.
6. Спроєктувати інтерфейс користувача для ролей гостя, читача, бібліотекаря та адміністратора.
7. Реалізувати серверну частину на Spring Boot із підсистемами автентифікації, каталогу, бронювань, видач і адміністрування.
8. Реалізувати клієнтську частину на Next.js, що споживає REST API.
9. Виконати модульне й інтеграційне тестування та локальне розгортання застосунку.

Об'єкт дослідження — процеси автоматизації роботи бібліотеки: облік фонду, обслуговування читачів, видача та повернення видань, адміністрування каталогу й користувачів.

Предмет дослідження — методи та засоби побудови вебзастосунку для бібліотеки на основі клієнт-серверної архітектури з розділенням відповідальності між серверною частиною на Spring Boot і клієнтською частиною на Next.js.

Методи дослідження. У роботі застосовано аналіз і порівняння (під час дослідження предметної області та аналогів), методи об'єктно-орієнтованого

аналізу та проєктування (під час побудови доменної моделі), методи реляційного моделювання (під час проєктування бази даних), принципи проєктування REST-інтерфейсів, а також методи модульного й інтеграційного тестування (під час перевірки коректності реалізації).

Практичне значення одержаних результатів полягає у створенні готового до використання вебзастосунку, який реалізує повний цикл бібліотечного обслуговування, має зрозумілу модульну архітектуру, демонстраційні дані й тести, а тому може бути впроваджений у навчальних та невеликих публічних бібліотеках, а також використаний як основа для подальшого розширення функціональності.

Апробація результатів дослідження. Результати дослідження були представлені на XVIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука–2026».

Глосарій

API — Application Programming Interface — програмний інтерфейс застосунку.

БД — База даних.

CORS (Cross-Origin Resource Sharing) — механізм спільного використання ресурсів між джерелами.

CRUD (Create, Read, Update, Delete) — базові операції над даними.

CSR — Client-Side Rendering — рендеринг на боці клієнта.

DTO — Data Transfer Object — об'єкт передавання даних.

HTTP — HyperText Transfer Protocol — протокол передавання гіпертексту.

IFLA — International Federation of Library Associations — Міжнародна федерація бібліотечних асоціацій.

ILS — Integrated Library System — інтегрована бібліотечна система.

ISBN — International Standard Book Number — міжнародний стандартний книжковий номер.

JPA — Jakarta Persistence API — специфікація доступу до даних у Java.

JSON — JavaScript Object Notation — текстовий формат обміну даними.

JWT — JSON Web Token — стандарт токена для автентифікації та авторизації.

MARC21 — Machine-Readable Cataloging 21 — міжнародний формат машиночитаної каталогізації.

OPAC — Online Public Access Catalog — публічний онлайн-каталог доступу.

ORM — Object-Relational Mapping — об'єктно-реляційне відображення.

ПЗ — програмне забезпечення.

RBAC — Role-Based Access Control — рольова модель розмежування доступу.

REST — Representational State Transfer — архітектурний стиль побудови вебсервісів.

SaaS — Software as a Service — програмне забезпечення як послуга.

SPA — Single Page Application — односторінковий застосунок.

SQL — Structured Query Language — мова структурованих запитів.

SSG — Static Site Generation — статична генерація сторінок.

SSR — Server-Side Rendering — рендеринг на боці сервера.

СУБД — система управління базами даних.

UI — User Interface — інтерфейс користувача.

UNIMARC — Universal Machine-Readable Cataloging — універсальний формат машиночитаної каталогізації.

UX — User Experience — досвід користувача.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ

1.1 Призначення та типові функції бібліотечних систем

Під бібліотечною інформаційною системою (англ. Integrated Library System, ILS) розуміють програмний комплекс, що автоматизує ключові процеси роботи бібліотеки — від формування й підтримки каталогу до обліку примірників (як фізичних, так і електронних), обслуговування читачів, відстеження руху видань і підготовки звітності. Подібні системи з'явилися ще наприкінці 1960-х – у 1970-х роках, спочатку лише як інструменти автоматизації каталогізації, проте з часом їхній функціонал розширився на весь цикл бібліотечного обслуговування. Сьогодні переважна більшість таких рішень виконана у вигляді вебзастосунків, доступ до яких не потребує встановлення окремого клієнтського програмного забезпечення.

Незалежно від масштабу бібліотеки, можна виокремити стабільний набір типових функцій, які становлять ядро будь-якої бібліотечної системи. До них належать каталогізація (опис видань з усіма бібліографічними атрибутами), облік примірників (відстеження кожної фізичної копії видання з її інвентарним номером і станом), пошук і навігація по фонду, обслуговування читачів (реєстрація, видача, повернення, продовження, бронювання), а також адміністрування (керування довідниками, користувачами та правами доступу).

Каталог є центральним елементом системи. Кожне видання описується сукупністю бібліографічних атрибутів: назва, автор або колектив авторів, видавництво, рік видання, мова, кількість сторінок, міжнародний стандартний книжковий номер (ISBN) та тематична категорія. Важливо розрізняти поняття «видання» (бібліографічний опис твору) і «примірник» (конкретна фізична копія, що зберігається у фонді). Одному виданню може відповідати кілька примірників, кожен з яких має власний інвентарний номер, місце зберігання і поточний статус (доступний, виданий, зарезервований, на ремонті, списаний). Саме примірник, а не видання, бере участь в операціях видачі та повернення,

тому коректне моделювання цього зв'язку є принциповим для правильної роботи всієї системи.

Обслуговування читачів охоплює кілька взаємопов'язаних процесів. Бронювання дозволяє читачеві закріпити за собою доступний примірник видання до моменту його отримання. Видача (loan) фіксує факт передавання примірника читачеві на визначений строк із зазначенням дати повернення. Повернення завершує цикл видачі та повертає примірник у фонд. Окремою важливою функцією є контроль простроченої заборгованості — виявлення видач, строк яких вичерпано, що дозволяє бібліотекарю своєчасно реагувати. Усі ці операції повинні зберігати історію, щоб забезпечити простежуваність дій та можливість аудиту.

Суттєвою характеристикою бібліотечної системи є розмежування ролей користувачів. Типово виокремлюють чотири категорії: гість (неавторизований відвідувач, якому доступний лише перегляд і пошук у каталозі), читач (зареєстрований користувач, який може бронювати видання і переглядати свої бронювання та видачі), бібліотекар (співробітник, що підтверджує бронювання, видає та приймає видання, контролює прострочення) та адміністратор (користувач із повними правами на керування каталогом, довідниками й обліковими записами). Кожна роль має чітко окреслений набір дозволених дій, що визначає вимоги до підсистеми авторизації.

Зведений перелік типових функцій бібліотечної системи з прив'язкою до ролей наведено в таблиці 1.1.

Таблиця 1.1 — Типові функції бібліотечної системи за ролями

Функція	**Гість**	**Читач**	**Бібліотекар**	**Адміністратор**
Пошук і перегляд каталогу	+	+	+	+
Перегляд деталей видання	+	+	+	+
Бронювання видання	—	+	+	+
Перегляд власних бронювань / видач	—	+	—	—
Підтвердження / скасування бронювань	—	—	+	+
Видача примірника	—	—	+	+
Повернення примірника	—	—	+	+
Контроль прострочень	—	—	+	+
Керування каталогом і довідниками (CRUD)	—	—	—	+
Керування користувачами	—	—	—	+
Перегляд дашборду показників	—	—	+	+

Наведений набір функцій і рольова модель формують основу, на яку спирається подальше проєктування. Перш ніж перейти до вибору технологій, доцільно розглянути, як ці функції реалізовано в наявних на ринку рішеннях.

1.2 Огляд існуючих рішень

Для обґрунтування доцільності власної розробки та виявлення сильних і слабких сторін наявних підходів було проаналізовано чотири поширені бібліотечні системи різного класу: дві відкриті промислові системи (Koha, Evergreen), одну застарілу, але показову відкриту систему (OpenBiblio) та одне комерційне хмарне рішення (LibraryWorld). Критеріями порівняння обрано: модель розгортання, технологічний стек, повноту функціоналу, складність впровадження й супроводу, сучасність інтерфейсу та відкритість коду.

1.2.1 Koha

Koha [1] розробляється з 1999 року і станом на сьогодні є, ймовірно, найпоширенішою у світі повнофункціональною бібліотечною системою з відкритим кодом. Система покриває практично весь спектр бібліотечних операцій — каталогізацію за міжнародними стандартами MARC21 та UNIMARC, облік примірників, обслуговування читачів (circulation), комплектування фонду, роботу із серіальними виданнями, формування звітів, а також публічний онлайн-каталог OPAC (Online Public Access Catalog). Технологічно Koha реалізована на Perl з використанням СУБД MariaDB/MySQL, а її інтерфейс будується за принципом серверного рендерингу сторінок.

Сильними сторонами Koha є надзвичайна функціональна повнота, відповідність бібліотечним стандартам, велика спільнота та підтримка багатьох мов. Водночас саме повнота й історичність системи є джерелом її недоліків з погляду цієї роботи: значний обсяг успадкованого коду на Perl, складність розгортання й супроводу, важка для модифікації архітектура та інтерфейс, що візуально та з погляду досвіду користувача поступається сучасним односторінковим застосункам (SPA — Single Page Application). Для невеликої навчальної бібліотеки Koha часто є надлишково складною.

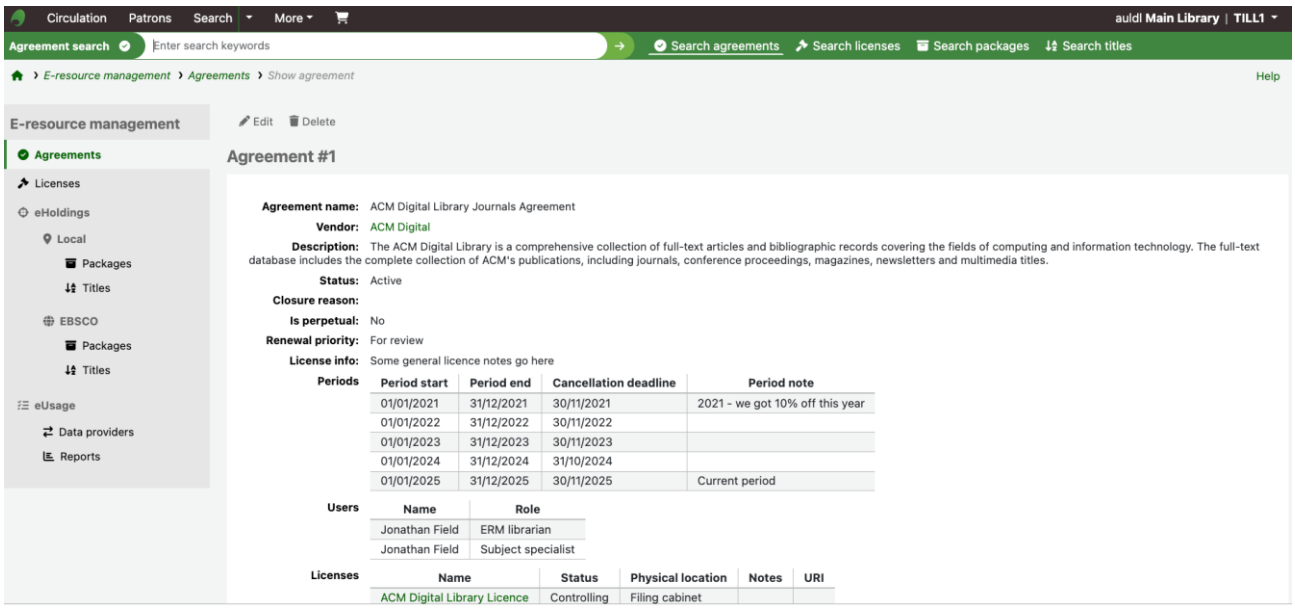


Рисунок 1.1 — Інтерфейс бібліотечної системи Koha

1.2.2 OpenBiblio

OpenBiblio [2] — значно простіша система, написана на PHP з використанням MySQL. Її функціональність обмежується базовим набором: каталогізацією, обліком читачів, операціями видачі й повернення та елементарним онлайн-каталогом. Завдяки невибагливості до ресурсів і простоті розгортання вона свого часу набула популярності серед невеликих бібліотек і шкіл.

Проте проєкт фактично припинив активний розвиток, його технологічний стек і архітектура застаріли, інтерфейс не відповідає сучасним вимогам до зручності та адаптивності, а функціональність обмежена базовими операціями без розвинених механізмів розмежування прав, бронювання та звітності. OpenBiblio цінна як приклад мінімалістичного рішення, однак не може розглядатися як актуальна основа для нової розробки.

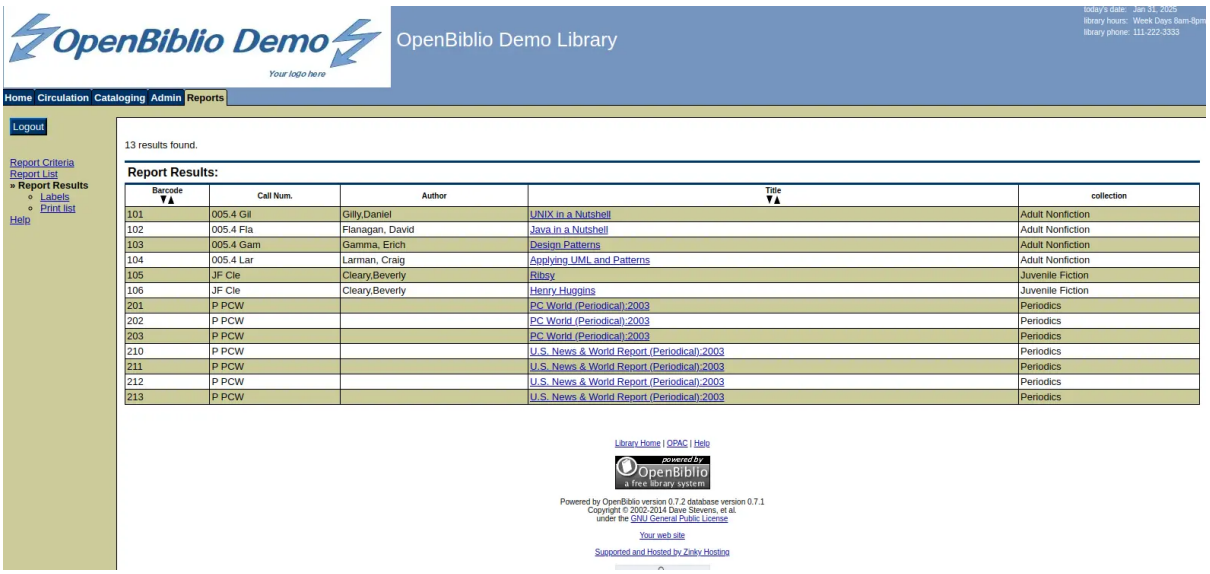


Рисунок 1.2 — *Інтерфейс системи OpenBiblio*

1.2.3 Evergreen

Evergreen [3] спочатку створювався для потреб великих бібліотечних консорціумів, тому його ключовою особливістю є саме орієнтація на масштабованість і підтримку мереж з багатьох бібліотек із спільним каталогом. Реалізована на стеку Perl/C з СУБД PostgreSQL, ця система пропонує розвинений функціонал обслуговування читачів, каталогізації та адміністрування.

Сильна сторона Evergreen — здатність обслуговувати великі обсяги даних і складні організаційні структури. Це водночас є і її обмеженням у контексті цієї роботи: розгортання й налаштування системи потребують значних зусиль і кваліфікації, а її можливості надлишкові для окремої навчальної чи невеликої публічної бібліотеки. Архітектурно Evergreen також є складним для модифікації успадкованим рішенням.

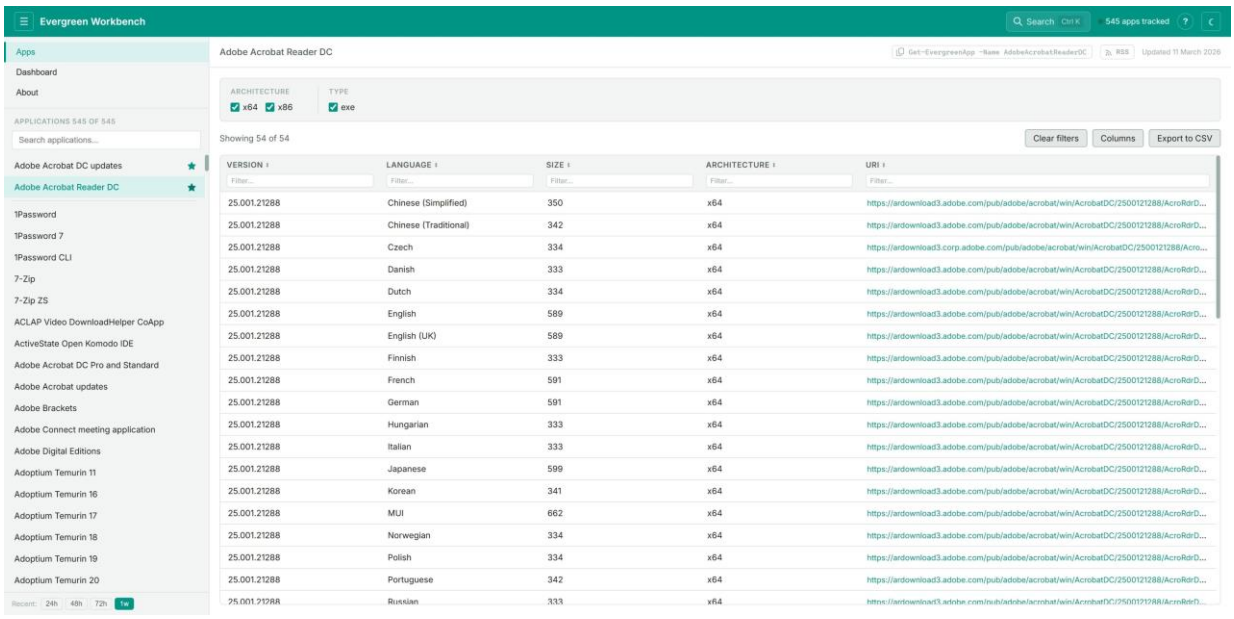


Рисунок 1.3 — *Інтерфейс бібліотечної системи Evergreen*

1.2.4 LibraryWorld

На відміну від трьох попередніх рішень, LibraryWorld [4] є комерційним продуктом, що поширюється за моделлю SaaS (програмне забезпечення як послуга): доступ надається через браузер за підпискою, а питання розгортання й супроводу серверної інфраструктури повністю лягають на постачальника. Функціонально система покриває каталогізацію, обслуговування читачів, онлайн-каталог, звітність і має мобільний доступ.

Перевагою LibraryWorld є простота початку роботи й відсутність витрат на адміністрування інфраструктури. Недоліками з погляду цієї роботи є закритість коду (неможливість дослідити чи модифікувати систему), залежність від постачальника послуги, передавання даних на зовнішні сервери та абонентська плата. Для навчального проєкту, метою якого є демонстрація інженерних компетенцій з розробки, закрите хмарне рішення не може слугувати ні основою, ні прямим аналогом.

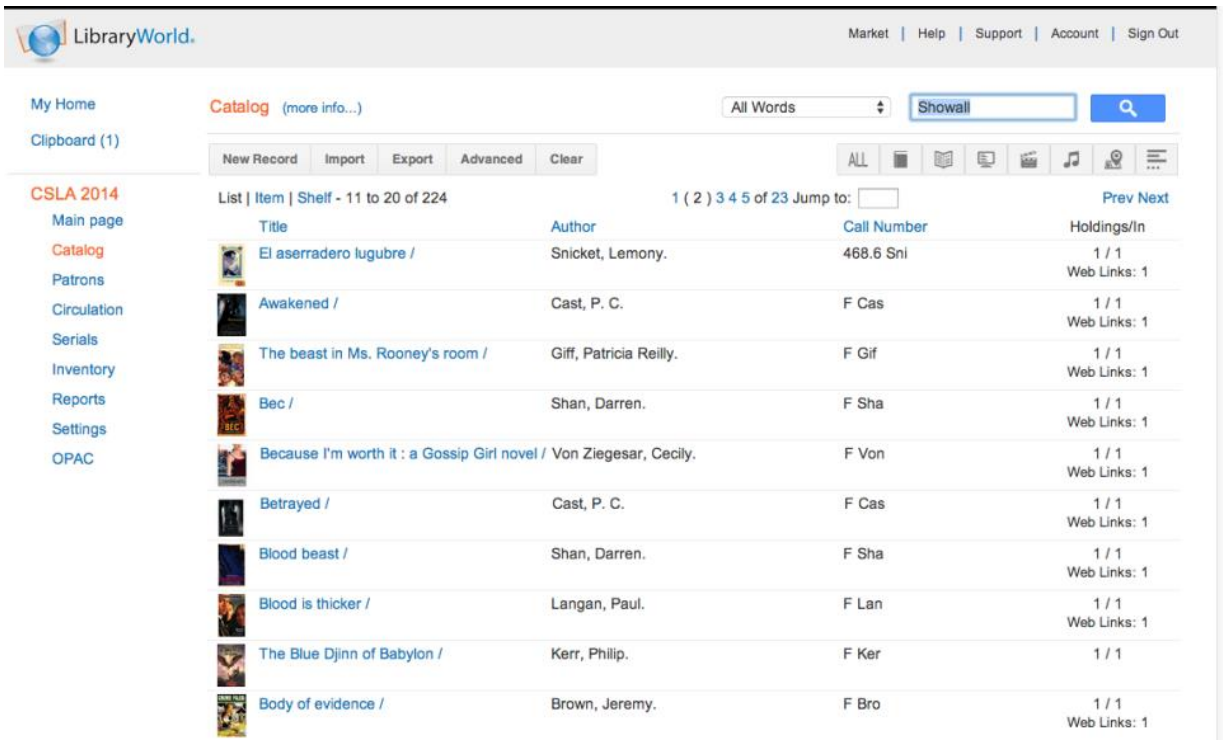


Рисунок 1.4 — *Інтерфейс хмарної системи LibraryWorld*

Результати порівняльного аналізу розглянутих рішень узагальнено в таблиці 1.2.

Таблиця 1.2 — *Порівняльний аналіз бібліотечних систем*

Критерій	**Koha**	**OpenBiblio**	**Evergreen**	**LibraryWorld**	**Розроблювана система**
Модель розгортання	Локальне	Локальне	Локальне	Хмарне (SaaS)	Локальне / хмарне
Технологічний стек	Perl, MariaDB/MySQL	PHP, MySQL	Perl/C, PostgreSQL	Закритий	Java, Spring Boot, Next.js, MySQL

Відкритість коду	Відкрита	Відкрита	Відкрита	Закрита	Відкрита (навчальна)
Повнота функціоналу	Висока	Низька	Висока	Середня–висока	Достатня для повного циклу
Складність впровадження	Висока	Низька	Дуже висока	Дуже низька	Низька
Сучасність інтерфейсу	Середня	Низька	Середня	Середня	Висока (SPA)
Адаптивність / мобільність	Часткова	Ні	Часткова	Так	Так

Проведений аналіз показує, що наявні рішення або надлишково складні та орієнтовані на великі організації (Koha, Evergreen), або застарілі та функціонально обмежені (OpenBiblio), або закриті комерційні (LibraryWorld). Це підтверджує доцільність розробки власного вебзастосунку, який поєднує достатню для повного циклу обслуговування функціональність із сучасним технологічним стеком, зручним адаптивним інтерфейсом і відкритою, придатною до розвитку архітектурою.

1.3 Аналіз технологій для реалізації

Вибір технологічного стеку є одним із ключових інженерних рішень, що визначає якість, супроводжуваність і масштабованість майбутньої системи. В основу проєкту покладено принцип чіткого розділення відповідальності між серверною частиною (backend), яка реалізує бізнес-логіку та доступ до даних, і клієнтською частиною (frontend), яка відповідає за подання й взаємодію з

користувачем. Взаємодія між ними здійснюється через REST API. Такий підхід (так звана «decoupled», або «headless», архітектура) дозволяє розробляти, тестувати й масштабувати частини системи незалежно одна від одної, а також у разі потреби підключати до того самого API інші клієнти, наприклад мобільний застосунок.

1.3.1 Spring Boot (серверна частина)

Серверну частину системи реалізовано мовою Java 25 (LTS-версія з довготривалою підтримкою) із застосуванням фреймворку Spring Boot 4.0.6 [5]. Spring Boot надбудовується над екосистемою Spring і за рахунок механізму автоконфігурації, вбудованого вебсервера та продуманих налаштувань за замовчуванням знімає з розробника значну частину рутинного конфігурування, дозволяючи зосередитися безпосередньо на бізнес-логіці застосунку.

Серед компонентів екосистеми Spring у роботі використано такі модулі. Spring Web MVC [6] забезпечує побудову REST-контролерів та обробку HTTP-запитів. Spring Data JPA [7] надає високорівневий доступ до бази даних через об'єктно-реляційне відображення на базі Hibernate [8], що дозволяє працювати з даними як з об'єктами Java та автоматично генерувати типові запити через репозиторії. Spring Security [9] відповідає за автентифікацію й авторизацію, зокрема за розмежування доступу до ресурсів на основі ролей. Bean Validation (Jakarta Validation [10]) забезпечує декларативну перевірку коректності вхідних даних за допомогою анотацій. Для документування програмного інтерфейсу застосовано бібліотеку springdoc-openapi [11], що автоматично формує специфікацію OpenAPI [12] та інтерактивний інтерфейс Swagger UI.

Окремої уваги заслуговує підхід до керування схемою бази даних. Замість автоматичної генерації таблиць засобами Hibernate у проєкті застосовано інструмент Flyway [13], який реалізує концепцію версіонованих міграцій: кожна зміна схеми описується окремим SQL-сценарієм із

порядковим номером версії (V1, V2, …), що застосовується послідовно й контрольовано. JPA при цьому працює в режимі лише перевірки відповідності (ddl-auto: validate), тобто звіряє об'єктну модель зі схемою, але не змінює її. Такий підхід («database-first» з контролем версій) є галузевим стандартом для промислових систем, оскільки забезпечує відтворюваність і керованість еволюції бази даних.

Обґрунтування вибору Java та Spring Boot полягає в тому, що це зрілий, перевірений промисловістю стек [14] із потужною екосистемою, статичною типізацією (що зменшує кількість помилок на етапі компіляції), розвиненими засобами тестування та широкими можливостями масштабування. Альтернативні варіанти серверних платформ (Node.js/Express, Python/Django, PHP/Laravel, ASP.NET) також розглядалися, проте Spring Boot обрано як рішення, що найкраще поєднує надійність, структурованість і відповідність сучасним промисловим практикам, а також є показовим з погляду демонстрації інженерних компетенцій.

1.3.2 Next.js (клієнтська частина)

Для клієнтської частини обрано фреймворк Next.js 16 [15], бібліотеку React 19 [16] та мову TypeScript [17]. React реалізує компонентний підхід до побудови інтерфейсу, коли вся вебсторінка складається з невеликих ізольованих компонентів, кожен з яких інкапсулює власний стан і може повторно використовуватися в різних частинах застосунку. Next.js, своєю чергою, надбудовує над React функціонал повноцінного фреймворку — маршрутизацію, кілька стратегій рендерингу, оптимізацію збірки та зручну структуру проєкту.

Використано модель маршрутизації App Router, що є сучасним підходом Next.js до організації сторінок і дозволяє гнучко поєднувати рендеринг на боці сервера (SSR), статичну генерацію (SSG) та рендеринг на боці клієнта (CSR) залежно від характеру конкретної сторінки. Для сторінок, що працюють з динамічними даними користувача (робочий простір читача, бібліотекаря,

адміністратора), доцільним є клієнтський рендеринг із завантаженням даних через API, тоді як публічні сторінки можуть використовувати серверні можливості для пришвидшення першого відображення.

Застосування мови TypeScript забезпечує статичну типізацію коду клієнтської частини, що суттєво зменшує кількість помилок, пов'язаних із некоректними типами даних, і покращує підтримуваність коду завдяки автодоповненню та контролю типів на етапі розробки. Для керування серверним станом і кешуванням запитів використано бібліотеку TanStack Query [18], яка бере на себе завантаження, кешування, повторні спроби та інвалідацію даних, отриманих з API, звільняючи розробника від написання значної кількості шаблонного коду. Стилізацію виконано засобами Tailwind CSS 4 [19] — утилітарного CSS-фреймворку, що дозволяє швидко будувати узгоджений адаптивний інтерфейс безпосередньо в розмітці компонентів. Для іконок використано бібліотеку lucide-react [20], а керування пакетами здійснюється за допомогою пакетного менеджера pnpm [21].

Вибір Next.js обґрунтовується тим, що він є одним із провідних сучасних фреймворків для побудови вебзастосунків, поєднує продуктивність розробки з якісним результатом для кінцевого користувача (швидкість, адаптивність, зручність) і добре узгоджується з обраною decoupled-архітектурою, в якій клієнтська частина споживає окремий REST API.

1.3.3 MySQL (база даних)

Як засіб зберігання даних у проєкті використано реляційну СУБД MySQL 8.4 [22]. Предметна область бібліотечної системи добре вписується в реляційну модель: між основними сутностями — виданнями, авторами, категоріями, примірниками, читачами, бронюваннями та видачами — існують чіткі зв'язки, дотримання яких є критичним для коректної роботи всієї системи. Саме тому обрано реляційну модель з підтримкою зовнішніх ключів, обмежень унікальності та транзакційності.

MySQL обрано як одну з найпоширеніших, добре задокументованих і надійних реляційних СУБД з відкритим кодом, що має широку підтримку в екосистемі Spring і зручні засоби розгортання, зокрема через контейнеризацію. Підтримка транзакцій (ACID — Atomicity, Consistency, Isolation, Durability: атомарність, узгодженість, ізольованість, стійкість), надійність якої є критичною для систем, що працюють з даними [23], є принциповою для бізнес-операцій бібліотеки: наприклад, видача примірника має атомарно змінювати і статус примірника, і стан відповідного бронювання, і створювати запис про видачу — частковий збій тут неприпустимий. Для забезпечення зручності тестування в проєкті додатково використано вбудовану СУБД H2 у режимі сумісності з MySQL, що дозволяє виконувати інтеграційні тести без потреби в реальному сервері бази даних.

1.3.4 REST API та JWT

В основу взаємодії клієнтської та серверної частин покладено архітектурний стиль REST (Representational State Transfer) [24; 25]. У межах цього підходу кожен ресурс системи — видання, бронювання, видача, користувач — отримує власну уніфіковану URL-адресу, а операції над ним виконуються стандартними методами HTTP (GET — отримання, POST — створення, PUT — оновлення, DELETE — видалення), а дані передаються у форматі JSON. Такий підхід забезпечує простоту, передбачуваність роботи API та його незалежність від конкретної клієнтської реалізації, що добре узгоджується з обраною архітектурою застосунку.

Архітектурний стиль REST є безстановим (stateless): сервер не зберігає інформацію про сесію між окремими запитами. Тому для автентифікації та авторизації в проєкті застосовано JSON Web Token (JWT) [26] — підписаний токен, що містить дані про користувача та його повноваження і додається клієнтом до заголовка кожного захищеного запиту. Перевіряючи підпис токена, сервер встановлює особу користувача та його права без звернення до будь-якого сховища сесій. Реалізована схема передбачає два типи токенів:

короткоживучий access-токен для доступу до захищених ресурсів і довгоживучий refresh-токен, що зберігається на сервері й дозволяє безпечно оновити access-токен після завершення його строку дії без повторної автентифікації паролем. Подібна схема ротації access/refresh-токенів є поширеною практикою, яка поєднує високий рівень безпеки із зручністю використання.

Розмежування доступу реалізовано за моделлю RBAC (Role-Based Access Control): кожному користувачеві призначається одна або кілька ролей, а доступ до конкретних ресурсів API визначається набором дозволених для ролі дій. Це дозволяє централізовано й декларативно керувати правами та легко розширювати рольову модель за потреби.

1.4 Формування вимог до розроблюваної системи

На основі аналізу предметної області, огляду наявних рішень та обраного технологічного стеку сформульовано вимоги до розроблюваної системи. Вимоги поділено на функціональні (що система повинна робити) та нефункціональні (якими властивостями вона має володіти).

Функціональні вимоги. Система повинна забезпечувати:

1) перегляд і пошук у каталозі видань для всіх категорій користувачів, зокрема неавторизованих, з фільтрацією за категорією та статусом наявності, сортуванням і посторінковим виведенням (пагінацією);
2) перегляд детальної інформації про видання з переліком авторів, видавництва, бібліографічних атрибутів і кількості доступних примірників;
3) реєстрацію та автентифікацію користувачів із видачею JWT-токенів;
4) бронювання доступного видання читачем та перегляд читачем власних бронювань і видач;
5) підтвердження й скасування бронювань бібліотекарем;

6) видачу примірника читачеві та повернення примірника із фіксацією дат і відповідальної особи;
7) виявлення та перегляд прострочених видач;
8) адміністрування каталогу: створення, редагування та видалення видань, примірників, авторів, категорій і видавництв (CRUD);
9) керування обліковими записами користувачів та їхніми ролями;
10) відображення зведених показників діяльності (дашборд) з урахуванням ролі користувача.
11) ведення журналу аудиту дій користувачів (хто, яку дію, над яким об'єктом і коли виконав) для забезпечення простежуваності та контролю операцій.

Нефункціональні вимоги. Система повинна:

1. Реалізовувати рольове розмежування доступу для чотирьох категорій користувачів (гість, читач, бібліотекар, адміністратор) із належним захистом ресурсів.
2. Забезпечувати безпеку автентифікації на основі JWT із хешуванням паролів і механізмом оновлення токенів.
3. Мати чітку модульну архітектуру з розділенням серверної та клієнтської частин, що взаємодіють через REST API.
4. Гарантувати цілісність і узгодженість даних засобами реляційної СУБД та транзакцій.
5. Надавати зручний, адаптивний інтерфейс, придатний для роботи як на настільних, так і на мобільних пристроях.
6. Керувати схемою бази даних через версіоновані міграції.
7. Супроводжуватися документацією програмного інтерфейсу (OpenAPI/Swagger).
8. Бути покритою автоматизованими тестами критичної бізнес-логіки.
9. Забезпечувати просте локальне розгортання, зокрема засобами контейнеризації.

Сформульовані вимоги є основою для проєктування системи, що розглядається в наступному розділі.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Архітектура застосунку

Архітектуру системи побудовано за принципом розділення на дві незалежні частини — серверну (backend) і клієнтську (frontend), що взаємодіють виключно через REST API. Такий підхід відповідає сучасній decoupled-архітектурі вебзастосунків і узгоджується з принципами чистої архітектури та предметно-орієнтованого проєктування [27; 28]; він має низку переваг: частини можна розробляти, тестувати, розгортати й масштабувати незалежно; до того самого API можна підключати різні клієнти; межа між частинами чітко визначена контрактом API, що спрощує супровід. Загальну схему взаємодії компонентів наведено на рисунку 2.1.

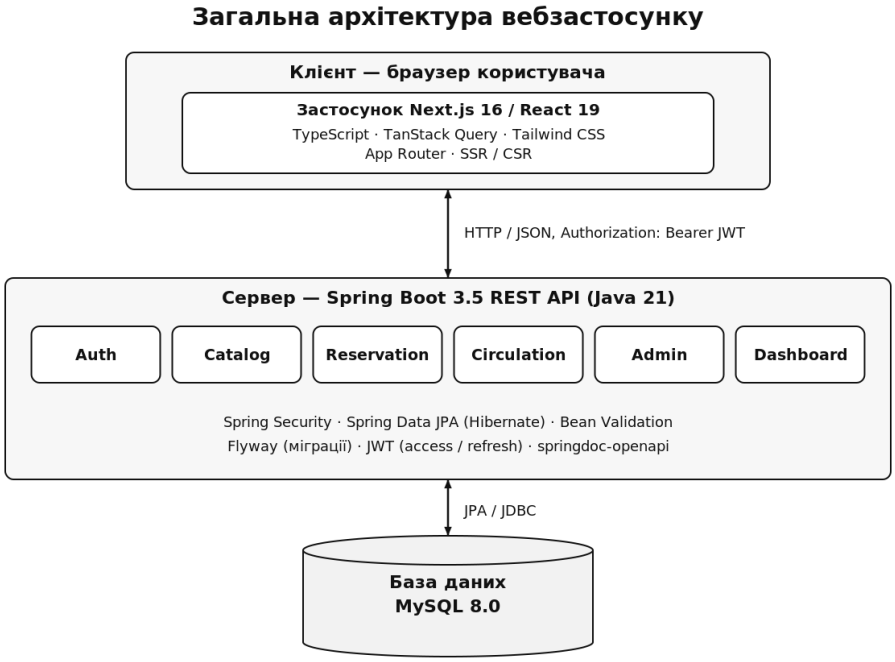


Рисунок 2.1 — *Загальна архітектура вебзастосунку*

Серверна частина побудована за принципом організації коду «package-by-feature» (поділ за функціональними можливостями), на відміну від традиційного поділу «за технічними шарами». Замість того щоб групувати весь код за типом (усі контролери разом, усі сервіси разом тощо), код групується навколо бізнес-можливостей: кожна функціональна область (автентифікація, каталог, бронювання, видача, адміністрування, дашборд) утворює окремий пакет, що містить усі потрібні їй компоненти. Такий підхід підвищує зв'язність (cohesion) усередині модулів, зменшує зв'язаність (coupling) між ними і полегшує орієнтацію в проєкті та його розвиток [29]. Структуру функціональних пакетів серверної частини наведено в таблиці 2.1.

Таблиця 2.1 — *Функціональні пакети серверної частини*

Пакет	**Призначення**
admin	Адміністративний CRUD для довідників, видань, примірників і користувачів
audit	Журнал аудиту дій (модель і репозиторій записів аудиту)
auth	Автентифікація, реєстрація, поточний користувач, ротація refresh-токенів, фільтр bearer-токена
catalog	Публічний каталог видань, автори, категорії, доступність примірників
circulation	Видача, повернення, обробка прострочених видач
dashboard	Зведені показники з урахуванням ролі користувача
reservation	Бронювання читачами та правила підтвердження / скасування персоналом
shared	Спільні компоненти: помилки API, перевірка стану, DTO пагінації, базові сутності
user	Модель користувачів і ролей

Усередині кожного функціонального модуля код організовано за класичним багатошаровим патерном, що забезпечує розділення відповідальності між шарами. Контролер (controller) приймає HTTP-запити, перетворює їх на виклики бізнес-логіки та повертає відповіді у вигляді об'єктів передавання даних (DTO), не оперуючи безпосередньо сутностями бази даних.

Сервіс (service) містить бізнес-логіку та керує транзакціями. Репозиторій (repository) відповідає виключно за доступ до даних. Окремо виділяються моделі-сутності (model/entity), що відображають таблиці бази даних, та об'єкти передавання даних (DTO), що визначають формат обміну з клієнтом. Таке розділення убезпечує внутрішню доменну модель від прямого впливу зовнішнього контракту API і навпаки. Багатошарову структуру модуля наведено на рисунку 2.2.

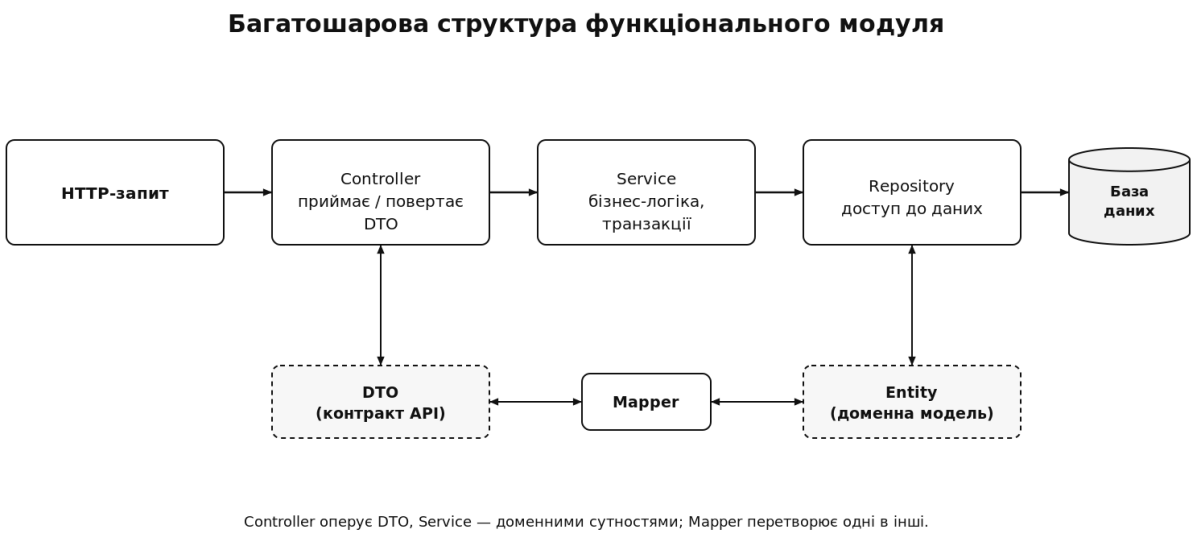


Рисунок 2.2 — *Багатошарова структура функціонального модуля*

Клієнтська частина організована навколо реальних користувацьких сценаріїв і структурно поділена на застосунковий шар (app — маршрути та сторінки App Router), функціональний шар (features — компоненти й логіка конкретних функціональних областей, зокрема робочого простору бібліотеки та каталогу) та спільний шар (shared — централізований API-клієнт, повторно використовувані UI-компоненти, допоміжні функції). Централізація роботи з API в окремому шарі дозволяє єдиним чином обробляти автентифікацію,

додавання токенів до запитів, оновлення простроченого access-токена та обробку помилок.

Обмін даними між частинами відбувається у форматі JSON через HTTP. Кожен захищений запит від клієнта супроводжується заголовком Authorization: Bearer <access-токен>. На сервері спеціальний фільтр перехоплює запит, перевіряє токен і встановлює контекст безпеки, на основі якого механізм авторизації вирішує, чи дозволено виконання операції для ролі поточного користувача. Для коректної роботи клієнта й сервера з різних джерел (різні порти під час розробки) на сервері налаштовано політику CORS.

2.2 Модель даних

2.2.1 Концептуальна модель

Концептуальна модель описує ключові сутності предметної області та зв'язки між ними незалежно від конкретної реалізації в СУБД. На основі аналізу предметної області (підрозділ 1.1) виокремлено такі основні сутності.

Користувач (User) — особа, що взаємодіє із системою. Має набір атрибутів (електронна пошта, ім'я, прізвище, телефон, хешований пароль, статус) і пов'язаний із однією або кількома *ролями (Role)*, які визначають його повноваження. Зв'язок «користувач — роль» є відношенням «багато до багатьох».

Видання (Book) — бібліографічний опис твору з атрибутами назви, ISBN, опису, мови, року видання та кількості сторінок. Видання пов'язане з *видавництвом (Publisher)* відношенням «багато до одного», з *авторами (Author)* — відношенням «багато до багатьох» (один твір може мати кількох авторів, один автор — кілька творів), а також із *категоріями (Category)* — також «багато до багатьох» (видання може належати до кількох тематичних категорій).

Примірник (BookCopy) — конкретна фізична копія видання з власним інвентарним номером, статусом, місцем зберігання та датою надходження.

Пов'язаний із виданням відношенням «багато до одного». Саме примірник є об'єктом операцій видачі.

Бронювання (Reservation) — закріплення читачем доступного видання (та, після підтвердження, конкретного примірника) на визначений строк. Пов'язане з читачем (користувачем), виданням і, опційно, примірником. Має статус (очікує, готове, виконане, скасоване) та позначки часу створення, виконання й скасування.

Видача (Loan) — факт передавання примірника читачеві. Пов'язана з примірником, читачем і бібліотекарем, що оформив видачу. Має статус (видано, повернено, прострочено), дату видачі, дату очікуваного повернення та фактичну дату повернення.

Запис журналу аудиту (AuditLog) — фіксація значущих дій у системі (хто, яку дію, над яким об'єктом і коли виконав) для забезпечення простежуваності.

Окремо виділяється сутність *refresh-токена (RefreshToken)*, що зберігає видані довгоживучі токени оновлення сесії та пов'язана з користувачем.

Концептуальну модель у вигляді діаграми «сутність — зв'язок» (ER-діаграми) наведено на рисунку 2.3.

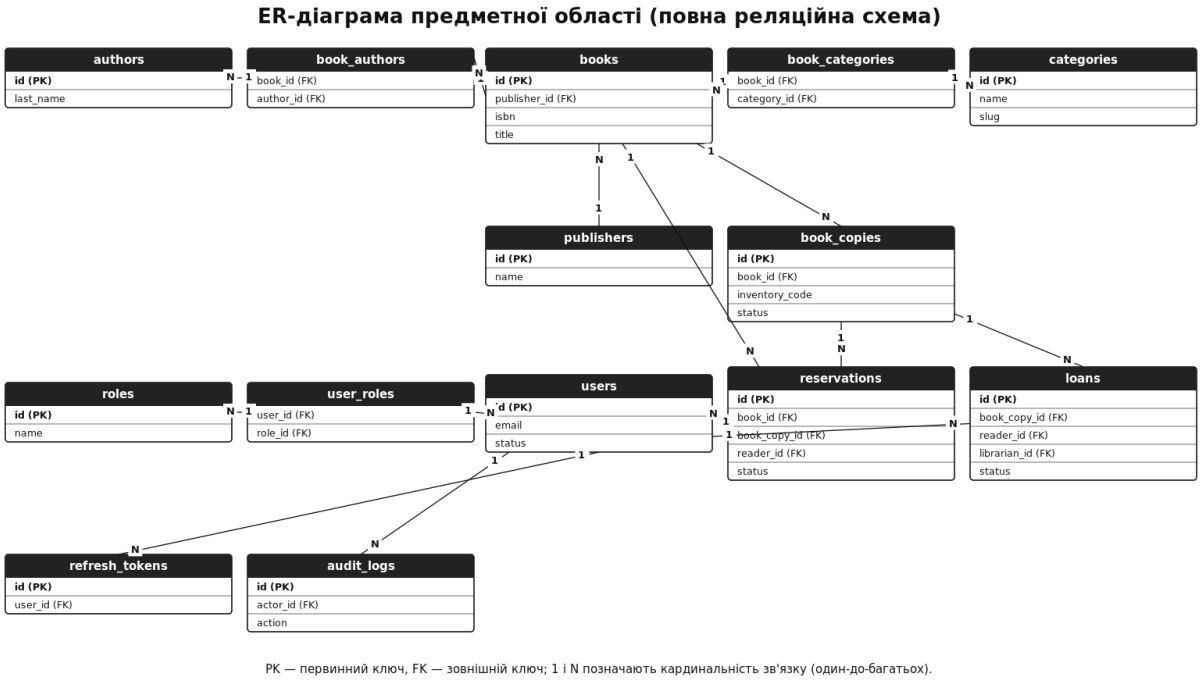


Рисунок 2.3 — *ER-діаграма предметної області*

2.2.2 Структура бази даних

Концептуальну модель реалізовано у вигляді реляційної схеми бази даних MySQL. Схему створено й керовано через версіоновані міграції Flyway. Базова схема описується міграцією V1, додавання таблиці refresh-токенів — міграцією V3, а зв'язування бронювань із конкретними примірниками — міграцією V4. Текст базової міграції схеми наведено в підрозділі 3.1.5 (лістинг 3.6). Перелік основних таблиць бази даних та їх призначення подано в таблиці 2.2.

Таблиця 2.2 — Основні таблиці бази даних

Таблиця	Призначення
roles	Ролі користувачів
users	Облікові записи користувачів
user_roles	Зв'язок користувачів і ролей (багато до багатьох)
authors	Автори

publishers	Видавництва
categories	Тематичні категорії
books	Видання (бібліографічний опис)
book_authors	Зв'язок видань і авторів (багато до багатьох)
book_categories	Зв'язок видань і категорій (багато до багатьох)
book_copies	Фізичні примірники видань
reservations	Бронювання
loans	Видачі
audit_logs	Журнал аудиту дій
refresh_tokens	Токени оновлення сесії

Розглянемо ключові таблиці детальніше. Таблиця users містить ідентифікатор, унікальну електронну пошту (обмеження унікальності uk_users_email), хеш пароля, ім'я, прізвище, телефон, статус та позначки часу створення й оновлення. Зв'язок з ролями реалізовано через сполучну таблицю user_roles із складеним первинним ключем і зовнішніми ключами на users і roles.

Таблиця `books` зберігає бібліографічний опис видання та має зовнішній ключ на `publishers` і обмеження унікальності за полем `isbn`. Зв'язки «багато до багатьох» з авторами та категоріями реалізовано через сполучні таблиці `book_authors` і `book_categories`. Таблиця `book_copies` зберігає примірники з унікальним інвентарним номером (`uk_book_copies_inventory_code`), статусом і зовнішнім ключем на `books`.

Таблиця `reservations` пов'язана з виданням, читачем і (після міграції V4) з конкретним примірником; містить статус та позначки часу бронювання, строку дії, виконання й скасування. Таблиця `loans` пов'язана з примірником, читачем і бібліотекарем, містить статус, дату видачі, дату очікуваного

повернення й фактичну дату повернення. Для пришвидшення типових запитів створено індекси за назвою видання, статусами примірників, бронювань і видач, датою повернення видачі та датою створення записів аудиту.

Реляційні обмеження (зовнішні ключі, унікальність) забезпечують цілісність даних на рівні СУБД, унеможливлюючи створення «осиротілих» або суперечливих записів. У поєднанні з транзакційністю бізнес-операцій на рівні сервісів це гарантує узгодженість даних навіть за умови одночасних звернень.

2.3 Проектування REST API

Програмний інтерфейс системи спроєктовано відповідно до принципів REST. Ресурси згруповано за функціональними областями, кожна з яких має власний префікс URL у межах базового шляху /api. Дії над ресурсами виконуються відповідними методами HTTP, а результат повертається у форматі JSON із відповідним кодом стану HTTP (200 — успіх, 201 — створено, 400 — некоректний запит, 401 — не автентифіковано, 403 — недостатньо прав, 404 — не знайдено, 409 — конфлікт). Для одноманітної обробки помилок спроєктовано єдину структуру відповіді про помилку (ApiError) та централізований обробник винятків.

Виокремлено такі групи ресурсів API: автентифікація (/api/auth), публічний каталог (/api/catalog), бронювання (/api/reservations), обіг видань (/api/circulation), адміністрування каталогу (/api/admin/catalog), адміністрування користувачів (/api/admin/users), зведені показники (/api/dashboard) та перевірка стану (/api/health). Розмежування доступу до груп ресурсів реалізовано так: ендпоінти автентифікації (логін, реєстрація, оновлення токена) і публічний каталог доступні без авторизації; ендпоінти бронювань і обігу доступні автентифікованим користувачам із відповідними ролями; адміністративні ендпоінти доступні лише ролі адміністратора.

Проєктований розподіл доступу за групами ресурсів наведено в таблиці 2.3, а повну зведену таблицю ендпоінтів наведено в таблиці 3.5 (підрозділ 3.1.4).

Таблиця 2.3 — *Групи ресурсів REST API та рівень доступу*

Група	**Базовий шлях**	**Доступ**
Автентифікація	/api/auth	Публічний (login, register, refresh); автентифікований (me)
Каталог	/api/catalog	Публічний (GET)
Бронювання	/api/reservations	Читач (створення, власні); бібліотекар / адміністратор (перегляд усіх, підтвердження, скасування)
Обіг видань	/api/circulation	Читач (власні видачі); бібліотекар / адміністратор (видача, повернення, прострочення)
Адміністрування каталогу	/api/admin/catalog	Адміністратор
Адміністрування користувачів	/api/admin/users	Адміністратор
Дашборд	/api/dashboard	Автентифікований (показники залежать від ролі)
Перевірка стану	/api/health	Публічний

Під час проєктування API дотримано кількох принципів. По-перше, контролери оперують об'єктами передавання даних (DTO), а не сутностями бази даних, що відокремлює зовнішній контракт від внутрішньої моделі та убезпечує від випадкового розкриття зайвих полів. По-друге, для операцій, що повертають великі набори даних (зокрема каталог видань), передбачено пагінацію, сортування та фільтрацію через параметри запиту; для стабільності

контракту відповідь пагінації повертається через власний DTO (PageResponse), а не через стандартний серіалізований об'єкт сторінки фреймворку. По-третє, вхідні дані валідуються декларативно за допомогою анотацій Bean Validation, а порушення валідації перетворюються на структуровану відповідь про помилку.

Як приклад розглянемо проєктування ключового бізнес-сценарію — повного циклу обслуговування. Читач надсилає запит на бронювання доступного видання (POST /api/reservations). Бібліотекар переглядає перелік бронювань (GET /api/reservations) і підтверджує потрібне (POST /api/reservations/{id}/approve), після чого за бронюванням закріплюється доступний примірник. Далі бібліотекар оформлює видачу (POST /api/circulation/loans), що змінює статус примірника на «виданий» і створює запис видачі. Після повернення видання бібліотекар оформлює повернення (POST /api/circulation/loans/{id}/return), що повертає примірник у фонд. Кожна з цих операцій (бронювання, видача, повернення) додатково фіксується у таблиці audit_logs — для забезпечення повного журналу дій і можливості аудиту. Діаграму послідовності цього сценарію наведено на рисунку 2.5.



Рисунок 2.5 — *Діаграма послідовності повного циклу обслуговування*

Документування API виконується автоматично засобами springdoc-openapi: на основі анотацій контролерів формується специфікація OpenAPI, доступна за шляхом /v3/api-docs, та інтерактивний інтерфейс Swagger UI [11] за шляхом /swagger-ui.html, який дозволяє переглядати й випробовувати ендпоінти безпосередньо в браузері.

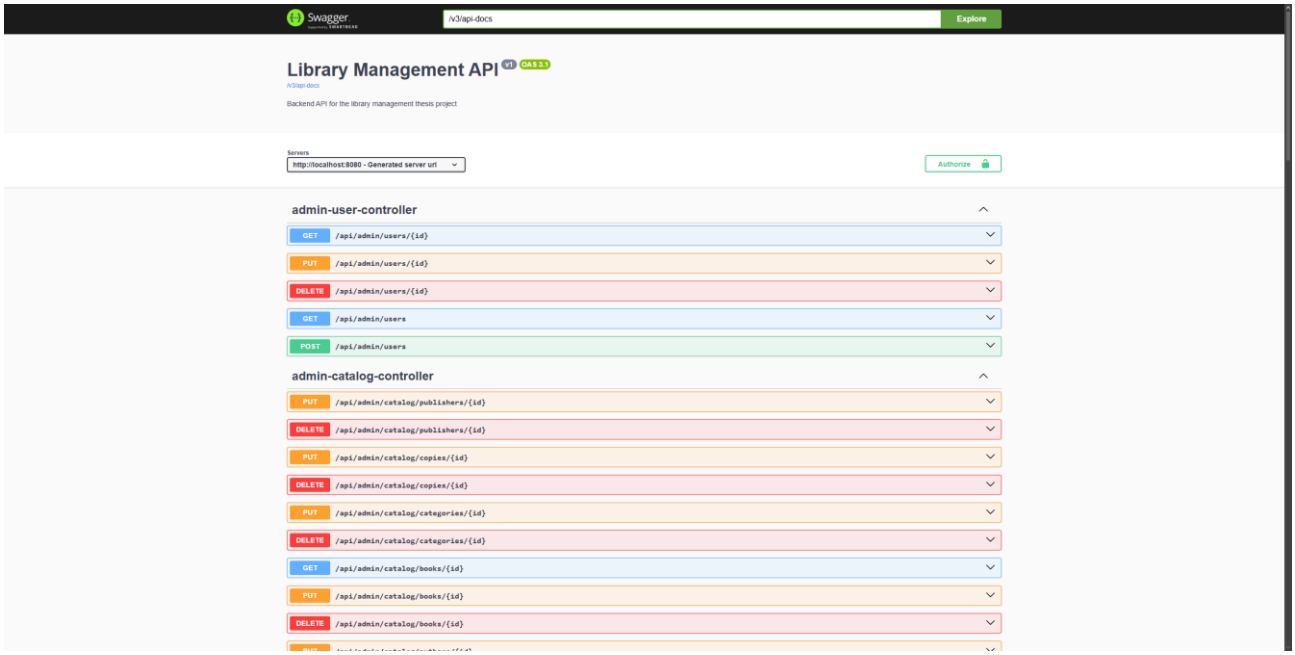


Рисунок 2.6 — *Документація REST API в інтерфейсі Swagger UI*

2.4 Проектування інтерфейсу користувача

Інтерфейс користувача спроєктовано виходячи з реальних сценаріїв роботи різних ролей, а не навколо декоративного оформлення. Основою інтерфейсу є концепція єдиного робочого простору (workspace), у якому залежно від ролі користувача та обраного розділу відображається відповідний набір даних і дій. Такий підхід забезпечує цілісність досвіду та зменшує кількість окремих сторінок.

Інформаційну структуру застосунку організовано навколо кількох ключових екранів: операційний огляд (зведені показники та поточні операції), каталог видань (пошук, фільтрація, перегляд), робочий простір читача (бронювання й видачі), робочий простір бібліотекаря (підтвердження бронювань, видача, повернення, прострочення) та адміністративна панель (керування каталогом, довідниками й користувачами). Навігацію між розділами винесено в бічну панель, а дії, пов'язані з обліковим записом (вхід, оновлення, сповіщення, вихід), згруповано у верхній панелі. Схему навігації наведено на рисунку 2.7.



Рисунок 2.7 — *Карта екранів застосунку*

Під час проєктування інтерфейсу враховано необхідність опрацювання різних станів кожного екрана: стан завантаження даних, стан помилки, порожній стан (відсутність даних) та основний стан із даними. Це підвищує якість досвіду користувача й узгодженість поведінки інтерфейсу. Дії, що змінюють дані або є незворотними (бронювання, скасування, повернення видання, видалення записів), супроводжуються діалогами підтвердження, а

результат операцій — спливними сповіщеннями (toast). Форми створення й редагування записів реалізовано в модальних вікнах, що дозволяє виконувати дії, не залишаючи поточного контексту.

Візуальне рішення інтерфейсу побудовано в мінімалістичному монохромному (чорно-білому) стилі з акцентом на читабельність, контрастність і чисту типографіку. Інтерфейс адаптивний: верстку реалізовано так, щоб коректно відображатися як на широких екранах настільних комп'ютерів, так і на вузьких екранах мобільних пристроїв, без горизонтального прокручування. Загальний візуальний стиль і компонування основних екранів подано на прототипах (рисунки 2.8–2.9).

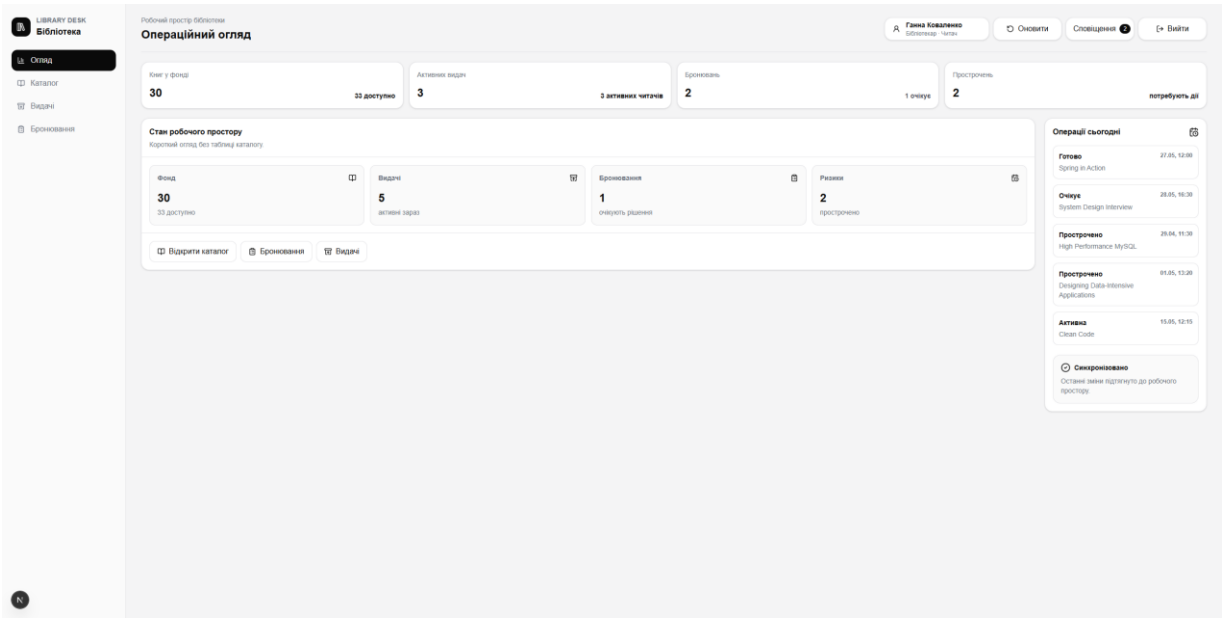


Рисунок 2.8 — *Макет головного екрана (операційний огляд)*

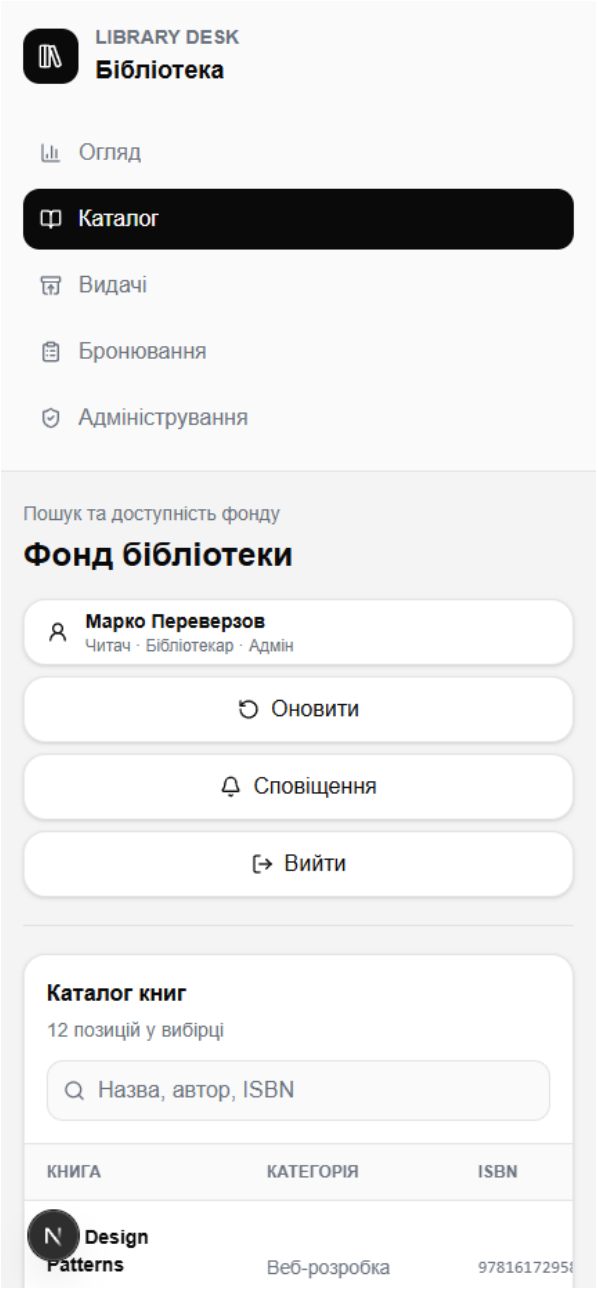


Рисунок 2.9 — *Адаптивний інтерфейс каталогу на мобільному пристрої*

Спроєктовані архітектура, модель даних, програмний інтерфейс та інтерфейс користувача утворюють завершений проєкт системи, реалізацію якого описано в наступному розділі.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація серверної частини (Spring Boot)

3.1.1 Налаштування проекту

Серверну частину створено на основі Spring Boot 4.0.6 та Java 25 із керуванням залежностями через Maven. Структуру залежностей проєкту визначено у файлі pom.xml; до складу проєкту включено стартери Spring Web, Spring Security, Spring Data JPA та Bean Validation, бібліотеку Flyway для міграцій, драйвер MySQL, бібліотеку jjwt для роботи з JWT, springdoc-openapi для документування API, Lombok [30] для скорочення шаблонного коду, MapStruct [31] для відображення об'єктів, а також H2 і spring-boot-starter-test для тестування. Перелік основних залежностей наведено в таблиці 3.1.

Таблиця 3.1 — *Основні залежності серверної частини*

Залежність	Призначення
spring-boot-starter-web	REST-контролери, обробка HTTP-запитів
spring-boot-starter-data-jpa	Доступ до даних, ORM (Hibernate)
spring-boot-starter-security	Автентифікація й авторизація
spring-boot-starter-validation	Валідація вхідних даних
flyway-core, flyway-mysql	Версіоновані міграції бази даних
mysql-connector-j	Драйвер MySQL
jjwt-api / impl / jackson	Формування й перевірка JWT
springdoc-openapi-starter-webmvc-ui	Документація OpenAPI / Swagger UI
lombok	Генерація шаблонного коду
mapstruct	Відображення DTO
h2, spring-boot-starter-test, spring-security-test	Тестування

Конфігурацію застосунку винесено у файл application.yml. У ньому визначено параметри підключення до бази даних (з можливістю перевизначення через змінні середовища), увімкнення Flyway, режим роботи

JPA (ddl-auto: validate — лише перевірка відповідності схеми без її зміни), шляхи OpenAPI/Swagger та параметри підсистеми безпеки JWT (видавець токена, секретний ключ, строк дії access-токена — 20 хвилин, строк дії refresh-токена — 14 днів). Винесення чутливих і середовищезалежних параметрів у змінні середовища відповідає принципам конфігурування промислових застосунків. Фрагмент конфігураційного файлу наведено нижче (лістинг 3.1).

Лістинг 3.1 — *Конфігурація серверної частини (application.yml)*

```
server:
  port: ${SERVER_PORT:8080}

spring:
  application:
    name: library-backend
  datasource:
    url:
${SPRING_DATASOURCE_URL:jdbc:mysql://localhost:3306/library
_db?createDatabaseIfNotExist=true&serverTimezone=UTC}
    username: ${SPRING_DATASOURCE_USERNAME:library_user}
    password:
${SPRING_DATASOURCE_PASSWORD:library_password}
  flyway:
    enabled: true
    locations: classpath:db/migration
  jpa:
    open-in-view: false
    hibernate:
      ddl-auto: validate
    properties:
      hibernate:
        format_sql: true
```

```
springdoc:
  api-docs:
    path: /v3/api-docs
  swagger-ui:
    path: /swagger-ui.html

app:
  security:
    jwt:
      issuer: library-backend
      secret: ${JWT_SECRET:local-development-secret-key-
for-library-backend-please-change-me-2026}
      access-token-minutes: ${JWT_ACCESS_TOKEN_MINUTES:20}
      refresh-token-days: ${JWT_REFRESH_TOKEN_DAYS:14}
```

Загальну структуру пакетів серверної частини організовано за принципом «package-by-feature» (підрозділ 2.1): кореневий пакет com.markpereverzov.library містить функціональні модулі auth, catalog, reservation, circulation, admin, dashboard, user та спільний модуль shared. Структуру каталогів проєкту наведено на рисунку 3.1.



Рисунок 3.1 — *Структура пакетів серверної частини*

3.1.2 Реалізація шару сутностей (Entity)

Доменну модель реалізовано у вигляді JPA-сутностей, що відображають таблиці бази даних. Спільні для всіх сутностей поля (ідентифікатор та позначки часу створення й оновлення) винесено в базовий клас BaseEntity, від якого успадковуються всі сутності. Це усуває дублювання та забезпечує єдиний підхід до аудиту часу змін.

Клас BaseEntity позначено анотацією @MappedSuperclass (специфікація JPA [10]), що вказує постачальнику JPA (Hibernate) включати поля класу до схеми таблиці кожного нащадка, не створюючи для нього окремої таблиці. Поле id позначено @Id та @GeneratedValue(strategy = GenerationType.IDENTITY) — первинний ключ генерується автоматично засобами бази даних (auto increment). Поля createdAt і updatedAt анотовано @Column(nullable = false, updatable = false) та @Column(nullable = false) відповідно; вони заповнюються автоматично зворотними викликами @PrePersist (перед першим збереженням) та @PreUpdate (перед кожним оновленням) через методи onCreate() і onUpdate(), що встановлюють поточний момент часу через Instant.now(). Фрагмент коду базової сутності наведено нижче:

```
@Getter @Setter
@MappedSuperclass
public abstract class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "created_at", nullable = false, updatable = false)
    private Instant createdAt;

    @Column(name = "updated_at", nullable = false)
    private Instant updatedAt;

    @PrePersist
    protected void onCreate() {
        Instant now = Instant.now();
        createdAt = now;
        updatedAt = now;
    }

    @PreUpdate
    protected void onUpdate() { updatedAt = Instant.now(); }
}
```

Розглянемо як приклад сутність Book (видання). Її анотовано @Entity та @Table(name = "books") — це позначає клас як JPA-сутність і прив'язує його до конкретної таблиці бази даних. Анотації Lombok [30] @Getter, @Setter, @Builder, @NoArgsConstructor, @AllArgsConstructor автоматично генерують методи доступу, конструктори й будівельник. Поля сутності відповідають

стовпцям таблиці: title (nullable = false), isbn (nullable = false, unique = true), description, language, publicationYear, pageCount. Зв'язок із видавництвом описано анотацією @ManyToOne(fetch = FetchType.LAZY) з @JoinColumn(name = "publisher_id") — це однонаправлений зв'язок із боку Book. Зв'язки з авторами та категоріями описано @ManyToMany(fetch = FetchType.LAZY) з @JoinTable, що явно вказує назву сполучної таблиці (book_authors, book_categories) та стовпці зовнішніх ключів — також з боку Book (однонаправлений). Зв'язок з примірниками @OneToMany(mappedBy = "book") — зворотний бік двонаправленого зв'язку, де mappedBy вказує, що BookCopy.book є власником зв'язку. Для всіх зв'язків застосовано «ліниве» завантаження (FetchType.LAZY), що завантажує пов'язані дані лише за потреби й запобігає надмірним запитам до бази. Повний текст класу Book наведено нижче (лістинг 3.2).

Лістинг 3.2 — *JPA-сутність видання (Book.java)*

```
package com.markpereverzov.library.catalog.model;

import com.markpereverzov.library.shared.model.BaseEntity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;
```

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "books")
public class Book extends BaseEntity {

    @Column(nullable = false, length = 240)
    private String title;

    @Column(nullable = false, unique = true, length = 20)
    private String isbn;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(nullable = false, length = 40)
    private String language;

    @Column(name = "publication_year")
    private Integer publicationYear;

    @Column(name = "page_count")
    private Integer pageCount;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "publisher_id")
    private Publisher publisher;

    @Builder.Default
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
            name = "book_authors",
            joinColumns = @JoinColumn(name = "book_id"),
            inverseJoinColumns = @JoinColumn(name =
"author_id")
    )
    private Set<Author> authors = new LinkedHashSet<>();

    @Builder.Default
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
            name = "book_categories",
            joinColumns = @JoinColumn(name = "book_id"),
            inverseJoinColumns = @JoinColumn(name =
"category_id")
    )
    private Set<Category> categories = new
LinkedHashSet<>();

    @Builder.Default
    @OneToMany(mappedBy = "book")
    private List<BookCopy> copies = new ArrayList<>();
}
```

Статуси примірників, бронювань і видач реалізовано через перелічення (enum): BookCopyStatus (доступний, виданий, зарезервований тощо), ReservationStatus (очікує, готове до видачі, виконане, скасоване), LoanStatus

(активна, повернена, прострочена). Використання перелічень замість рядкових констант підвищує типобезпеку та зменшує ймовірність помилок, що відповідає рекомендованим практикам мови Java [32]. Шар доступу до даних реалізовано засобами Spring Data JPA [7; 14]. Для кожної кореневої сутності створено інтерфейс-репозиторій, що успадковує JpaRepository, завдяки чому без написання коду стають доступними стандартні CRUD-операції, посторінкове отримання даних (через Pageable) і сортування. Spring Data JPA під час старту застосунку автоматично генерує реалізації цих інтерфейсів як Spring-біни, тому окремих класів-реалізацій репозиторіїв писати не потрібно. Специфічні вибірки оголошуються декларативно — похідними методами запитів за назвами (наприклад, findByReaderIdOrderByLoanedAtDesc, findByStatusAndDueAtBefore), які Spring Data JPA транслює у відповідні SQL-запити на основі імені методу. Для випадків, що потребують одночасного завантаження пов'язаних сутностей, застосовано спеціалізовані методи з вибіркою деталей (наприклад, findWithDetailsById), а для динамічного пошуку й фільтрації каталогу за змінним набором критеріїв — механізм специфікацій (Specification) JPA Criteria API, що дозволяє програмно комбінувати умови запиту залежно від переданих параметрів. Перелік репозиторіїв за модулями наведено в коді проєкту; усі вони містять лише оголошення доступу до даних і не містять бізнес-логіки, яку зосереджено в сервісному шарі.

3.1.3 Реалізація бізнес-логіки (сервіси)

Бізнес-логіку зосереджено в сервісному шарі. Сервіси інкапсулюють правила предметної області та керують транзакціями за допомогою анотації @Transactional (специфікація Jakarta Transactions, що входить до Jakarta EE [10]). Ця анотація гарантує атомарність складних операцій: якщо в тілі методу виникає виняток, всі зміни бази даних автоматично відкочуються (rollback), забезпечуючи принцип ACID — жодна з часткових змін не буде збережена. На рівні Spring Framework анотація реалізується через AOP-проксі навколо методу: перед викликом відкривається транзакція, після успішного

завершення фіксуються зміни (commit), а у разі RuntimeException — виконується rollback. Розглянемо реалізацію ключових бізнес-процесів.

Сервіс каталогу (CatalogService) реалізує пошук видань із динамічною фільтрацією за пошуковим рядком (за назвою, ISBN або автором), категорією та статусом наявності, а також сортування й пагінацію. Для побудови динамічних запитів застосовано механізм специфікацій JPA, що дозволяє гнучко комбінувати критерії фільтрації залежно від переданих параметрів.

Найскладнішу бізнес-логіку містить сервіс обігу видань (CirculationService), що відповідає за видачу й повернення (лістинг 3.3). Метод видачі issue(CreateLoanRequest request, Long librarianId) отримує об'єкт CreateLoanRequest з полями reservationId (необов'язковий — ідентифікатор попереднього бронювання), copyId (ідентифікатор примірника для прямої видачі), readerId (ідентифікатор читача), а також ідентифікатор бібліотекаря, що оформлює видачу. Метод повертає LoanDto — об'єкт передавання даних з відомостями про щойно створену видачу. Метод реалізує дві гілки сценарію. У разі видачі за бронюванням сервіс перевіряє, що бронювання належить зазначеному читачеві та перебуває у статусі «готове до видачі», бере закріплений за бронюванням примірник, перевіряє його статус «зарезервований», після чого переводить бронювання у статус «виконане». У разі прямої видачі (без бронювання) сервіс перевіряє наявність і доступність зазначеного примірника. У обох випадках статус примірника змінюється на «виданий», створюється запис видачі зі строком повернення (за замовчуванням — 21 день), а всі перевірки супроводжуються поверненням відповідних кодів стану HTTP (404 — не знайдено, 409 — конфлікт станів). Метод повернення (returnLoan) реалізовано ідемпотентно: повторне повернення вже поверненої видачі не призводить до помилки, а коректне повернення переводить видачу у статус «повернена», фіксує дату й повертає примірник у фонд. Метод виявлення прострочень (overdueLoans) знаходить активні видачі зі сплилим строком і переводить їх у статус «прострочена». Текст класу CirculationService наведено нижче (лістинг 3.3).

Лістинг 3.3 — *Сервіс обігу видань (CirculationService.java)*

```
@Service
@RequiredArgsConstructor
public class CirculationService {

    private final LoanRepository loanRepository;
    private final ReservationRepository
reservationRepository;
    private final BookCopyRepository bookCopyRepository;
    private final UserRepository userRepository;

    @Transactional
    public LoanResponse issue(CreateLoanRequest request,
LibraryUserPrincipal principal) {
        AppUser reader =
userRepository.findById(request.readerId())
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Reader not
found"));
        AppUser librarian =
userRepository.findById(principal.id())
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Librarian
not found"));
        Instant now = Instant.now();
        BookCopy copy;

        if (request.reservationId() != null) {
            Reservation reservation =
reservationRepository.findWithDetailsById(request.reservati
onId())
```

```
                    .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Reservation
not found"));
            if
(!reservation.getReader().getId().equals(reader.getId())) {
                throw new
ResponseStatusException(HttpStatus.CONFLICT, "Reservation
reader does not match loan reader");
            }
            if (reservation.getStatus() !=
ReservationStatus.READY_FOR_PICKUP) {
                throw new
ResponseStatusException(HttpStatus.CONFLICT, "Reservation
must be ready for pickup before issuing");
            }
            copy = reservation.getBookCopy();
            if (copy == null || copy.getStatus() !=
BookCopyStatus.RESERVED) {
                throw new
ResponseStatusException(HttpStatus.CONFLICT, "Reserved copy
is not available for issuing");
            }

reservation.setStatus(ReservationStatus.FULFILLED);
            reservation.setFulfilledAt(now);
        } else {
            if (request.bookCopyId() == null) {
                throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "bookCopyId
is required when reservationId is not provided");
            }
            copy =
bookCopyRepository.findById(request.bookCopyId())
```

```
                    .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Book copy
not found"));
            if (copy.getStatus() !=
BookCopyStatus.AVAILABLE) {
                throw new
ResponseStatusException(HttpStatus.CONFLICT, "Book copy is
not available for issuing");
            }
        }

        copy.setStatus(BookCopyStatus.LOANED);
        Loan loan = Loan.builder()

.bookCopy(copy).reader(reader).librarian(librarian)
                .status(LoanStatus.ACTIVE).loanedAt(now)
                .dueAt(request.dueAt() == null ?
now.plus(21, ChronoUnit.DAYS) : request.dueAt())
                .build();
        return toResponse(loanRepository.save(loan));
    }

    @Transactional
    public LoanResponse returnLoan(Long id,
LibraryUserPrincipal principal) {
        Loan loan = loanRepository.findWithDetailsById(id)
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Loan not
found"));
        if (loan.getStatus() == LoanStatus.RETURNED) {
            return toResponse(loan);
        }
```

```
        if (loan.getStatus() != LoanStatus.ACTIVE &&
loan.getStatus() != LoanStatus.OVERDUE) {
            throw new
ResponseStatusException(HttpStatus.CONFLICT, "Loan cannot
be returned in current status");
        }
        loan.setStatus(LoanStatus.RETURNED);
        loan.setReturnedAt(Instant.now());

loan.getBookCopy().setStatus(BookCopyStatus.AVAILABLE);
        return toResponse(loan);
    }

    // Методи myLoans(), findLoans(), overdueLoans() та
допоміжний toResponse()
    // реалізовано у вихідному коді проєкту; тут опущено
задля стислості.
}
```

Сервіс бронювань (ReservationService) реалізує створення бронювання читачем із закріпленням доступного примірника, перегляд бронювань, підтвердження бронювання бібліотекарем (переведення у статус «готове до видачі») та скасування бронювання з вивільненням примірника. Сервіс адміністрування (AdminCatalogService, AdminUserService) реалізує операції CRUD над довідниками, виданнями, примірниками й користувачами з перевірками унікальності, обробкою конфліктів реляційних обмежень і безпечним архівуванням користувачів. Сервіс дашборду (DashboardService) обчислює зведені показники з урахуванням ролі користувача: загальну кількість видань і примірників, кількість доступних примірників, активних і прострочених видач, бронювань, що очікують і готові до видачі, та активних читачів.

3.1.4 Реалізація REST-контролерів

REST-контролери утворюють зовнішній інтерфейс серверної частини. Кожен контролер відповідає за окрему групу ресурсів і делегує виконання бізнес-логіки відповідному сервісу, приймаючи й повертаючи виключно об'єкти передавання даних (DTO). Як приклад розглянемо контролер каталогу (CatalogController) — публічний, не потребує автентифікації. Він обробляє такі маршрути: GET /api/catalog/books — отримання сторінки видань із параметрами пошуку (query), фільтрації за категорією (categoryId) та статусом наявності (available), а також пагінацією і сортуванням; GET /api/catalog/books/{id} — отримання детальних відомостей про видання; GET /api/catalog/authors та GET /api/catalog/categories — довідникові переліки для UI-фільтрів. Текст контролера наведено нижче (лістинг 3.4).

Лістинг 3.4 — *REST-контролер каталогу (CatalogController.java)*

```
package com.markpereverzov.library.catalog.controller;

import
com.markpereverzov.library.catalog.dto.AuthorSummaryRespons
e;
import
com.markpereverzov.library.catalog.dto.BookDetailsResponse;
import
com.markpereverzov.library.catalog.dto.BookSummaryResponse;
import
com.markpereverzov.library.catalog.dto.CategoryResponse;
import
com.markpereverzov.library.catalog.model.BookCopyStatus;
import
com.markpereverzov.library.catalog.service.CatalogService;
import com.markpereverzov.library.shared.dto.PageResponse;
import java.util.List;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.PageableDefault;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestParam;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/catalog")
public class CatalogController {

    private final CatalogService catalogService;

    @GetMapping("/books")
    public PageResponse<BookSummaryResponse> books(
            @RequestParam(required = false) String search,
            @RequestParam(required = false) Long
categoryId,
            @RequestParam(required = false) BookCopyStatus
status,
            @PageableDefault(size = 12, sort = "title")
Pageable pageable
    ) {
        return catalogService.findBooks(search, categoryId,
status, pageable);
    }
```

```
    @GetMapping("/books/{id}")
    public BookDetailsResponse book(@PathVariable Long id)
{
        return catalogService.getBook(id);
    }

    @GetMapping("/categories")
    public List<CategoryResponse> categories() {
        return catalogService.getCategories();
    }

    @GetMapping("/authors")
    public List<AuthorSummaryResponse> authors() {
        return catalogService.getAuthors();
    }
}
```

Контролери інших модулів побудовано аналогічно. Контролер автентифікації (AuthController, маршрути /api/auth/**) обробляє реєстрацію (POST /api/auth/register), вхід (POST /api/auth/login), оновлення токена (POST /api/auth/refresh) й отримання даних поточного користувача (GET /api/auth/me). Контролер бронювань (ReservationController, /api/reservations/**) дозволяє читачеві створювати бронювання (POST) і переглядати власні (GET /api/reservations/my), а бібліотекарю — переглядати всі бронювання, підтверджувати (POST /api/reservations/{id}/approve) і скасовувати (POST /api/reservations/{id}/cancel). Контролер обігу (CirculationController, /api/circulation/**) надає бібліотекарю функції видачі (POST /api/circulation/loans), повернення (POST /api/circulation/loans/{id}/return) та обробки прострочень (POST /api/circulation/loans/process-overdue). Адміністративні контролери (AdminCatalogController, AdminUserController) надають CRUD-операції, доступні лише адміністратору.

Підсистему безпеки реалізовано на основі Spring Security [9]. Конфігурацію (SecurityConfig) налаштовано на безстановий режим (без серверних сесій): відключено CSRF-захист (CSRF — Cross-Site Request Forgery, підроблення міжсайтового запиту) — бо REST API не використовує cookie-сесій і вразливість CSRF до нього незастосовна; налаштовано CORS (Cross-Origin Resource Sharing — спільне використання ресурсів між різними джерелами) для дозволу запитів із домену клієнтської частини (http://localhost:3000 в режимі розробки); визначено правила доступу до ресурсів: ендпоінти /api/auth/**, публічний каталог /api/catalog/**, документація /swagger-ui/** та перевірка стану /actuator/health доступні без авторизації, решта запитів потребують автентифікації. Перед стандартним фільтром автентифікації додано власний фільтр JwtAuthenticationFilter, який перехоплює запити, витягує JWT із заголовка Authorization: Bearer, перевіряє його за допомогою сервісу JwtTokenService і встановлює контекст безпеки Spring Security. Розмежування доступу за ролями реалізовано декларативно за допомогою методної безпеки (@EnableMethodSecurity) з анотаціями @PreAuthorize на конкретних методах контролерів. Паролі зберігаються у вигляді хешів, обчислених алгоритмом BCrypt. Помилки авторизації коректно перетворюються на відповіді з кодом 403, а не на загальну помилку сервера. Конфігурацію безпеки наведено нижче (лістинг 3.5).

Лістинг 3.5 — Конфігурація безпеки (SecurityConfig.java)

```
@Configuration @EnableWebSecurity @EnableMethodSecurity
@RequiredArgsConstructor
@EnableConfigurationProperties(JwtProperties.class) public
class SecurityConfig {
private final JwtAuthenticationFilter
jwtAuthenticationFilter;

@Bean
```

```
public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
    return http
            .csrf(AbstractHttpConfigurer::disable)
        .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
            .formLogin(AbstractHttpConfigurer::disable)
            .httpBasic(AbstractHttpConfigurer::disable)
            .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELE
SS))
            .authorizeHttpRequests(auth -> auth
            .requestMatchers(HttpMethod.POST,
"/api/auth/login", "/api/auth/register",
"/api/auth/refresh").permitAll()
                    .requestMatchers(HttpMethod.GET,
"/api/health").permitAll()
                    .requestMatchers(HttpMethod.GET,
"/api/catalog/**").permitAll()
                    .requestMatchers("/v3/api-docs/**",
"/swagger-ui/**", "/swagger-ui.html").permitAll()
                    .anyRequest().authenticated())
        .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
            .build();
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
```

```
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new
CorsConfiguration();

configuration.setAllowedOrigins(List.of("http://localhost:3
000", "http://localhost:3001"));
    configuration.setAllowedMethods(List.of("GET", "POST",
"PUT", "PATCH", "DELETE", "OPTIONS"));

configuration.setAllowedHeaders(List.of("Authorization",
"Content-Type"));

configuration.setExposedHeaders(List.of("Authorization"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);

    return source;
}

}
```

Уніфіковану обробку помилок реалізовано через глобальний обробник винятків (GlobalExceptionHandler), що перехоплює винятки (зокрема порушення валідації та бізнес-конфлікти) і перетворює їх на структуровану відповідь ApiError з відповідним кодом стану HTTP. Це забезпечує передбачуваний для клієнта формат помилок.

Реалізований програмний інтерфейс охоплює усі групи ресурсів, визначені під час проєктування (підрозділ 2.3): автентифікацію, публічний каталог, бронювання, обіг видань та адміністрування. Зведений перелік усіх

реалізованих ендпоінтів із зазначенням HTTP-методу, шляху, призначення та рівня доступу наведено в таблиці 3.5.

Таблиця 3.5 — *Зведений перелік ендпоінтів REST API*

№	Метод	Шлях	Призначення	Доступ
1	POST	/api/auth/register	Реєстрація нового користувача	Публічний
2	POST	/api/auth/login	Вхід, видача access / refresh токенів	Публічний
3	POST	/api/auth/refresh	Оновлення access-токена	Публічний
4	GET	/api/auth/me	Дані поточного користувача	Автентифікований
5	GET	/api/catalog/books	Перелік видань (пошук, фільтри, пагінація)	Публічний
6	GET	/api/catalog/books/{id}	Деталі видання	Публічний
7	GET	/api/catalog/categories	Перелік категорій	Публічний
8	GET	/api/catalog/authors	Перелік авторів	Публічний
9	POST	/api/reservations	Створення бронювання	Читач
10	GET	/api/reservations/my	Власні бронювання читача	Читач
11	GET	/api/reservations	Перелік усіх бронювань	Бібліотекар, Адміністратор
12	POST	/api/reservations/{id}/approve	Підтвердження бронювання	Бібліотекар, Адміністратор

13	POST	/api/reservations/{id}/cancel	Скасування бронювання	Читач (власні), Бібліотекар, Адміністратор
14	POST	/api/circulation/loans	Видача примірника	Бібліотекар, Адміністратор
15	GET	/api/circulation/loans/my	Власні видачі читача	Читач
16	GET	/api/circulation/loans	Перелік усіх видач	Бібліотекар, Адміністратор
17	GET	/api/circulation/loans/overdue	Перелік прострочених видач	Бібліотекар, Адміністратор
18	POST	/api/circulation/loans/{id}/return	Повернення видачі	Бібліотекар, Адміністратор
19	*	/api/admin/catalog/books	CRUD видань	Адміністратор
20	*	/api/admin/catalog/copies	CRUD примірників	Адміністратор
21	*	/api/admin/catalog/authors	CRUD авторів	Адміністратор
22	*	/api/admin/catalog/categories	CRUD категорій	Адміністратор
23	*	/api/admin/catalog/publishers	CRUD видавництв	Адміністратор
24	GET	/api/admin/users	Перелік користувачів	Адміністратор
25	POST	/api/admin/users	Створення користувача	Адміністратор
26	PUT	/api/admin/users/{id}	Оновлення користувача	Адміністратор
27	DELETE	/api/admin/users/{id}	Видалення (архівування) користувача	Адміністратор
28	GET	/api/dashboard/summary	Зведені показники (залежать від ролі)	Автентифікований
29	GET	/api/health	Перевірка стану сервісу	Публічний

3.1.5 Ініціалізація бази даних

Схему бази даних і початкові дані ініціалізовано через версіоновані міграції Flyway, розміщені в каталозі db/migration. Міграція V1__create_library_schema.sql (лістинг 3.6) створює всі основні таблиці, зв'язки, обмеження унікальності та індекси. Міграція V2__seed_demo_data.sql наповнює базу демонстраційними даними: ролями, користувачами, авторами, категоріями, виданнями, примірниками, а також прикладами бронювань і видач. Міграція V3__create_refresh_tokens.sql додає таблицю refresh-токенів, а V4__link_reservations_to_book_copies.sql додає до бронювань зв'язок із конкретними примірниками. Послідовне застосування міграцій забезпечує відтворюваність бази даних: на чистому екземплярі СУБД система автоматично розгортає повну схему з демонстраційними даними, готовими до сценарію демонстрації.

Лістинг 3.6 — *Міграція створення схеми бази даних (V1__create_library_schema.sql)*

```
CREATE TABLE roles (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(40) NOT NULL,
    description VARCHAR(255),
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT uk_roles_name UNIQUE (name)
);

CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(160) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
```

```
    first_name VARCHAR(80) NOT NULL,
    last_name VARCHAR(80) NOT NULL,
    phone VARCHAR(40),
    status VARCHAR(30) NOT NULL,
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT uk_users_email UNIQUE (email)
);

CREATE TABLE user_roles (
    user_id BIGINT NOT NULL,
    role_id BIGINT NOT NULL,
    PRIMARY KEY (user_id, role_id),
    CONSTRAINT fk_user_roles_user FOREIGN KEY (user_id)
REFERENCES users (id),
    CONSTRAINT fk_user_roles_role FOREIGN KEY (role_id)
REFERENCES roles (id)
);

-- Таблиці authors, publishers, categories, а також
сполучні book_authors
-- і book_categories створюються аналогічно (наведено у
вихідному коді міграції V1).

CREATE TABLE books (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    publisher_id BIGINT,
    title VARCHAR(240) NOT NULL,
    isbn VARCHAR(20) NOT NULL,
    description TEXT,
    language VARCHAR(40) NOT NULL,
```

```
    publication_year INT,
    page_count INT,
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT uk_books_isbn UNIQUE (isbn),
    CONSTRAINT fk_books_publisher FOREIGN KEY
(publisher_id) REFERENCES publishers (id)
);

CREATE TABLE book_copies (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_id BIGINT NOT NULL,
    inventory_code VARCHAR(80) NOT NULL,
    status VARCHAR(30) NOT NULL,
    location VARCHAR(120),
    acquired_at DATE,
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT uk_book_copies_inventory_code UNIQUE
(inventory_code),
    CONSTRAINT fk_book_copies_book FOREIGN KEY (book_id)
REFERENCES books (id)
);

CREATE TABLE reservations (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_id BIGINT NOT NULL,
    reader_id BIGINT NOT NULL,
    status VARCHAR(30) NOT NULL,
```

```
    reserved_at TIMESTAMP(6) NOT NULL,
    expires_at TIMESTAMP(6) NOT NULL,
    fulfilled_at TIMESTAMP(6),
    cancelled_at TIMESTAMP(6),
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT fk_reservations_book FOREIGN KEY (book_id)
REFERENCES books (id),
    CONSTRAINT fk_reservations_reader FOREIGN KEY
(reader_id) REFERENCES users (id)
);

CREATE TABLE loans (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    book_copy_id BIGINT NOT NULL,
    reader_id BIGINT NOT NULL,
    librarian_id BIGINT,
    status VARCHAR(30) NOT NULL,
    loaned_at TIMESTAMP(6) NOT NULL,
    due_at TIMESTAMP(6) NOT NULL,
    returned_at TIMESTAMP(6),
    created_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    updated_at TIMESTAMP(6) NOT NULL DEFAULT
CURRENT_TIMESTAMP(6),
    CONSTRAINT fk_loans_book_copy FOREIGN KEY
(book_copy_id) REFERENCES book_copies (id),
    CONSTRAINT fk_loans_reader FOREIGN KEY (reader_id)
REFERENCES users (id),
    CONSTRAINT fk_loans_librarian FOREIGN KEY
(librarian_id) REFERENCES users (id));
```

Демонстраційний набір даних включає 4 ролі, кількох користувачів (адміністратор, бібліотекар, читачі), 10 авторів, 6 тематичних категорій, 30 видань із примірниками, а також приклади активних бронювань і видач. Усі демонстраційні облікові записи використовують єдиний тестовий пароль. Перелік демонстраційних облікових записів наведено в таблиці 3.2.

Таблиця 3.2 — *Демонстраційні облікові записи*

Роль	**Електронна пошта**	**Призначення**
Адміністратор	admin@library.local	Керування довідниками, виданнями, примірниками, користувачами
Бібліотекар	librarian@library.local	Бронювання, видача, повернення, прострочення
Читач	reader.olena@library.local	Пошук у каталозі, бронювання, власні видачі
Читач	reader.dmytro@library.local	Додаткові демонстраційні дані
Читач	reader.iryna@library.local	Додаткові демонстраційні дані

3.2 Реалізація клієнтської частини (Next.js)

3.2.1 Налаштування проекту та авторизації

Клієнтську частину створено на основі Next.js 16 із використанням React 19, мови TypeScript і моделі маршрутизації App Router. Керування пакетами здійснюється менеджером pnpm [21]. До складу залежностей включено TanStack Query для роботи з серверним станом, React Hook Form і Zod для роботи з формами та їх валідації, Tailwind CSS 4 для стилізації, бібліотеку lucide-react [20] для іконок та допоміжні утиліти (clsx, tailwind-merge). Структуру вихідного коду організовано за шарами app (маршрути й сторінки),

features (компоненти функціональних областей) і shared (спільні компоненти, API-клієнт, утиліти).

Центральним елементом взаємодії з сервером є централізований HTTP-клієнт (http-client.ts), реалізований як обгортка над стандартним інтерфейсом fetch. Клієнт автоматично встановлює заголовок Content-Type для запитів із тілом, додає заголовок Authorization: Bearer <токен> за наявності токена, серіалізує тіло запиту у JSON, обробляє коди стану відповіді та перетворює помилки на спеціалізований клас ApiError із кодом стану. Це дозволяє єдиним чином опрацьовувати запити в усьому застосунку. Текст HTTP-клієнта наведено нижче (лістинг 3.7).

Лістинг 3.7 — *Централізований HTTP-клієнт (http-client.ts)*

```
const API_URL = process.env.NEXT_PUBLIC_API_URL ??
"http://localhost:8080/api";

export class ApiError extends Error {
  constructor(
    message: string,
    public readonly status: number,
  ) {
    super(message);
    this.name = "ApiError";
  }
}

type RequestOptions = Omit<RequestInit, "body"> & {
  body?: unknown;
  token?: string;
};

export async function apiRequest<TResponse>(path: string,
options: RequestOptions = {}) {
```

```
  const headers = new Headers(options.headers);

  if (options.body !== undefined && !headers.has("Content-
Type")) {
    headers.set("Content-Type", "application/json");
  }

  if (options.token && !headers.has("Authorization")) {
    headers.set("Authorization", `Bearer
${options.token}`);
  }

  const response = await fetch(`${API_URL}${path}`, {
    ...options,
    headers,
    body: options.body === undefined ? undefined :
JSON.stringify(options.body),
  });

  if (!response.ok) {
    const errorBody = await response.json().catch(() =>
null) as { message?: string } | null;

    throw new ApiError(errorBody?.message ??
response.statusText, response.status);
  }

  if (response.status === 204) {
    return undefined as TResponse;
  }

  return (await response.json()) as TResponse;
}
```

Авторизацію на боці клієнта реалізовано на основі JWT. Після успішного входу клієнт зберігає видані токени та дані сесії й супроводжує ними подальші захищені запити. Окрему увагу приділено обробці завершення строку дії access-токена: реалізовано механізм автоматичного оновлення токена через refresh-токен із захистом від одночасних повторних спроб (single-flight), завдяки чому користувач не втрачає сесію та не потребує повторного входу під час роботи. У разі неможливості оновлення сесії застосунок коректно завершує її та повідомляє користувача. Для коректної роботи серверного й клієнтського рендерингу враховано гідрацію збереженого стану сесії, щоб уникнути розбіжностей між сервером і клієнтом.

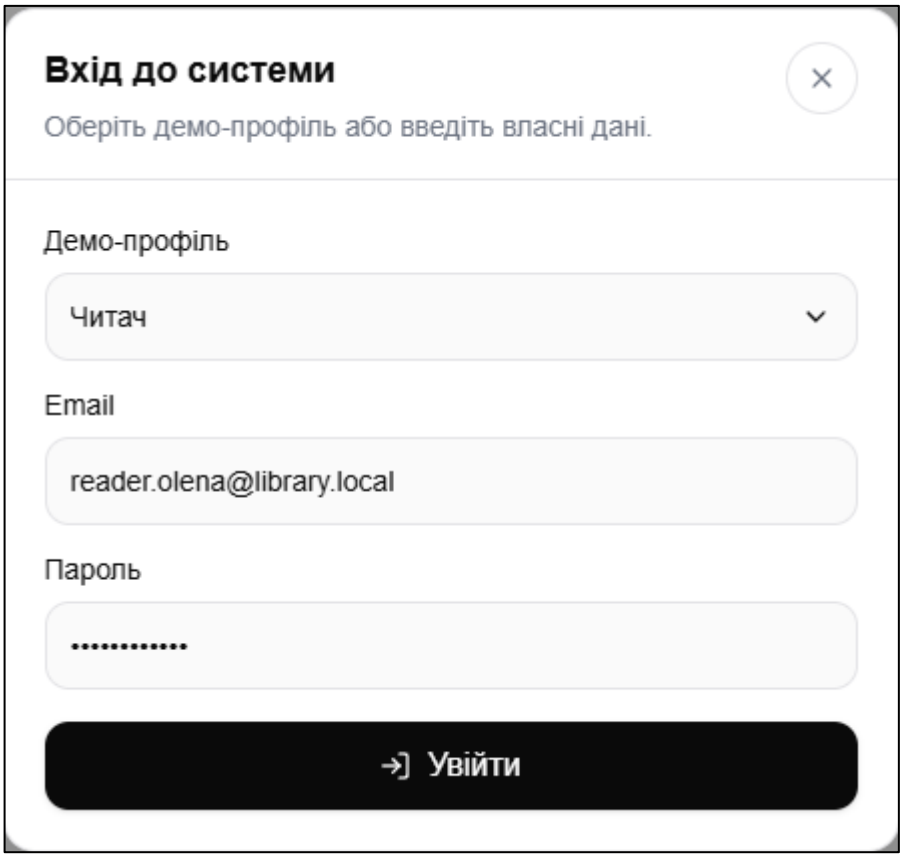


Рисунок 3.2 — *Форма авторизації користувача*

3.2.2 Каталог книг

Каталог є центральним і єдиним публічно доступним екраном застосунку. Він реалізує перегляд видань у вигляді таблиці з основними атрибутами (назва, автор, рік, категорія, ISBN, кількість доступних примірників і статус), пошук за пошуковим рядком, фільтрацію за категорією та статусом наявності, сортування й посторінкове виведення. Дані завантажуються з ендпоінта GET /api/catalog/books через TanStack Query, що забезпечує кешування результатів, повторні запити й коректне опрацювання станів завантаження, помилки та порожнього результату.

Каталог доступний усім категоріям користувачів, зокрема неавторизованим гостям. Для авторизованих читачів каталог доповнюється можливістю бронювання доступного видання безпосередньо з його рядка; кнопка бронювання змінює свій стан залежно від ролі користувача, доступності примірників і наявності запиту, що виконується. Дію бронювання супроводжує діалог підтвердження, а результат — спливне сповіщення. Загальний вигляд каталогу наведено на рисунку 3.3, а вигляд для авторизованого читача — на рисунку 3.4.

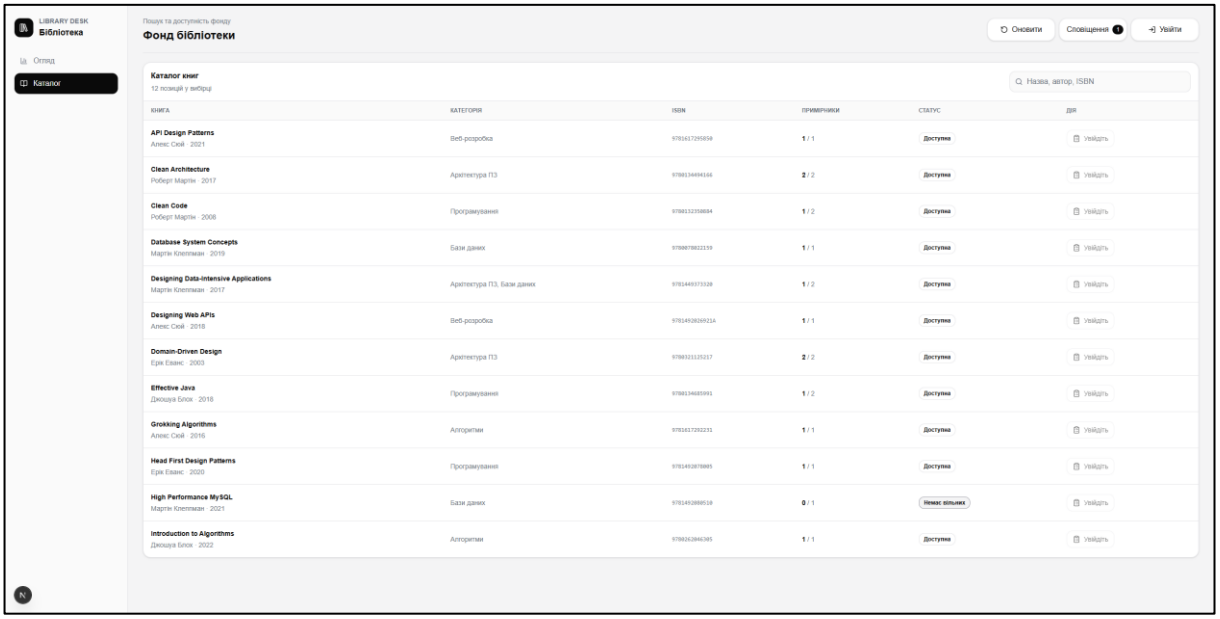


Рисунок 3.3 — *Каталог видань*

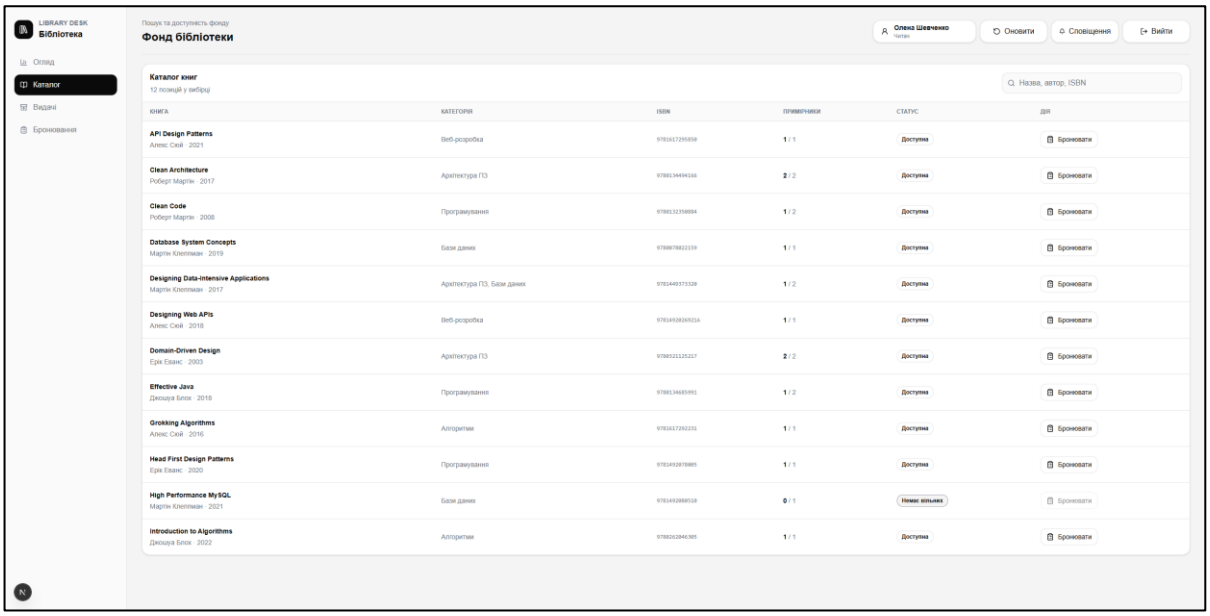


Рисунок 3.4 — *Каталог видань з функцією бронювання*

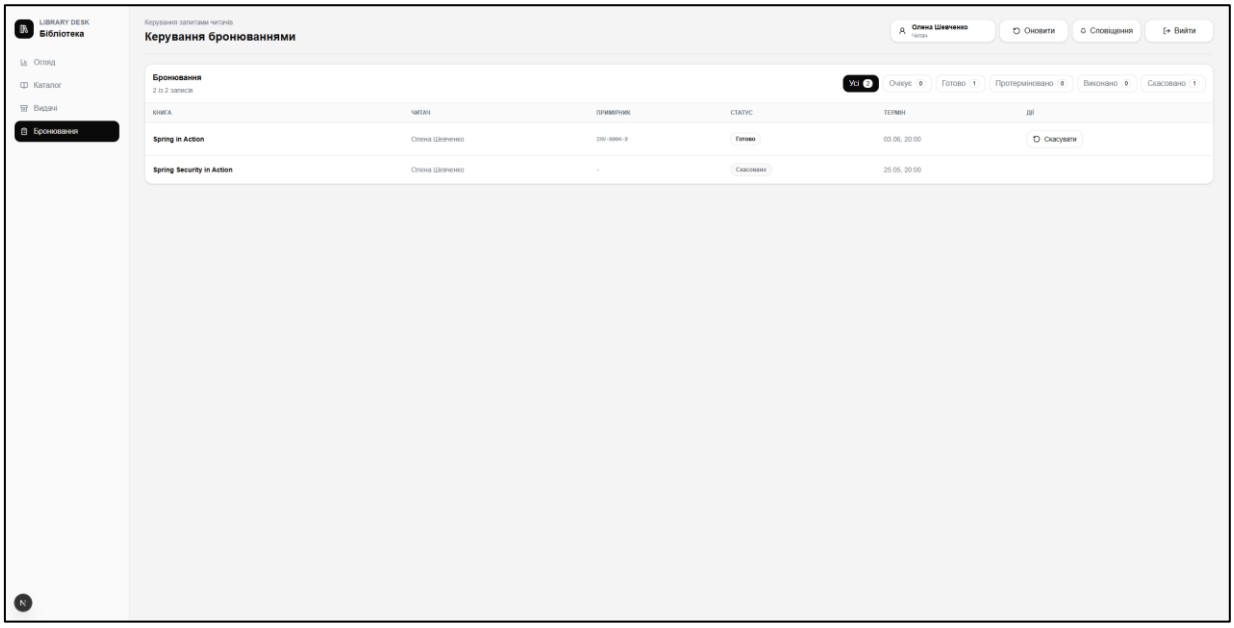


Рисунок 3.5 — *Робочий простір читача*

Робочий простір бібліотекаря реалізує перелік бронювань із можливістю підтвердження та скасування, видачу й повернення видань, а також перегляд прострочених видач. Списки доповнено фільтрами за статусом, пріоритетним

сортуванням і станами окремих рядків під час виконання операцій. Загальний вигляд робочого простору бібліотекаря наведено на рисунку 3.6.

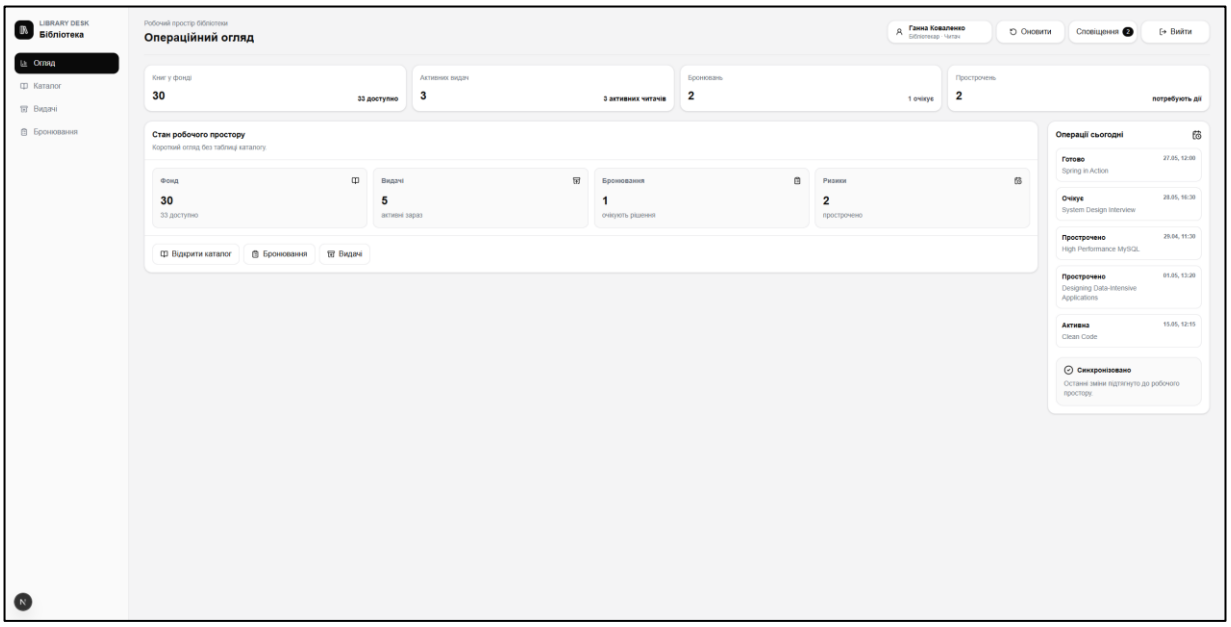


Рисунок 3.6 — *Робочий простір бібліотекаря*

3.2.3 Адміністративна панель

Адміністративна панель доступна користувачам із роллю адміністратора й реалізує повний набір операцій керування даними системи. Панель об'єднує керування виданнями, примірниками, користувачами, авторами, категоріями та видавництвами. Усі форми створення й редагування реалізовано в модальних вікнах, що відкриваються з панелі інструментів та з дій редагування в рядках таблиць; результат операцій супроводжується спливними сповіщеннями, а незворотні дії — діалогами підтвердження. Для великих таблиць (видання, користувачі) реалізовано клієнтську пагінацію, що запобігає надмірному розтягуванню сторінки.

Стандартні нативні елементи керування замінено власними повторно використовуваними компонентами (модальне вікно, спливні сповіщення, діалог підтвердження, випадний список вибору) в єдиному монохромному

стилі, що забезпечує узгодженість інтерфейсу. Загальний вигляд адміністративної панелі наведено на рисунках 3.7 і 3.8.

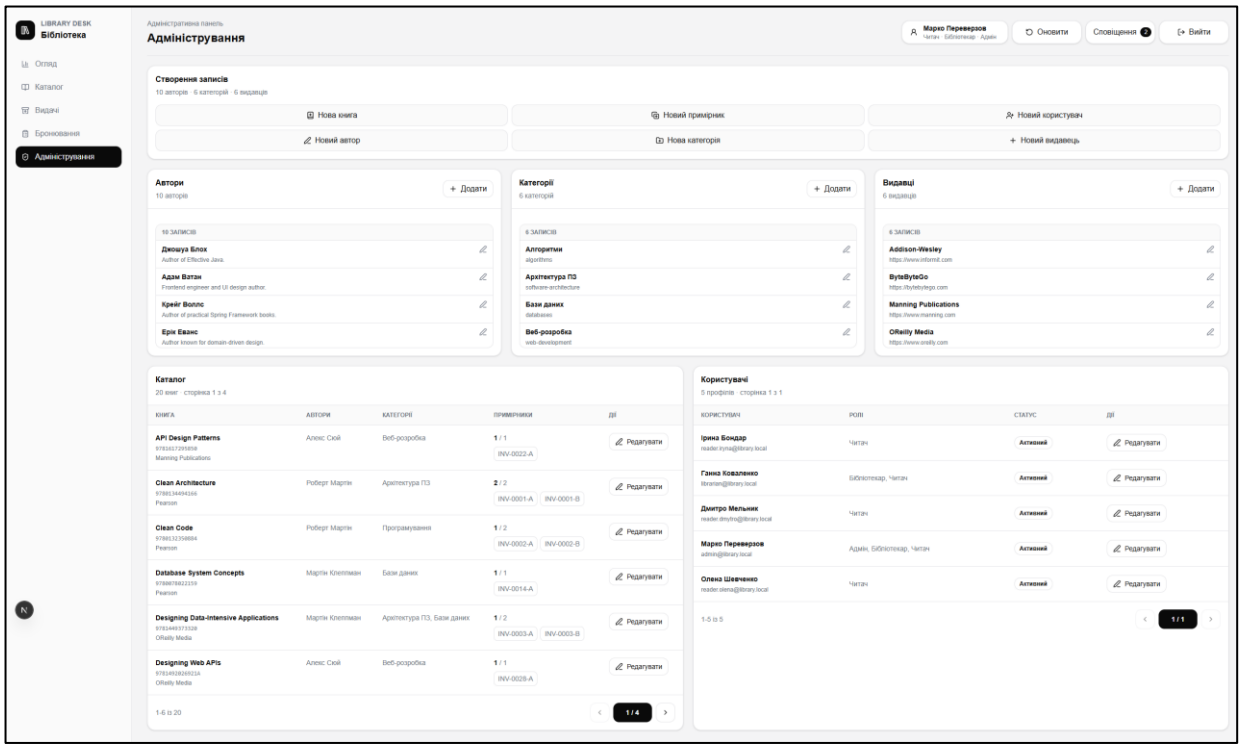


Рисунок 3.7 — Адміністративна панель

3.3 Розгортання та тестування

3.3.1 Локальне розгортання

Для локального розгортання системи передбачено зручний і відтворюваний процес. Базу даних MySQL розгортають у контейнері за допомогою Docker Compose [33] (docker-compose.yml), що усуває потребу в ручному встановленні та налаштуванні СУБД. Параметри підключення визначено у файлі .env.example й узгоджено зі значеннями за замовчуванням серверної частини. Серверну частину запускають через Maven-обгортку, після чого стають доступними REST API за адресою http://localhost:8080/api, інтерактивна документація Swagger UI та ендпоінт перевірки стану. Клієнтську частину встановлюють і запускають через pnpm, після чого

застосунок доступний за адресою http://localhost:3000 і за замовчуванням звертається до локального API. Зміст файлу docker-compose.yml наведено нижче (лістинг 3.8).

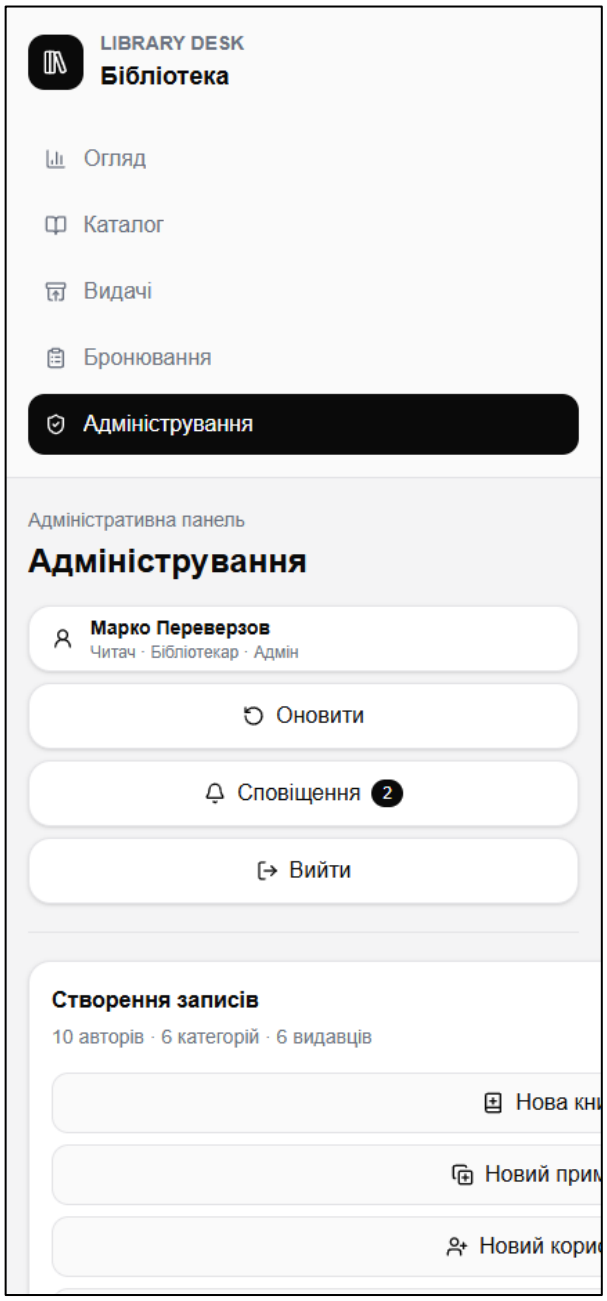


Рисунок 3.8 — *Адаптивне відображення адміністративної панелі*

Лістинг 3.8 — *Конфігурація Docker Compose (docker-compose.yml)*

```
services:
  mysql:
    image: mysql:8.0
```

```
    container_name: library-mysql
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE:-library_db}
      MYSQL_USER: ${MYSQL_USER:-library_user}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD:-library_password}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD:-
root_password}
    ports:
      - "3306:3306"
    volumes:
      - library_mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h",
"localhost", "-u", "root", "-p${MYSQL_ROOT_PASSWORD:-
root_password}"]
      interval: 10s
      timeout: 5s
      retries: 10
      start_period: 20s

volumes:
```

Для швидкої демонстрації без розгортання MySQL передбачено альтернативний режим запуску серверної частини на вбудованій СУБД H2 у режимі сумісності з MySQL. Послідовність кроків локального розгортання наведено в таблиці 3.3.

Таблиця 3.3 — *Кроки локального розгортання*

Крок	**Команда / дія**	**Результат**
1	docker compose up -d mysql	Піднято контейнер MySQL

2	mvn -f backend/pom.xml spring-boot:run	REST API на порту 8080, Swagger UI
3	pnpm --dir frontend install	Встановлено залежності клієнтської частини
4	pnpm --dir frontend dev	Застосунок доступний на порту 3000

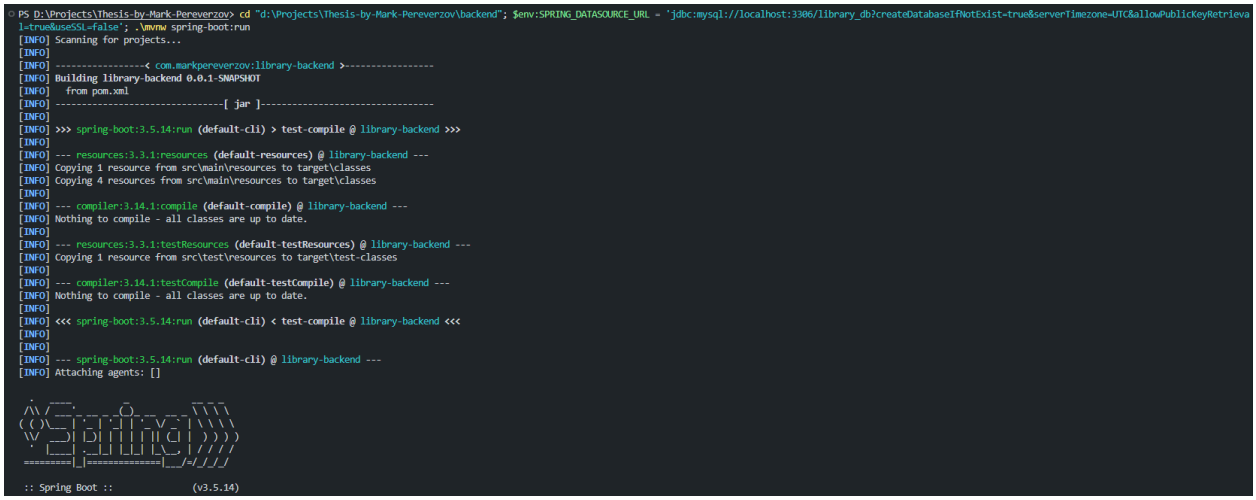


Рисунок 3.9 — *Успішний запуск серверної частини*

3.3.2 Тестування функціоналу

Коректність роботи системи перевірено за допомогою автоматизованих тестів та ручного наскрізного випробування. Критичну бізнес-логіку серверної частини покрито модульними (unit) та інтеграційними тестами на базі JUnit 5, Spring Boot Test і Spring Security Test із використанням вбудованої СУБД H2. Загалом реалізовано 21 тест, які успішно виконуються без помилок.

Тестами охоплено ключові сценарії: автентифікацію (вхід демонстраційного користувача, отримання даних поточного користувача, ротацію refresh-токена, відхилення невірного пароля), публічний каталог (перелік, пошук, деталі, довідники), бізнес-логіку бронювань (закріплення примірника, вивільнення під час скасування, конфлікт за відсутності доступних примірників), бізнес-логіку обігу (видача за готовим бронюванням,

відхилення видачі за непідтвердженим бронюванням, відхилення видачі недоступного примірника, ідемпотентність повернення), а також адміністрування (керування каталогом і користувачами, відмова в доступі читачеві). Перелік груп тестів наведено в таблиці 3.4.

Для ілюстрації підходу до тестування наведемо по одному конкретному прикладу модульного та інтеграційного тесту.

Приклад модульного тесту сервісного рівня.

Метод issueFromReadyReservationFulfillsReservationAndLoansReservedCopy класу CirculationServiceTests перевіряє ключовий бізнес-сценарій видачі примірника за готовим бронюванням. *Вхідні дані:* бронювання з ідентифікатором 1 зі статусом READY_FOR_PICKUP та закріпленим примірником; запит CreateLoanRequest з полем reservationId = 1, readerId = 3; контекст бібліотекаря з ідентифікатором 2. *Очікуваний результат:* повернута видача має статус ACTIVE, ідентифікатор примірника збігається з тим, що був зарезервований, статус примірника в базі змінився на LOANED, а бронювання переведено у статус FULFILLED з непорожнім полем fulfilledAt. *Тестовий дублікат:* вбудована СУБД H2 у режимі сумісності з MySQL (тип — **fake**; реальна реалізація СУБД замінена спрощеним in-memory аналогом); транзакція автоматично відкочується після кожного тесту анотацією @Transactional, що забезпечує ізольованість і повторюваність.

Приклад інтеграційного тесту контролера.

Метод loginMeAndRefreshTokenFlowWorksForSeedReader класу AuthControllerIntegrationTests перевіряє наскрізний сценарій автентифікації через HTTP-ендпоінти. *Вхідні дані:* POST-запит на /api/auth/login з тілом {"email": "reader.olena@library.local", "password": "Password123!"} (демонстраційний акаунт читача із сід-даних). *Очікуваний результат:* відповідь 200 OK, тіло містить поля tokenType = "Bearer", непорожні accessToken і refreshToken, у блоці user — email читача та роль ROLE_READER; подальший запит GET /api/auth/me із отриманим токеном

повертає 200 OK із коректними даними профілю. *Тестовий дублікат:* MockMvc — stub HTTP-рівня, що симулює надходження HTTP-запитів до Spring-контролерів без реального запуску вебсервера (тип — **stub**); база даних — H2 in-memory (fake), завантажена демонстраційними сід-даними через Flyway-міграції.

Таблиця 3.4 — *Групи автоматизованих тестів*

Тестовий клас / група	**Що перевіряє**
AuthControllerIntegrationTests	Вхід, поточний користувач, ротація токена, відхилення невірного пароля
CatalogControllerIntegrationTests	Перелік, пошук, деталі видання, автори, категорії
ReservationServiceTests	Закріплення / вивільнення примірника, конфлікт за відсутності копій
CirculationServiceTests	Видача з бронювання, відхилення непідтвердженого / недоступного, ідемпотентне повернення
CirculationWorkflowIntegrationTests	Наскрізний сценарій: бронювання -> видача -> повернення -> прострочення
DashboardControllerIntegrationTests	Показники для персоналу, читача, гостя
AdminControllerIntegrationTests	Керування каталогом / користувачами, відмова в доступі читачеві

Окрім автоматизованих тестів, виконано ручне наскрізне (end-to-end) випробування повного сценарію демонстрації на реальному стеку (Docker MySQL + Spring Boot + Next.js): перегляд каталогу гостем, вхід читача, бронювання видання, вхід бібліотекаря, підтвердження бронювання, видача, повернення, а також керування довідниками й користувачами

адміністратором. Під час випробування контролювалася відсутність помилок у консолі браузера та на сторінці, а також коректність адаптивного відображення на мобільному пристрої без горизонтального прокручування. Якість серверного коду додатково забезпечувалася контролем складання й проходженням усіх тестів, а клієнтського — успішним проходженням статичного аналізу (lint) та продакшн-збірки.

Результати тестування підтвердили, що реалізована система коректно виконує всі заявлені функції, забезпечує цілісність даних і коректне розмежування доступу, а отже, відповідає сформованим у підрозділі 1.4 вимогам.

ВИСНОВКИ

У ході кваліфікаційної роботи спроєктовано та реалізовано вебзастосунок для управління бібліотечним фондом і обслуговуванням читачів. Він охоплює повний бібліотечний цикл, розмежовує доступ за ролями й побудований на клієнт-серверній архітектурі, де Spring Boot і Next.js з'єднані через REST API. Поставлену мету досягнуто, усі сформульовані у вступі задачі виконано.

1. Проаналізовано предметну область бібліотечних інформаційних систем, визначено типові функції (каталогізація, облік примірників, бронювання, видача й повернення, контроль прострочень, адміністрування) та рольову модель користувачів (гість, читач, бібліотекар, адміністратор), що стало основою для формування вимог.
2. Виконано огляд і порівняльний аналіз наявних рішень (Koha, OpenBiblio, Evergreen, LibraryWorld). Аналіз показав, що наявні системи або надлишково складні й орієнтовані на великі організації, або застарілі та функціонально обмежені, або закриті комерційні, що обґрунтувало доцільність власної розробки на сучасному стеку.
3. Обґрунтовано вибір технологічного стеку: Java 25 та Spring Boot 4.0.6 для серверної частини, Next.js 16 і React 19 з TypeScript для клієнтської частини, MySQL 8.4 як реляційної СУБД, REST API для інтеграції та JWT для автентифікації з рольовим розмежуванням доступу.
4. Сформовано функціональні та нефункціональні вимоги до системи й на їх основі спроєктовано архітектуру (decoupled, package-by-feature, багатошаровий поділ), модель даних (концептуальну ER-модель і реляційну схему з керуванням через міграції Flyway), програмний інтерфейс REST API та інтерфейс користувача для всіх ролей.
5. Реалізовано серверну частину з підсистемами автентифікації (JWT із ротацією refresh-токенів, BCrypt, RBAC), публічного каталогу з пошуком, фільтрацією та пагінацією, бронювань, обігу видань із

транзакційною бізнес-логікою, адміністрування та зведених показників, а також документуванням API через OpenAPI/Swagger.

6. Реалізовано клієнтську частину на Next.js, що споживає REST API: централізований HTTP-клієнт із автоматичним оновленням токенів, каталог, робочі простори читача й бібліотекаря, адміністративну панель із модальними формами, спливними сповіщеннями та діалогами підтвердження в адаптивному монохромному інтерфейсі.
7. Виконано тестування (21 автоматизований тест критичної бізнес-логіки без помилок, а також наскрізне ручне випробування повного сценарію демонстрації) та забезпечено просте локальне розгортання засобами Docker Compose.

Практичне значення роботи полягає у створенні готового до використання та подальшого розвитку вебзастосунку, який реалізує повний цикл бібліотечного обслуговування й може бути впроваджений у навчальних та невеликих публічних бібліотеках. Реалізована архітектура з чітким розділенням частин і документованим API є відкритою для розширення.

Перспективними напрямами подальшого розвитку системи є: впровадження повнотекстового пошуку та рекомендаційних механізмів; додавання модулів звітності та аналітики; інтеграція зі сторонніми каталогами та бібліографічними стандартами (MARC); розроблення мобільного клієнта на основі наявного API; впровадження сповіщень читачів (електронною поштою) про готовність бронювань і наближення строку повернення; а також налаштування неперервної інтеграції та розгортання (CI/CD). Роботу оформлено відповідно до ДСТУ 8302:2015 [34] та ДСТУ 3008:2015 [35].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Koha — Open Source Library Management System [Electronic resource] / Koha Community. URL: https://koha-community.org/ (date of access: 31.05.2026).

2. OpenBiblio — Open Source Automated Library System [Electronic resource]. URL: https://sourceforge.net/projects/obiblio/ (date of access: 31.05.2026).

3. Evergreen ILS [Electronic resource] / Evergreen Community. URL: https://evergreen-ils.org/ (date of access: 31.05.2026).

4. LibraryWorld — Library Automation System [Electronic resource]. URL: https://www.libraryworld.com/ (date of access: 31.05.2026).

5. Spring Boot Reference Documentation [Electronic resource] / Broadcom (VMware Tanzu). URL: https://docs.spring.io/spring-boot/index.html (date of access: 31.05.2026).

6. Spring Framework Documentation [Electronic resource] / Broadcom (VMware Tanzu). URL: https://docs.spring.io/spring-framework/reference/ (date of access: 31.05.2026).

7. Spring Data JPA — Reference Documentation [Electronic resource] / Broadcom (VMware Tanzu). URL: https://docs.spring.io/spring-data/jpa/reference/ (date of access: 31.05.2026).

8. Hibernate ORM User Guide [Electronic resource] / Red Hat. URL: https://hibernate.org/orm/documentation/ (date of access: 31.05.2026).

9. Spring Security Reference [Electronic resource] / Broadcom (VMware Tanzu). URL: https://docs.spring.io/spring-security/reference/ (date of access: 31.05.2026).

10. Jakarta Persistence Specification [Electronic resource] / Eclipse Foundation. URL: https://jakarta.ee/specifications/persistence/ (date of access: 31.05.2026).

11. springdoc-openapi Documentation [Electronic resource]. URL: https://springdoc.org/ (date of access: 31.05.2026).

12. OpenAPI Specification [Electronic resource] / OpenAPI Initiative. URL: https://spec.openapis.org/oas/latest.html (date of access: 31.05.2026).

13. Flyway Documentation [Electronic resource] / Redgate. URL: https://documentation.red-gate.com/flyway (date of access: 31.05.2026).

14. Walls C. Spring in Action. 6th ed. Shelter Island : Manning Publications, 2022. 520 p.

15. Next.js Documentation [Electronic resource] / Vercel. URL: https://nextjs.org/docs (date of access: 31.05.2026).

16. React Documentation [Electronic resource] / Meta Open Source. URL: https://react.dev/ (date of access: 31.05.2026).

17. TypeScript Documentation [Electronic resource] / Microsoft. URL: https://www.typescriptlang.org/docs/ (date of access: 31.05.2026).

18. TanStack Query Documentation [Electronic resource]. URL: https://tanstack.com/query/latest (date of access: 31.05.2026).

19. Tailwind CSS Documentation [Electronic resource] / Tailwind Labs. URL: https://tailwindcss.com/docs (date of access: 31.05.2026).

20. Lucide React [Electronic resource] / Lucide Contributors. URL: https://lucide.dev/ (date of access: 31.05.2026).

21. pnpm [Electronic resource] / pnpm Contributors. URL: https://pnpm.io/ (date of access: 31.05.2026).

22. MySQL 8.4 Reference Manual [Electronic resource] / Oracle Corporation. URL: https://dev.mysql.com/doc/refman/8.4/en/ (date of access: 31.05.2026).

23. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

24. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Doctoral dissertation. Irvine : University of California, 2000. 162 p.

25. Richardson L., Ruby S. RESTful Web Services. Sebastopol : O'Reilly Media, 2007. 446 p.

26. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519 [Electronic resource] / Internet Engineering Task Force (IETF). 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (date of access: 31.05.2026).

27. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 560 p.

28. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

29. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Upper Saddle River : Prentice Hall, 2008. 464 p.

30. Project Lombok [Electronic resource]. URL: https://projectlombok.org/ (date of access: 31.05.2026).

31. MapStruct — Java bean mappings [Electronic resource]. URL: https://mapstruct.org/ (date of access: 31.05.2026).

32. Bloch J. Effective Java. 3rd ed. Boston : Addison-Wesley, 2018. 416 p.

33. Docker Documentation [Electronic resource] / Docker Inc. URL: https://docs.docker.com/ (date of access: 31.05.2026).

34. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.

35. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 26 с.

36. Переверзов М. Розробка веб-застосунку для автоматизації роботи бібліотеки з використанням Spring Boot та Next.js. Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, університетська конференція «МОЛОДА НАУКА-2026» Том III «Секції Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ. Запоріжжя, 2026. С. 114–115.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Переверзов Марк Андрійович, студент 4 курсу денної форми навчання Інженерного навчально-наукового інституту ім. Ю.М. Потебні Запорізького національного університету, спеціальність 121 «Інженерія програмного забезпечення» освітньо-професійної програми «Програмне забезпечення систем», — підтверджую, що написана мною кваліфікаційна робота бакалавра на тему «Розробка вебзастосунку для бібліотеки з інтеграцією Spring Boot та Next.js» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у статті 42 Закону України «Про освіту», зі змістом яких ознайомлений:

– засвідчую, що подана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– засвідчую, що дана кваліфікаційна робота виконана мною самостійно, не містить академічного плагіату, фабрикації, фальсифікації, а всі матеріали та результати інших авторів, використані в роботі, належним чином процитовані з посиланням на відповідні джерела;

– даю згоду на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 07.06.2026 _____ Переверзов Марк Андрійович
(студент)

Дата 07.06.2026 _____ Коломоєць Геннадій Павлович
(керівник)