МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ Економіки та бізнес-освіти

Кафедра Економіки та цифрового бізнесу

Спеціальність 122 «Комп'ютерні науки»

Форма навчання Денна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Білої Єлизавети Святославівни
(прізвище, ім'я, по батькові здобувача)

на тему Розробка SPA для керування книжною бібліотекою
(повна назва теми)

за матеріалами

(повна назва бази дослідження)

науковий керівник к.т.н. _____ Селезньов М.Є.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 12 червня 2026 р. № 15
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг – 2026

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384
Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри _____ В.М. Радько

"30" березня 2026 року

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

Білій Єлизаветі Святославівні

1. Тема роботи _Розробка SPA для керування книжною бібліотекою_

науковий керівник роботи _____Селезньов Максим Євгенович, к.т.н._____
затверджені наказом вищого навчального закладу від «23» березня 2026 р. № 193-ст

2. Строк подання здобувачем роботи 29.05.2026р.

3. Зміст кваліфікаційної бакалаврської роботи, об'єкт, предмет та мета дослідження:
Розділ 1 Теоретичні аспекти розробки spa для керування книжною бібліотекою
Розділ 2 Проєктування та розробка spa для керування книжною бібліотекою
Розділ 3 Функціонування та елементи інтерфейсу spa для керування книжною бібліотекою

Об'єкт дослідження – _ процес керування книжною бібліотекою засобами вебтехнологій

Предмет дослідження – _розробка SPA-вебдодатку для автоматизації перегляду книжкового каталогу, роботи з персональною бібліотекою та керування книжковим фондом

Мета кваліфікаційної бакалаврської роботи – розробити SPA-додаток для керування книжною бібліотекою, спрямований на підвищення зручності обліку книг, користувачів і персональних книжкових полиць у сучасному вебсередовищі

4. Дата видачі завдання 03.04.2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 17.04.2026р.	16.04.2026р.
2	Підготовка розділу 2	до 08.05.2026р.	07.05.2026р.
3	Підготовка розділу 3	до 25.05.2026р.	24.05.2026р.
4	Реєстрація завершеної кваліфікаційної роботи	до 29.05.2026р.	28.05.2026р.
5	Отримання відгуку від наукового керівника	04.06.2026р.	04.06.2026р.
6	Отримання зовнішньої рецензії	05.06.2026р.	05.06.2026р.
7	Попередній захист кваліфікаційної роботи на кафедрі	08.06.2026р.	08.06.2026р.
8	Перевірка кваліфікаційної роботи на плагіат	12.06.2026р.	12.06.2026р.
9	Допуск кафедрою кваліфікаційної роботи до захисту	12.06.2026р.	12.06.2026р.
10	Підготовка студента до захисту в ЕК	до 19.06.2026р.	18.06.2026р.

Завдання підготував науковий керівник _____ Селезньов М. Є.

Завдання одержав здобувач _____ Білая Є. С.

РЕФЕРАТ

Робота містить 72 сторінки, 22 малюнка, 43 джерела, 2 додатка.

Об’єкт дослідження: процес керування книжною бібліотекою засобами вебтехнологій.

Предмет дослідження: розробка SPA-вебдодатку для автоматизації перегляду книжкового каталогу, роботи з персональною бібліотекою та керування книжковим фондом.

Мета роботи: розробити SPA-додаток для керування книжною бібліотекою, спрямований на підвищення зручності обліку книг, користувачів і персональних книжкових полиць у сучасному вебсередовищі.

У результаті дослідження було розглянуто сучасні підходи до електронного керування бібліотекою, особливості SPA-архітектури, принципи побудови клієнт-серверних вебдодатків та засоби організації взаємодії між користувацьким інтерфейсом, серверною частиною і базою даних. Було визначено функціональні вимоги до системи, спроєктовано інформаційну модель і структуру бази даних. Розроблений SPA-додаток забезпечує зручну взаємодію користувача з електронним каталогом і дозволяє зменшити кількість ручних дій під час роботи з бібліотечними даними.

Область застосування: результати роботи можуть бути використані в навчальних проєктах, невеликих бібліотеках, електронних каталогах, інформаційних системах обліку книжкового фонду, а також як основа для подальшого розширення вебсервісів бібліотечного керування.

АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, БІБЛІОТЕКА, ВЕБІНТЕРФЕЙС, ЕЛЕКТРОННИЙ КАТАЛОГ, КНИЖКОВИЙ ФОНД, ОДНОСТОРІНКОВИЙ ВЕБДОДАТОК, SPA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ……...…….…….... 6
ВСТУП ……...…….... ... 7
РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ …...……..... ... 10
1.1 Огляд сучасних підходів до електронного керування книжною бібліотекою …….... 10
1.2 Аналіз існуючих SPA-систем і технологій їх розробки 19
1.3 Вибір технологічного стеку та архітектури веб-додатку …...……..... 23
Висновки до розділу 1 ……..... 27
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ ……...……...……..……... 29
2.1 Проєктування інформаційної моделі та бази даних вебдодатку ……... 29
2.2 Опис основних алгоритмів та методів збору та обробки даних ….. ... 36
2.3 Реалізація серверної та клієнтської частини вебдодатку ……...….. 41
Висновки до розділу 2 ...…….…..... 47
РОЗДІЛ 3 ФУНКЦІОНУВАННЯ ТА ЕЛЕМЕНТИ ІНТЕРФЕЙСУ SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ ……..... 49
3.1 Огляд реалізованого функціоналу розробленого SPA ……..... 49
3.2 Сценарії взаємодії користувача з розробленим вебдодатком …….. 57
3.3 Аналіз ефективності розробленої системи та можливих обмежень 61
Висновки до розділу 3 ……...……..... 64
ВИСНОВКИ ……...…….…..... 66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ …….....…. 68
ДОДАТКИ ….……..... 73

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SPA – Single page application (Односторінковий додаток)

UI – User Interface (інтерфейс користувача)

API – Application Programming Interface (інтерфейс програмування застосунків)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

JSON – JavaScript Object Notation (формат обміну даними)

SQL – Structured Query Language (мова структурованих запитів)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

JS – JavaScript (мова програмування JavaScript)

REST – Representational State Transfer (архітектурний стиль взаємодії вебсервісів)

ER-діаграма – Entity-Relationship Diagram (діаграма «сутність–зв’язок»)

БД – База даних

ВСТУП

У сучасних умовах розвитку інформаційних технологій все більшого значення набуває автоматизація різних сфер діяльності людини. Однією з таких сфер є бібліотечна справа, яка, попри активний розвиток цифрових сервісів, усе ще часто використовує застарілі або лише частково автоматизовані підходи до ведення обліку книжкок, читачів та операцій видачі літератури. У зв'язку з цим виникає потреба у створенні нових, сучасних, зручних і функціональних інформаційних систем, які будуть дозволяти спрощення керування бібліотечними процесами та підвищення ефективності роботи.

Одним із найбільш актуальних напрямів сучасної веб-розробки є створення SPA – односторінкових веб-застосунків. Такі застосунки забезпечують швидку взаємодію користувача з інтерфейсом, динамічне оновлення даних, яке не потребує повного перезавантаження сторінки, а також більш комфортний досвід для користувача. Саме тому використання SPA підходу є доцільним для створення системи керування книжною бібліотекою.

Сьогодні автоматизація бібліотечних процесів є значущою складовою цифрової трансформації для освітніх і культурних установ. Сучасні бібліотеки потребують не лише електронного обліку книг, а й можливості швидкого пошуку, зручного перегляду інформації, контролю наявності літератури, обліку читачів та відстеження операцій по видачі й поверненню книг. Використання веб-технологій у поєднанні з сучасними підходами до розробки інтерфейсів дає змогу створити зручний інструмент для вирішення схожих завдань.

Актуальність теми полягає в необхідності розробки сучасного веб-застосунку для автоматизації процесів керування книжною бібліотекою. Використання SPA, як різновиду веб-сторінки, дає змогу підвищити швидкість роботи системи, спростити доступ до інформації та покращити взаємодію користувача із застосунком, зробивши процес управління бібліотечними ресурсами більш зручним і ефективним.

На сьогоднішній день питання розвитку бібліотеки у цифровому світі розглядається як у практичному, так і в науковому аспектах. Багато сучасних бібліотечних систем орієнтовані саме на електронний облік, створення цифрових каталогів, управління користувачами та поєднання з онлайн-сервісами. Світова практика показує, що розвиток таких систем спрямований на підвищення доступності інформації, оптимізацію внутрішніх процесів та покращення якості обслуговування користувачів. Одним з вдалих прикладів може послугувати DDB (Deutsche Digitale Bibliothek) – найбільша онлайн-бібліотека у Німеччині. Робота над проєктом розпочалася ще у 2007 році за участю бібліотек, музеїв та архівів, а офіційний запуск платформи відбувся у 2012 році. У середині зібрано більш ніж 5,6 мільйону книг, архивів, аудиозаписів та репродукцій картин. Даний веб-ресурс хоча і не є повністю SPA-застосунком, але завдяки SPA-елементам сайт був забезпечений зручністю користування, швидким оновленням інформації та більш динамічною взаємодією з контентом. Це свідчить про те, що навіть великі цифрові бібліотеки орієнтуються на сучасні веб-технології для підвищення ефективності роботи та покращення користувацького досвіду.

Метою дипломної роботи є розробка односторінкового веб-застосунку для керування бібліотекою, який забезпечує зручне ведення обліку книг, читачів та операцій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– дослідити сучасні підходи до розробки SPA та особливості їх використання у веб-застосунках;

– проаналізувати особливості автоматизації бібліотечної діяльності;

– визначити функціональні вимоги до системи керування книжною бібліотекою;

– спроєктувати структуру застосунку та базу даних для зберігання інформації;

– розробити інтерфейс користувача для взаємодії з бібліотечною системою;

– реалізувати основний функціонал для керування книгами, читачами та операціями видачі й повернення літератури;

–провести тестування створеного програмного продукту та оцінити його роботу.

Об’єктом дослідження є процес автоматизації бібліотечної діяльності з використанням веб-застосунків. Предметом дослідження є методи, технології та програмні засоби розробки SPA для керування книжною бібліотекою.

У процесі виконання дипломної роботи були використані такі методи дослідження: аналіз наукової та навчальної літератури, аналіз технічної документації, системний підхід до проєктування інформаційних систем, методи моделювання структури даних і логіки роботи застосунку, а також практична реалізація програмного продукту з використанням сучасних веб-технологій.

Інформаційну базу дослідження становлять наукові праці, навчальні посібники, матеріали відкритих джерел, а також сучасні приклади реалізації односторінкових веб-застосунків.

Практичне значення отриманих результатів полягає у створенні SPA для керування бібліотекою книг, який може бути використаний як основа для автоматизації процесів у навчальних закладах, невеликих бібліотеках або інших установах, що працюють із книжковими фондами. Розроблений застосунок дозволяє спростити облік літератури, покращити систематизацію даних та підвищити зручність роботи користувачів із системою.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ

1.1 Огляд сучасних підходів до електронного керування книжною бібліотекою

У сучасному світі інформаційні технології дедалі глибше інтегруються в усі сфери людської діяльності. Освіта, наука, бізнес, державне управління, медицина, торгівля – усі ці галузі поступово переходять до цифрового формату. Не стала винятком і бібліотечна справа. Сучасні бібліотеки зазнають значних трансформацій, адаптуючись до швидких змін інформаційного середовища та потреб суспільства. Вони перетворюються з традиційних сховищ друкованих видань на багатофункціональні інформаційні центри, що поєднують фізичні та цифрові ресурси. Така еволюція зумовлює не лише зміну структури та функцій бібліотек, а й оновлення їхньої концепції – сьогодні все частіше використовуються терміни «медіатека», «науково-інформаційний центр», «цифрова бібліотека» тощо [1].

Традиційна модель ведення бібліотечної діяльності, яка базується на паперових журналах, картках читачів та ручному оформленні видачі книг, поступово втрачає ефективність у сучасних умовах. Збільшення обсягів інформації, необхідність швидкого пошуку літератури, потреба в контролі повернення книг, обліку користувачів та доступі до актуальної інформації в режимі реального часу роблять цифрові інструменти не просто зручними, а фактично необхідними. Саме тому електронне керування книжною бібліотекою сьогодні є важливим напрямом розвитку бібліотечних інформаційних систем.

Подальший розвиток бібліотечних технологій тісно пов’язаний із глобальними тенденціями цифровізації. Нові напрями еволюції інформаційних технологій у бібліотечній справі спрямовані на активне використання цифрових навчальних ресурсів, розвиток онлайн-комунікацій, упровадження принципів

ефективної взаємодії у віртуальному середовищі та на різноманітних платформах дистанційного доступу. Важливим аспектом є також формування цифрової компетентності фахівців, яка включає знання основ кібербезпеки, захисту персональних даних та навички безпечного користування інформаційними системами [2].

Усе це ще раз підтверджує, що сучасна бібліотека вже не може ефективно функціонувати без використання цифрових рішень. Під електронним керуванням книжною бібліотекою слід розуміти сукупність цифрових інструментів, методів і програмних рішень, які забезпечують автоматизацію основних бібліотечних процесів. До таких процесів, як правило, належать облік книжкового фонду, ведення інформації про читачів, оформлення видачі та повернення книг, пошук і фільтрація літератури, аналіз наявності та доступності примірників, а також адміністрування бібліотечних даних.

Сучасний підхід до бібліотечного керування передбачає, що бібліотека є не лише місцем фізичного зберігання книжок, а й інформаційним середовищем, у якому дані повинні бути впорядкованими, доступними, структурованими та зручними для обробки. Саме тому цифровізація бібліотечної діяльності є не просто технічним оновленням, а зміною самої логіки організації роботи.

Однією з основних причин переходу до електронних систем є необхідність підвищення ефективності роботи бібліотекаря. У традиційній моделі значна частина часу витрачається на пошук карток, перевірку наявності книг, ручне внесення записів та контроль термінів повернення. У цифровій системі більшість цих дій може бути виконана автоматично або за значно коротший час. Наприклад, пошук книги за автором, жанром або назвою займає лише кілька секунд, а інформація про читача, його історію видач та у якому статусі книжки, відображається одразу після запиту.

Ще одним важливим аспектом є зручність для користувачів бібліотеки. Сучасний користувач звик до швидкого доступу до інформації, зрозумілих інтерфейсів та можливості взаємодіяти з цифровими сервісами без зайвих труднощів. Саме тому бібліотечні системи повинні відповідати сучасним

очікуванням: мати логічну структуру, просту навігацію, можливість пошуку та фільтрації, а також надавати доступ до потрібної інформації без перевантаження інтерфейсу.

У зв’язку з цими тенденціями дедалі більшої популярності набувають програмні рішення, здатні забезпечити комплексну автоматизацію бібліотечних процесів та підтримку цифрової взаємодії користувачів із ресурсами. Розвиток таких систем став відповіддю на потребу бібліотек у створенні зручних, швидкодіючих і масштабованих платформ, які поєднують традиційні функції обліку фондів із можливостями електронного доступу. Саме тому на сучасному етапі значна увага приділяється розробці веб-орієнтованих систем управління бібліотеками, що використовують сучасні технології, фреймворки та архітектурні підходи.

Одним із найвідоміших відкритих рішень є автоматизована інформаційно-бібліотечна система Koha – повнофункціональна інтегрована бібліотечна система (Integrated Library System, ILS) [36], що підтримує каталогізацію, облік фондів, видачу й повернення книжок, роботу з читачами та формування звітів. Koha має веб-інтерфейс і API, що дозволяє інтегрувати її з іншими сервісами.

АІБС«Koha» забезпечує основні функції, які потрібні для обслуговування читачів:

- онлайн-доступ до електронного каталогу;
- електронне замовлення документів;
- каталогізація;
- база даних читачів бібліотеки;
- видача/повернення книжок;
- зв’язок між відділами та філіями бібліотеки.
- Крім перерахованих вище особливостей, АІБС «Koha» має і ряд інших:
- простий і зрозумілий інтерфейс;
- можливість широкого, багатоаспектного і оперативного пошуку;

– можливість керування документальними потоками і абонентами;

– можливість імпорту та експорту бібліографічних записів;

– можливість аналітичного опису статей;

– здатність працювати з будь-якою кількістю підрозділів, користувачів (категорій користувачів), примірників (категорій примірників), валют та інших бібліотечних даних [3].

Головною перевагою цієї системи є відкритий вихідний код, що забезпечує гнучкість налаштувань і відсутність ліцензійних витрат. Не менш важливою є активна підтримка з боку міжнародної спільноти, завдяки якій система постійно розвивається, оновлюється та адаптується до потреб різних бібліотек. Це робить Koha особливо привабливим рішенням для установ, які потребують функціональної системи без значних витрат на програмне забезпечення.

Ще одним поширеним рішенням є DSpace – платформа для створення цифрових репозиторіїв і збереження електронних ресурсів, зокрема дисертацій, статей, мультимедійних файлів [37]. Ця система була створена у співпраці фахівців корпорації Hewlett Packard і науковців Массачусетського технологічного інституту (США). Програмне забезпечення з відкритим вихідним кодом використовують з листопада 2002 року. Нині система DSpace є лідером серед схожих систем і використовується у понад 1000 організацій і установ у всьому світі. Програмне забезпечення призначене для довготривалого зберігання цифрових матеріалів [4].

Особливості системи DSpace:

– перегляд і пошук документів у системі, можуть виконуватися анонімно, але щоб виконати додавання документів, користувачеві потрібно зареєструватися;

– автентифікація здійснюється у традиційний спосіб, з використанням логіна та пароля користувача, які містяться у базі самої системи або у каталозі LDAP; також, автентифікація можлива з використанням сертифікатів X509;

– авторизація реалізується за допомогою системи прав DSpace, що дозволяє розмежовувати користувачам доступ до елементів в архіві, як на колекцію, так і на рівні окремих елементів;

– для кожної колекції можна вказати групу користувачів, яка виконуватиме редагування метаданих надісланих матеріалів;

– матеріали в архіві отримують унікальний і постійний URL (для створення цих ідентифікаторів використано механізм CNRI Handle System), наведений в описі кожного документа, який можна зазначати у бібліографічних посиланнях на даний документ;

– можливість створювати домашні сторінки зібрання із зображеннями описовим текстом;

– автоматичне розсилання повідомлень про останні надходження в репозитарій через службу підписки;

– можливість зберігати різноформатні дані, від текстових документів, зображень до наборів даних, відеоматеріалів та HTML-документів;

– потужна пошукова система (за зовнішніми посиланнями, автором, назвою, датою повнотекстовий пошук);

– персоналізація (наявність персональної сторінки користувача, на якій можна проглянути свої завантажені документи їх статус, підписки);

– процес внесення матеріалу може бути перерваний у будь-який момент часу без втрати введених даних;

– для кожної колекції матеріалів може бути налаштований робочий процес;

– модуль статистики дозволяє відстежувати кількість переглядів матеріалу і завантажень файлів матеріалу;

– пакет інструментів для імпорту й експорту колекції і матеріалів між DSpace та іншими системами [5].

Хоча DSpace не є класичною бібліотечною системою, вона широко використовується у вищих навчальних закладах і наукових установах для

організації відкритого доступу до цифрових матеріалів. DSpace підтримує стандарти метаданих і може бути інтегрована з іншими інформаційними системами.

Серед комерційних продуктів особливої уваги заслуговує Ex Libris Alma – сучасна хмарна платформа для управління бібліотечними ресурсами, орієнтована переважно на великі академічні та наукові бібліотеки [38]. Ця система забезпечує можливість комплексного управління як фізичними, так і електронними фондами, підтримує інтеграцію з discovery-сервісами (зокрема, Primo), автоматизує основні бібліотечні процеси та пропонує розширені інструменти аналітики.

Завдяки хмарній архітектурі Alma дозволяє бібліотекам об’єднувати свої каталоги в єдиному інформаційному просторі, що значно спрощує доступ до фондів і забезпечує ефективний обмін бібліографічними даними між установами. Такий підхід підвищує рівень співпраці між бібліотеками та сприяє створенню глобальних мереж знань [6].

Отже, аналіз існуючих рішень показує, що сучасні системи бібліотечного керування можуть мати різний функціональний напрям і різний рівень складності. Одні з них орієнтовані на повний цикл бібліотечного обслуговування, інші – на збереження й поширення цифрових матеріалів, ще інші – на комплексне хмарне керування ресурсами великих установ. Проте всі вони демонструють спільну тенденцію: бібліотечна сфера дедалі активніше переходить до використання цифрових платформ, що забезпечують швидкість, зручність та гнучкість роботи з інформацією.

У межах електронного керування бібліотекою можна виділити кілька основних функціональних напрямів. Першим із них є керування книжковим фондом. Це базова складова будь-якої бібліотечної системи, яка включає додавання нових книг, редагування наявної інформації, видалення записів, перегляд характеристик літератури, контроль кількості примірників та відображення статусу доступності. У сучасній системі книга вже не є просто записом у списку – це структурований об’єкт із набором характеристик, серед

яких назва, автор, рік видання, жанр, видавництво, ISBN, опис, статус, місце зберігання тощо.

Другим важливим напрямом є керування користувачами або читачами бібліотеки. Бібліотечна система повинна забезпечувати можливість створення, перегляду та редагування інформації про читачів. Це може бути ім’я, контактні дані, номер читацького квитка, дата реєстрації, історія користування бібліотекою. У більш сучасних реалізаціях також передбачаються ролі користувачів, наприклад адміністратор, бібліотекар, читач.

Третій напрям – облік операцій видачі та повернення книг. Саме цей функціонал забезпечує практичну взаємодію між книжковим фондом і читачами. У системі повинна бути можливість оформити видачу книги конкретному користувачу, зафіксувати дату видачі, очікувану дату повернення та фактичне повернення. Також важливим є контроль статусу книги: якщо примірник виданий, він не повинен відображатися як доступний для повторної видачі. Це дозволяє уникати помилок та підтримувати актуальність даних.

Окрему роль відіграє пошук і фільтрація інформації. Якщо бібліотечна система містить десятки, сотні або тисячі записів, користувач повинен мати можливість швидко знайти потрібну книгу або читача. Саме тому сучасні системи зазвичай реалізують пошук за ключовими словами, фільтрацію за категоріями, сортування за певними ознаками та інші інструменти навігації в даних.

Якщо говорити про сучасні підходи до електронного керування бібліотекою, то варто зазначити, що сьогодні існує кілька моделей організації таких систем.

Перший підхід – це локальні бібліотечні програми, які встановлюються на комп’ютери бібліотеки та використовуються в межах одного закладу. Їхньою перевагою є відносна простота впровадження та відсутність обов’язкової залежності від інтернет-з’єднання. Однак такі системи мають і суттєві недоліки: обмежений доступ, складність оновлення, проблеми з масштабуванням та менш гнучкий інтерфейс.

Другий підхід – веб-орієнтовані системи керування бібліотекою. Саме цей варіант сьогодні вважається найбільш перспективним, оскільки він забезпечує доступ до системи через браузер, спрощує адміністрування, оновлення та масштабування. Веб-застосунок не потребує складного встановлення на кожному пристрої та може використовуватися як на комп'ютерах бібліотекаря, так і потенційно на пристроях користувачів.

Третій підхід – хмарні бібліотечні системи, які зберігають дані на віддалених серверах і можуть надавати розширений функціонал, інтеграції та доступ із різних пристроїв. Хмарні рішення є особливо актуальними для великих бібліотечних мереж, однак для дипломного проєкту або системи малого масштабу зазвичай достатньо класичного веб-застосунку з серверною частиною та базою даних.

Важливо також розуміти, що сучасна бібліотечна система це не просто таблиця з книгами, а повноцінна інформаційна система, яка повинна відповідати певним вимогам. До основних вимог можна віднести:

- зручність використання;
- швидкість роботи;
- логічну структуру інтерфейсу;
- надійне зберігання даних;
- можливість редагування та оновлення інформації;
- масштабованість;
- безпеку доступу;
- простоту супроводу.

Зручність використання є особливо важливою, оскільки бібліотечні системи часто використовуються не технічними спеціалістами, а працівниками, для яких головне: простота, зрозумілість і передбачуваність роботи інтерфейсу. Якщо система є складною, перевантаженою або нелогічною, це знижує її практичну цінність навіть за наявності великої кількості функцій.

Питання впорядкування бібліотечних ресурсів, їх опису та зручного пошуку не є новим. Навіть у бібліотеках давнини простежуються уявлення про організацію, зберігання та опис книжкових зібрань. Ще у 1627 р. французький книгознавець і бібліотекар Габріель Ноде наголошував, що неорганізоване зібрання книг не можна повною мірою вважати бібліотекою. У своїй праці «Поради для облаштування бібліотек» він звертав увагу на необхідність упорядкування фонду, створення каталогів і забезпечення таких умов, за яких кожен міг би знайти в бібліотеці те, що шукає [7].

Тому окремої уваги заслуговує питання електронних каталогів. У багатьох сучасних бібліотеках саме електронний каталог стає основним інструментом доступу до інформації про літературу. На відміну від традиційних картотек, електронний каталог дозволяє здійснювати багатокритеріальний пошук, переглядати пов’язані записи, швидко оновлювати інформацію та забезпечувати централізований облік ресурсів. У межах SPA для керування книжною бібліотекою електронний каталог виступає ядром усієї системи.

Не менш важливим є й контроль актуальності даних. У бібліотечному середовищі інформація постійно змінюється: додаються нові книги, змінюється їхній статус, з’являються нові читачі, оформлюються видачі та повернення [39]. Якщо система не забезпечує актуальність даних у режимі реального часу або хоча б у максимально наближеному до нього форматі, вона швидко втрачає практичну цінність.

Саме тут сучасні веб-підходи, а особливо SPA, демонструють свою перевагу. Вони дозволяють швидко оновлювати окремі частини інтерфейсу без повного перезавантаження сторінки, що робить роботу із системою більш плавною, швидкою та комфортною. Це особливо важливо для систем, у яких користувач часто виконує послідовні дії: переглядає список книг, відкриває деталі, редагує запис, повертається назад, здійснює пошук тощо.

1.2 Аналіз існуючих SPA-систем і технологій їх розробки

Single page application (SPA) – це веб-додаток, який виконується безпосередньо на стороні клієнта у веб-браузері, зазвичай для написання такого типу веб-додатків використовується комбінація HTML, JavaScript та CSS. Додаток може отримувати доступ до структури веб-сторінки як до об’єктів DOM [8]. Завдяки цьому SPA забезпечує швидку зміну вмісту без необхідності повторного завантаження всієї сторінки, що значно підвищує продуктивність та комфорт користувача. Саме такі особливості зумовлюють зростання інтересу до цього підходу у сучасній веб-розробці, зокрема під час створення інформаційних систем, орієнтованих на швидку взаємодію з користувачем.

У сучасних умовах, коли швидкість обробки інформації та зручність користування є ключовими факторами, SPA набувають все більшого поширення. Вони дозволяють створювати веб-застосунки з ефектом безперервної роботи, забезпечуючи плавну взаємодію користувача з інтерфейсом та швидке оновлення контенту без зайвих затримок. Такий підхід особливо актуальний для систем, у яких користувач постійно взаємодіє з даними, зокрема для бібліотечних інформаційних систем [9].

Перед розробкою будь-якого веб-застосунку важливо проаналізувати вже існуючі рішення, визначити їх сильні та слабкі сторони, зрозуміти, які функціональні можливості є найбільш затребуваними, а також які технології доцільно використовувати для створення власного продукту. Це особливо актуально для систем керування книжною бібліотекою, оскільки така сфера вже має чимало прикладів цифрових рішень різного масштабу – від великих інформаційних платформ до невеликих локальних систем.

Якщо ж говорити саме про системи, ближчі до формату керування бібліотекою, а не лише перегляду цифрових колекцій, то можна виділити бібліотечні менеджмент-системи, які орієнтовані на такі основні задачі:

– облік книжкового фонду;

– реєстрацію читачів;

– видачу та повернення літератури;

– контроль статусу книг;

– управління користувачами;

– пошук і сортування записів.

У більшості сучасних реалізацій цього класу застосунків можна виділити спільні риси:

– наявність панелі керування;

– структуровані списки книг і користувачів;

– форми додавання та редагування даних;

– використання фільтрів, пошуку та сортування;

– поділ інтерфейсу на логічні модулі;

– застосування API для взаємодії між клієнтською та серверною частинами.

Усі ці характеристики безпосередньо пов’язані з принципами побудови SPA-застосунків, зокрема з компонентною структурою інтерфейсу, клієнтською маршрутизацією та асинхронною взаємодією з сервером.

З точки зору архітектури, основна відмінність SPA від традиційних багатосторінкових застосунків (MPA) полягає у способі обробки запитів і відображення контенту [40]. У класичних веб-системах кожна дія користувача супроводжується повним перезавантаженням сторінки, тоді як у SPA змінюється лише необхідна частина інтерфейсу. Це досягається за рахунок використання JavaScript та асинхронних запитів до сервера, які зазвичай повертають дані у форматі JSON.

Такий підхід забезпечує низку важливих переваг:

– швидший відгук інтерфейсу;

– плавну взаємодію з користувачем;

– відсутність візуальних затримок під час переходів;

– більш зручний користувацький досвід;

– можливість створення складних інтерфейсів, подібних до десктопних програм.

Для бібліотечної системи це має особливе значення. Наприклад, користувач може переглядати список книг, виконувати пошук, редагувати записи або оформлювати видачу без відчуття переходу між сторінками. Такий формат роботи значно спрощує взаємодію з системою та підвищує ефективність роботи.

Розглянемо більш детально, чому саме SPA є доцільним підходом для створення системи керування книжною бібліотекою.

По-перше, висока швидкодія інтерфейсу. Після початкового завантаження застосунку основні ресурси вже знаходяться в браузері, а надалі відбувається лише обмін даними з сервером. Це дозволяє значно скоротити час відгуку системи.

По-друге, покращений користувацький досвід. Користувач не втрачає контекст під час переходів між різними частинами застосунку, що робить роботу більш зручною та логічною.

По-третє, зручність роботи з формами. У бібліотечній системі велика кількість операцій пов’язана з введенням і редагуванням даних, тому можливість миттєвої валідації та оновлення інформації є важливою перевагою.

По-четверте, модульність. SPA дозволяє розділити систему на окремі компоненти (книги, користувачі, видача), що спрощує як розробку, так і подальшу підтримку.

По-п’яте, можливість масштабування. У разі потреби функціональність системи можна розширювати без кардинальної зміни архітектури.

Разом з тим, SPA-застосунки мають і певні обмеження, які необхідно враховувати при їх розробці [41]. По-перше, складність організації клієнтської логіки. У разі неправильного проектування структура коду може швидко ускладнюватися. По-друге, важливу роль відіграє правильна побудова API, через яке відбувається взаємодія з сервером. По-третє, можливі затримки під час початкового завантаження застосунку при великому обсязі ресурсів.

Незважаючи на це, для систем середнього масштабу, таких як бібліотечні застосунки, зазначені недоліки є контрольованими, а переваги значно переважають.

Основу розробки SPA становлять сучасні веб-технології. Ключову роль відіграє JavaScript, який забезпечує динамічну поведінку інтерфейсу, обробку подій, керування станом додатка та взаємодію з сервером . У поєднанні з HTML і CSS він дозволяє створювати інтерактивні та адаптивні інтерфейси.

Для спрощення розробки широко використовуються спеціалізовані фреймворки та бібліотеки, які реалізують основні принципи SPA:

React – це бібліотека JavaScript з відкритим кодом, яка використовується спеціально для побудови користувальницьких інтерфейсів. Основна перевага React JS полягає в тому, що він масштабований, простий та швидкий. React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux [10];

Vue.js – прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створено придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня представлення (view), що спрощує інтеграцію з іншими бібліотеками, та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторонніх додатків, якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками [11];

Angular – це потужний фреймворк для створення динамічних веб-додатків, розроблений Google. Він забезпечує широкий набір інструментів для розробки клієнтської частини додатків з використанням мови TypeScript [12].

Svelte – це новий, швидкий та простий у використанні фреймворк для розробки SPA. Svelte має унікальний підхід до роботи зі статичними компонентами, що дозволяє зменшити розмір коду та підвищити продуктивність додатку. Це робить Svelte привабливим вибором для розробки

додатків з обмеженими ресурсами та потужностями розробки SPA, який забезпечує швидку та ефективну розробку веб-додатків [13].

Для взаємодії з сервером застосовуються технології REST API або GraphQL, що забезпечують ефективний обмін даними. На серверному боці часто використовується Node.js, який дозволяє створювати високопродуктивні застосунки та використовувати єдину мову програмування як на клієнті, так і на сервері .

Додатково у розробці SPA застосовуються інструменти для оптимізації та підтримки проєкту, зокрема системи керування залежностями, засоби збірки коду та системи контролю версій.

1.3 Вибір технологічного стеку та архітектури веб-додатку

Після аналізу сучасних підходів до електронного керування бібліотекою та огляду існуючих SPA-систем наступним етапом є вибір технологічного стеку та архітектури майбутнього веб-застосунку. Від цього вибору залежить, наскільки зручно буде працювати із системою та як легко її буде підтримувати, чи можна буде її розширювати в майбутньому.

У даній роботі важливо підібрати не просто популярні технології, а такі, які підходять для створення SPA-застосунку на тему керування бібліотекою.

Під час вибору технологій враховувалися такі моменти:

- простота реалізації;
- зрозуміла логіка побудови;
- можливість реалізації CRUD-операцій;
- зручність роботи з інтерфейсом;
- взаємодія з сервером;
- швидкість роботи;
- можливість подальшого розширення;
- наявність документації.

Оскільки застосунок працює з такими даними, як книги, користувачі та бібліотека користувача, важливо було обрати технології, які добре підходять для роботи зі списками, формами, пошуком і API. Клієнтська частина застосунку може бути реалізована за допомогою:

– HTML;
– CSS;
– JavaScript;
– React.

HTML і CSS використовуються як основа для створення інтерфейсу застосунку. HTML відповідає за структуру сторінок, тоді як CSS визначає їх зовнішній вигляд та стилізацію. У межах проекту за допомогою цих технологій може бути реалізовано основні елементи інтерфейсу, зокрема сторінки зі списками книг, картки книг, форми входу та реєстрації, навігацію по застосунку, а також модальні вікна. Завдяки використанню CSS інтерфейс стає більш зручним і зрозумілим для користувача, що є дуже важливим, оскільки системою можуть користуватися люди без технічного досвіду.

JavaScript відіграє ключову роль у забезпеченні функціональності застосунку. Саме він відповідає за обробку дій користувача, роботу форм, реалізацію пошуку та фільтрації, оновлення даних у режимі реального часу та взаємодію із серверною частиною. Завдяки цьому застосунок працює динамічно і не потребує постійного перезавантаження сторінки.

Для реалізації SPA-застосунку доцільно обрати React, оскільки він добре підходить для створення сучасних динамічних інтерфейсів. Основна ідея цього підходу полягає у поділі інтерфейсу на окремі незалежні компоненти. У даному проекті це проявляється через використання окремих сторінок (наприклад, головна сторінка, сторінка авторизації, сторінка користувача), а також компонентів, таких як картка книги, пошук або модальні вікна. Такий підхід дозволяє зробити структуру коду більш зрозумілою та уникнути зайвої складності.

Крім цього, використання React дає змогу повторно використовувати компоненти в різних частинах застосунку, зручно працювати зі станом (наприклад, даними користувача після авторизації), а також швидко оновлювати інтерфейс без повного перезавантаження сторінки. У проекті також можна застосувати Context API, який використовується для збереження глобальних даних і передачі їх між компонентами без необхідності ускладнювати структуру застосунку.

Для запуску та збірки проєкту використовується Vite – сучасний інструмент, який значно спрощує процес розробки [42]. Його використання дозволяє швидко запускати застосунок, миттєво бачити зміни після редагування коду та уникнути складних налаштувань середовища. Це особливо зручно в межах навчального проекту, оскільки дозволяє більше уваги приділити логіці роботи застосунку, а не технічним деталям налаштування.

Для бекенду можна використати:

– Node.js;

– Express.js.

Node.js дозволяє писати серверну частину застосунку на JavaScript, що є досить зручним рішенням, оскільки не потрібно використовувати окрему мову програмування для бекенду. Це спрощує розробку і робить код більш цілісним. Для створення серверної частини у проекті також використовується Express.js – легкий і зрозумілий фреймворк, який застосовується для побудови API. З його допомогою можна обробляти HTTP-запити, створювати маршрути, працювати з даними у форматі JSON та організовувати взаємодію між клієнтською і серверною частинами застосунку.

У межах даного проекту сервер виконує кілька основних функцій. Він відповідає за авторизацію користувачів, збереження даних, а також забезпечує передачу інформації між клієнтом і сервером. Завдяки цьому фронтенд може отримувати необхідні дані та відображати їх користувачу без зайвих затримок.

У даній реалізації для зберігання даних використовується JSON-файл. Це досить просте рішення, яке добре підходить для навчального або

демонстраційного проекту, оскільки не потребує додаткового налаштування бази даних і дозволяє швидко реалізувати збереження інформації. Крім того, такий підхід є зручним для початкової розробки, коли основна увага приділяється логіці застосунку.

Разом із тим, використання JSON-файлу має певні обмеження. Зокрема, така система складніше масштабується, не забезпечує повноцінних зв’язків між даними і може викликати труднощі при збільшенні обсягу інформації. Саме тому в подальшому доцільно розглянути перехід на повноцінну базу даних, наприклад MySQL або MongoDB, що дозволить підвищити надійність і ефективність роботи системи.

У проекті може бути використано клієнт-серверну архітектуру, яка є типовою для сучасних SPA-застосунків. Вона передбачає поділ системи на кілька частин: клієнтську (фронтенд), серверну (бекенд) та сховище даних. Такий підхід дозволяє чітко розмежувати відповідальність між різними частинами системи та спростити її підтримку.

Принцип роботи застосунку є досить простим і логічним. Користувач відкриває веб-застосунок у браузері, після чого відображається інтерфейс. У разі необхідності фронтенд надсилає запит на сервер, сервер обробляє його та повертає відповідь у форматі JSON. Після цього інтерфейс оновлюється без повного перезавантаження сторінки. Саме така модель взаємодії забезпечує швидку і зручну роботу застосунку.

Функціонально система повинна бути орієнтована на виконання основних задач бібліотеки [43]. У застосунку необхідно реалізувати перегляд списку книг на головній сторінці та за категоріями, можливість пошуку та фільтрації, авторизацію і реєстрацію користувачів, роботу зі сторінкою користувача, а також функціонал, пов’язаний з особистою бібліотекою. Такий набір можливостей є базовим, але достатнім для демонстрації роботи системи керування бібліотекою.

Навіть незважаючи на те, що проект є навчальним, у ньому необхідно врахувати базові аспекти безпеки. Зокрема, реалізовати перевірку введених

даних, обробку можливих помилок та обмеження доступу до окремих функцій. Також важливим є питання захисту даних користувачів, зокрема паролів, що повинні зберігатися у захищеному вигляді.

Таким чином, використання React, Vite, Node.js та Express є обґрунтованим рішенням для створення SPA-застосунку керування бібліотекою. Обраний підхід дозволяє поєднати простоту розробки з сучасними технологіями та забезпечує можливість подальшого розвитку системи.

Висновки до розділу 1

У першому розділі було розглянуто теоретичні аспекти розробки SPA-застосунку для керування книжною бібліотекою. Проведений аналіз показав, що сучасна бібліотечна сфера активно розвивається у напрямі цифровізації та переходу від традиційних методів ведення обліку до електронних систем.

Було встановлено, що електронне керування бібліотекою дозволяє значно спростити основні процеси, зокрема облік книжкового фонду, роботу з користувачами, оформлення видачі та повернення книг, а також пошук і обробку інформації. Це підвищує ефективність роботи бібліотекаря та покращує зручність для користувачів.

Було проаналізовано існуючі програмні рішення, такі як Koha, DSpace та Ex Libris Alma, які демонструють різні підходи до організації бібліотечних систем. Встановлено, що сучасні системи можуть відрізнятися за функціональністю, масштабом та архітектурою, проте всі вони орієнтовані на автоматизацію процесів та забезпечення швидкого доступу до інформації.

Особливу увагу було приділено SPA-підходу, який дозволяє створювати динамічні вебзастосунки з високою швидкодією та зручним інтерфейсом. Визначено основні переваги SPA, зокрема відсутність повного перезавантаження сторінки, швидку взаємодію з користувачем, зручність роботи з даними та можливість побудови складних інтерфейсів.

Також було розглянуто сучасні технології розробки SPA-застосунків, включаючи JavaScript, React, Vue, Angular та інші інструменти. Аналіз показав, що використання таких технологій дозволяє створювати гнучкі, масштабовані та ефективні системи.

На основі проведеного аналізу було обґрунтовано вибір технологічного стеку для реалізації власного проєкту, який включає використання React для клієнтської частини, Node.js та Express для серверної частини, а також JSON як тимчасового рішення для зберігання даних. Обрана архітектура відповідає сучасним вимогам до веб-застосунків і дозволяє забезпечити зручність використання та можливість подальшого розвитку системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ

2.1 Проєктування інформаційної моделі та бази даних вебдодатку

Проєктування інформаційної моделі є одним із найважливіших етапів створення будь-якої інформаційної системи, оскільки саме на цьому етапі визначаються структура даних, логіка взаємодії між окремими компонентами та принципи подальшого функціонування програмного продукту. Його виконання впливає в кінцевому рахунку на успішність програмного засобу серед кінцевих користувачів [14]. Для SPA-додатку, призначеного для управління книжковою бібліотекою, правильна організація даних має особливе значення, адже система повинна забезпечувати швидкий доступ до інформації, стабільну роботу інтерфейсу та коректну синхронізацію між клієнтською і серверною частинами.

У межах розробленого вебдодатку інформаційна модель формувалася з урахуванням особливостей предметної області та архітектури SPA. Основною метою проєктування було визначення таких сутностей і зв’язків між ними, які дозволяють коректно організувати роботу користувача з книжковим каталогом, забезпечити збереження даних у базі та підтримати подальше розширення функціональних можливостей системи. Оскільки SPA-додаток працює без повного перезавантаження сторінки, важливим є не лише описати структуру даних, а й врахувати логіку взаємодії між клієнтською частиною, сервером і базою даних.

Першим етапом проєктування стало визначення основних сутностей системи та зв’язків між ними. Для цього було побудовано діаграму класів (див. рис. 2.1.), яка відображає узагальнену структуру вебдодатку та основні об’єкти, що беруть участь у його роботі. Така діаграма дозволяє показати, які елементи формують систему, як вони пов’язані між собою та яку роль виконують у загальній логіці функціонування SPA-додатку.

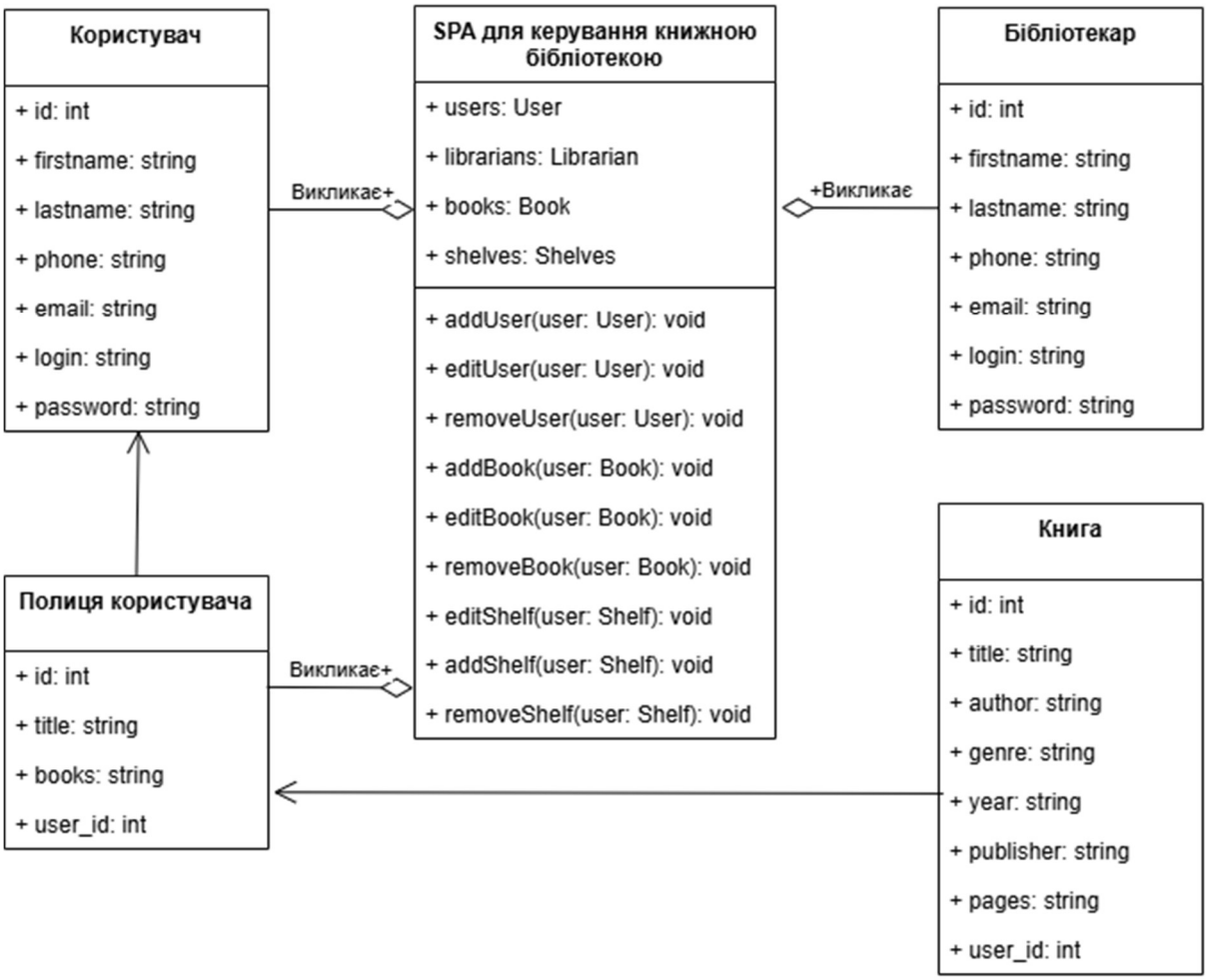


Рис. 2.1 Діаграма класів SPA-додатку для керування книжною бібліотекою
(розроблено автором)

Центральним елементом діаграми є клас «SPA для керування книжною бібліотекою», який відображає саму програмну систему. Цей клас виконує роль посередника між основними об’єктами предметної області. Через нього користувач і бібліотекар взаємодіють із каталогом книг, персональною полицею та функціями керування бібліотекою. До складу цього класу входять такі функціональні компоненти, як обробка запитів користувача, передача даних між інтерфейсом і серверною частиною, виконання операцій авторизації, оновлення інформації про книги та відображення змін у користувацькому інтерфейсі.

Клас «Користувач» описує звичайного читача, який працює із системою. Він є одним із головних сутностей вебдодатку, оскільки саме користувач

переглядає каталог, авторизується в системі та формує власну книжкову полицю. До основних атрибутів цього класу можна віднести ідентифікатор користувача, ім’я, логін, пароль, електронну пошту та номер телефону. Наявність цих атрибутів дозволяє системі ідентифікувати користувача, зберігати його облікові дані та забезпечувати доступ до персоналізованих функцій.

Клас «Полиця користувача» відображає персональну бібліотеку користувача. Він пов’язаний із класом «Користувач», оскільки кожна полиця належить конкретному читачеві та використовується для зберігання книг, доданих до його акаунта. Основними атрибутами цього класу є ідентифікатор книги, назва полиці та id читача. Завдяки цьому класу система може фіксувати, які саме книги користувач додав до власної бібліотеки, а також підтримувати поділ книг за окремими полицями або категоріями.

Клас «Книга» є важливим об’єктом предметної області, оскільки саме книжковий фонд становить основний зміст вебдодатку. Цей клас містить інформацію, необхідну для відображення книги в каталозі та її подальшого використання в системі. До атрибутів класу «Книга» належать ідентифікатор книги, назва, автор, жанр, опис, рік видання, видавництво, кількість сторінок, обкладинка та статус книги. Такі атрибути дозволяють повноцінно описати книжковий запис, забезпечити пошук за основними параметрами та відображати детальну інформацію про книгу на сторінці вебдодатку.

Клас «Бібліотекар» описує адміністративного учасника системи, який відповідає за підтримку актуальності книжкового фонду. На відміну від звичайного користувача, бібліотекар має розширені можливості роботи з даними про книги. До атрибутів цього класу можна віднести ім’я, ідентифікатор бібліотекаря, логін, пароль, роль у системі та рівень доступу. Бібліотекар взаємодіє з класом «Книга», оскільки саме він може додавати нові книжкові записи, редагувати наявну інформацію або видаляти книгу з каталогу.

Побудована діаграма класів дає змогу узгодити логіку програмної реалізації з інформаційною моделлю системи. Вона показує, що вебдодаток має

логічну структуру, у якій кожен об’єкт виконує окрему функцію, але водночас пов’язаний з іншими частинами системи.

Після визначення класів системи доцільно розглянути функціональні можливості вебдодатку з погляду користувачів. Для цього використовується UML-діаграма прецедентів (див. рис. 2.2), яка відображає основні сценарії взаємодії користувачів із системою. Якщо діаграма класів показує внутрішню структуру об’єктів, то діаграма прецедентів демонструє, які дії можуть виконувати учасники системи та які функції доступні для різних ролей.

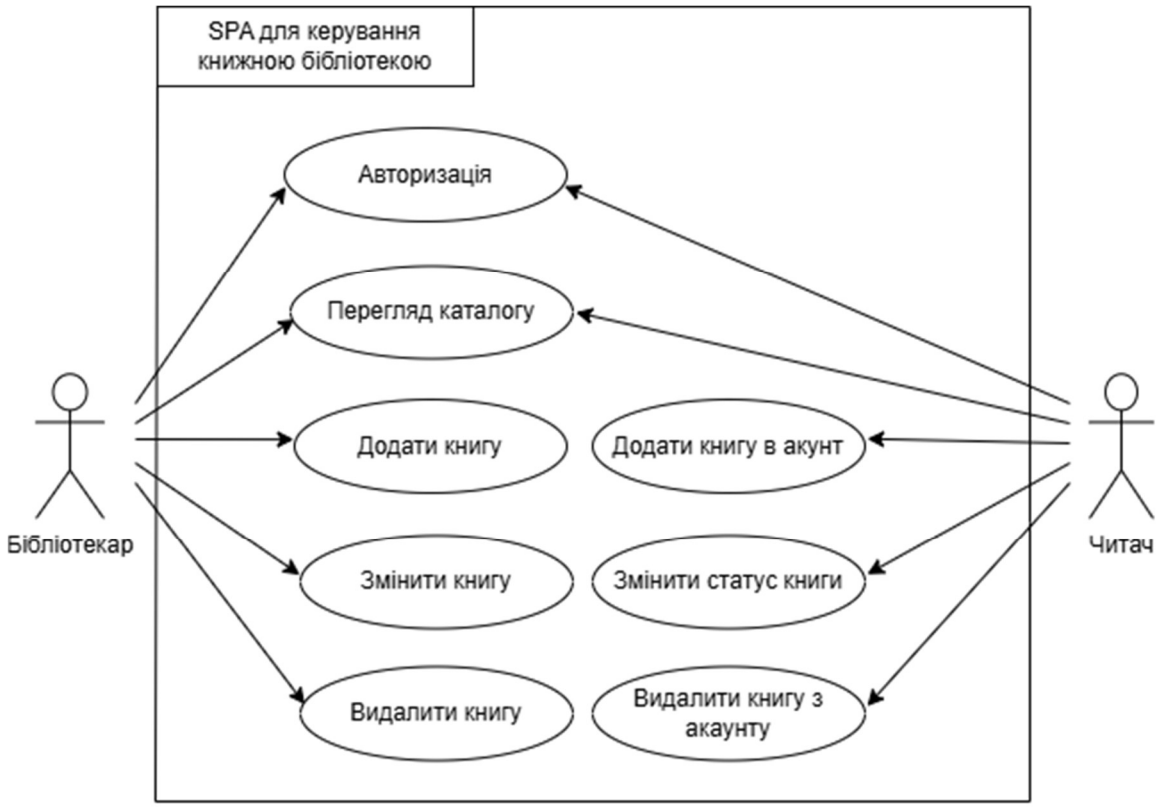


Рис. 2.2. UML-діаграма прецедентів SPA-додатку для керування книжною бібліотекою

(розроблено автором)

У системі передбачено дві основні ролі: користувач і бібліотекар. Користувач взаємодіє з вебдодатком через функції авторизації, перегляду каталогу, додавання книги до акаунта, зміни статусу книги на полицях та видалення книги з акаунта. Це відповідає основному сценарію використання

системи звичайним читачем, який заходить до додатку, переглядає доступні книги та формує власну бібліотеку.

Бібліотекар у межах системи виконує адміністративні дії, пов’язані з керуванням книжковим фондом. До таких дій належать авторизація, перегляд каталогу, додавання книги в систему, зміна інформації про книгу, видалення книги з системи. Наявність окремої ролі бібліотекаря дозволяє розмежувати права доступу між звичайним користувачем і адміністратором. Це важливо для коректної організації роботи вебдодатку, оскільки користувач повинен мати доступ лише до власної бібліотеки та перегляду каталогу, тоді як бібліотекар відповідає за підтримку актуальності загального книжкового фонду.

Для детальнішого опису процесу взаємодії між окремими частинами системи було побудовано діаграму послідовності прецедента додавання книги бібліотекарем (див. рис. 2.3.). Вона дозволяє простежити, як саме виконується ця дія в системі: від ініціювання запиту бібліотекарем до обробки цього запиту на сервері та звернення до бази даних. Така діаграма є важливою для SPA-додатку, оскільки демонструє зв’язок між frontend-частиною, backend-частиною та базою даних. Особливе значення ця діаграма має для візуалізації логіки адміністративних дій бібліотекаря. Наприклад, під час додавання або редагування книги бібліотекар виконує авторизацію, після чого вводить дані нової книги у frontend-інтерфейсі, потім вони передаються на backend, перевіряються та зберігаються в базі даних. Після успішного виконання операції система повертає оновлену інформацію, і зміни відображаються в інтерфейсі додатку (відповідна послідовність показана на рис. 2.3).

Завершальним етапом проєктування інформаційної моделі є створення ER-діаграми (див. рис. 2.4.), яка відображає структуру даних і взаємозв’язки між основними сутностями системи. Використання ER-моделі дозволяє наочно відобразити логіку організації даних, спростити процес подальшого проєктування та забезпечити цілісність структури вебдодатку. Головними компонентами ER-моделей є сутності, їх атрибути та зв’язки між сутностями [15].

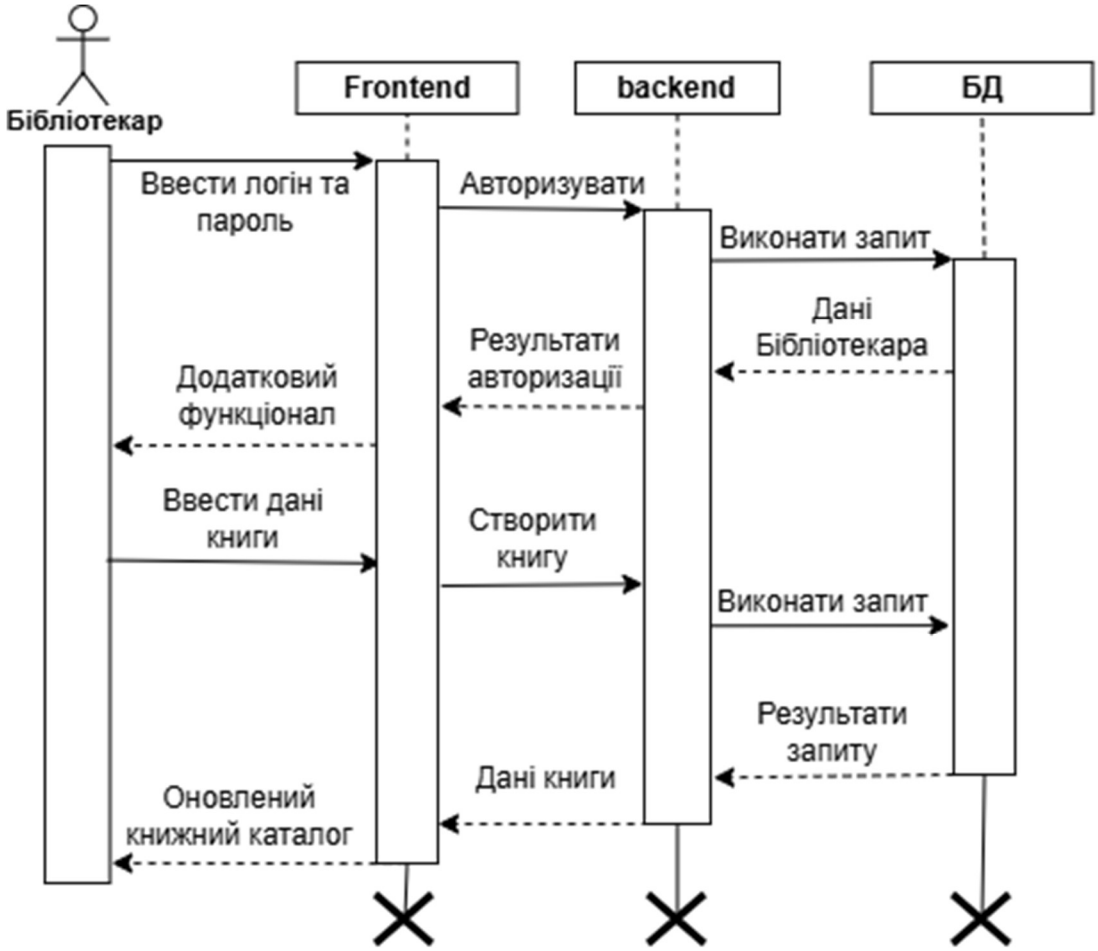


Рис. 2.3. Діаграма послідовності прецедента додавання нової книги у SPA-додатку для керування книжною бібліотекою

(розроблено автором)

До ключових сутностей SPA-додатку для керування книжною бібліотекою належать користувач, книга, полиця користувача та бібліотекар. Сутність «Користувач» відповідає за зберігання інформації про зареєстрованого читача та його взаємодію з системою. Сутність «Книга» містить дані про книжковий фонд, а сутність «Полиця користувача» використовується для організації персональної бібліотеки, до якої користувач може додавати книги.

Зв’язок між користувачем і полицею користувача відображає те, що один користувач може мати власну персональну бібліотеку з декількома полицями, у яких зберігаються обрані книги. Зв’язок між полицею користувача та книгою показує, що одна полиця може містити багато книг, а одна й та сама книга може бути додана до бібліотек різних користувачів.

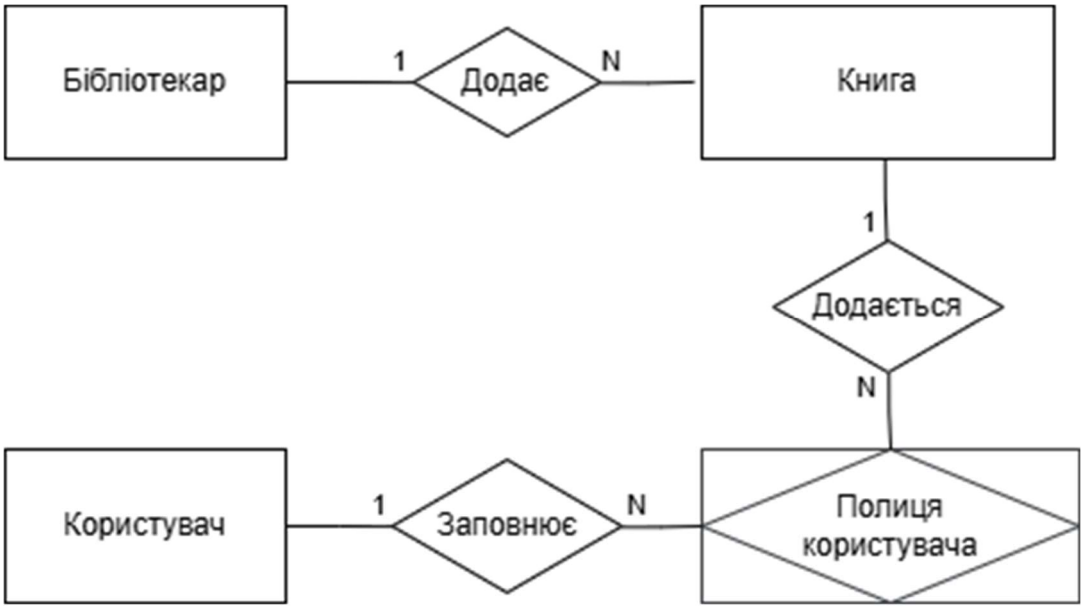


Рис. 2.4. ER-діаграма інформаційної моделі SPA-додатку для керування книжною бібліотекою

(розроблено автором)

Окремо в ER-діаграмі відображено роль бібліотекаря, який взаємодіє з книжковим фондом і підтримує актуальність даних про книги. Це відповідає функціональним можливостям, показаним на діаграмі прецедентів, де бібліотекар має право додавати, змінювати та видаляти книги. Завдяки цьому інформаційна модель поєднує як користувацьку, так і адміністративну логіку роботи системи.

У поточній реалізації вебдодатку для зберігання даних використовується реляційна база даних SQLite. На серверній частині створюється підключення до бази даних, після чого backend виконує запити на отримання, додавання, редагування або видалення даних. Структура бази даних реалізована через таблиці, що відповідають основним сутностям інформаційної моделі. Таблиця користувачів зберігає дані облікових записів, таблиця книг містить інформацію про книжковий каталог, а проміжна таблиця персональної бібліотеки фіксує зв’язок між користувачами та книгами, які вони додали до власного акаунта.

Важливою складовою інформаційної моделі також є механізм авторизації. Після входу до системи користувач отримує доступ до персоналізованих

функцій, зокрема до власної бібліотеки та можливості додавати або видаляти книги з акаунта. Для бібліотекаря авторизація є необхідною умовою доступу до адміністративних функцій, пов’язаних із керуванням книжковим фондом. Завдяки цьому система забезпечує розмежування можливостей між різними ролями та підтримує коректну логіку роботи вебдодатку.

2.2 Опис основних алгоритмів та методів збору та обробки даних

У процесі розробки SPA-додатку для керування книжною бібліотекою важливе значення має не лише структура даних, а й алгоритми їх обробки. Алгоритм – це послідовність обчислювальних кроків, які перетворюють вхідні дані у вихідні [16]. Саме алгоритмічна логіка визначає, яким чином користувач взаємодіє із системою, як відбувається отримання, перевірка, збереження та оновлення інформації, а також як клієнтська частина обмінюється даними із сервером.

Основним джерелом даних у розробленому веб-додатку є база даних SQLite, з якою взаємодіє серверна частина на основі Node.js з Express.js. Дані про книги, користувачів і персональні бібліотеки зберігаються у таблицях бази даних, а клієнтська частина отримує необхідну інформацію через REST API. Такий підхід дозволяє відокремити логіку зберігання даних від логіки відображення інтерфейсу.

Алгоритм отримання книжкового каталогу починається із запиту клієнтської частини до маршруту /api/books. Сервер виконує SQL-запит до таблиці books, отримує список книг і повертає його у форматі JSON. Після цього React-застосунок зберігає отримані дані у стані контексту BooksContext, що дозволяє використовувати книжковий каталог на головній сторінці, сторінках категорій, у результатах пошуку та на сторінці детального перегляду книги.

Також, важливим алгоритмом є – авторизації користувача (див. рис. 2.5). Він забезпечує перевірку введених облікових даних і надання доступу до персоналізованих можливостей системи.

Рис. 2.5. Алгоритм авторизації користувача у SPA-додатку для керування книжною бібліотекою

(розроблено автором)

Користувач вводить логін і пароль на сторінці входу, після чого клієнтська частина надсилає відповідний POST-запит на сервер. Сервер перевіряє наявність користувача у таблиці users бази даних, порівнює введений пароль із хешованим значенням і, у разі успішної перевірки, генерує токен доступу. Отриманий токен передається клієнтській частині та зберігається у localStorage. Надалі він використовується для підтвердження автентичності користувача під час звернення до захищених API-маршрутів.

Наступним алгоритмом є отримання профілю користувача після авторизації. Після збереження токена клієнтська частина може звертатися до захищеного маршруту профілю. При такому запиті токен додається до заголовка Authorization, а серверна частина перевіряє його через middleware. Якщо токен є коректним, сервер повертає дані користувача, які потім зберігаються у глобальному стані React-застосунку. Якщо токен відсутній або недійсний, система очищує локальні дані авторизації та повертає користувача до неавторизованого стану.

Особливу роль у роботі додатку відіграє алгоритм пошуку книг (див. рис. 2.6). Пошук книг реалізовано на основі отриманих із сервера даних.

Рис. 2.6. Алгоритм пошуку книги у каталозі

(розроблено автором)

Користувач вводить пошуковий запит, після чого система порівнює його з основними характеристиками книги: назвою, автором, жанром і

видавництвом. Також у серверній частині передбачено окремий API-маршрут /api/books-search, який може виконувати пошук безпосередньо на рівні бази даних за допомогою SQL-оператора LIKE. Це створює можливість подальшого розширення пошукової логіки та перенесення більшої частини обробки на backend.

Для зручності користувача пошук виконується без жорсткої прив’язки до регістру, тобто великі та малі літери не впливають на результат. Якщо знайдено збіги, вони відображаються на сторінці результатів у вигляді карток книг. Якщо відповідних записів немає, користувач отримує повідомлення про відсутність результатів.

Ще одним важливим алгоритмом є додавання книги до персональної бібліотеки користувача, який представлений на рис. 2.7.

Рис. 2.7. Алгоритм додавання книги до персональної бібліотеки

(розроблено автором)

Після переходу на сторінку детального перегляду книги користувач може додати її до однієї з полиць. Клієнтська частина передає на сервер ідентифікатор книги та назву обраної полиці, після чого сервер зберігає відповідний запис у таблиці user_books. У результаті формується зв’язок між конкретним користувачем, книгою та її статусом у персональній бібліотеці. Такий підхід є раціональним, оскільки вся основна інформація про книгу вже міститься у загальному каталозі, а персональна бібліотека лише фіксує зв’язок між користувачем, книгою та відповідною полицею.

Під час додавання книги система також перевіряє, чи не міститься вона вже в іншій категорії. Якщо книга переноситься з однієї полиці до іншої, попередній статус оновлюється. Це дозволяє уникнути ситуації, коли одна й та сама книга одночасно перебуває в кількох несумісних категоріях, наприклад «прочитано» і «заплановано до читання». Завдяки цьому персональна бібліотека залишається логічно-впорядкованою.

Методи збору даних у межах даного веб-додатку можна розглядати як поєднання клієнтських і серверних механізмів. Клієнтська частина збирає дані з форм введення, наприклад логін, пароль або пошуковий запит. Також вона отримує дані про дії користувача: вибір книги, перехід між сторінками, додавання або видалення книги з полиці. Серверна частина, у свою чергу, обробляє отриману інформацію, виконує SQL-запити до бази даних, перевіряє коректність введених даних і повертає результат у форматі JSON.

Таким чином, основні алгоритми веб-додатку забезпечують повний цикл взаємодії користувача із системою: від введення даних і авторизації до пошуку книг, оновлення персональної бібліотеки та збереження її стану в базі даних. Використання SQLite, REST API, Express-сервера, Context API та localStorage для збереження токена авторизації дозволяє організувати стабільну роботу системи, забезпечити швидке оновлення інтерфейсу та підтримати логічну цілісність даних. Завдяки цьому розроблений SPA-додаток має зрозумілу алгоритмічну структуру та може бути розширений у майбутньому за рахунок додавання нових модулів або розширення структури бази даних.

2.3 Реалізація серверної та клієнтської частини вебдодатку

Після визначення основних алгоритмів роботи системи наступним етапом є практична реалізація частин веб-додатку. Розроблений SPA-додаток побудований за клієнт-серверним принципом. Клієнтська частина відповідає за відображення інтерфейсу, навігацію між сторінками, роботу з каталогом книг, пошуком і персональною бібліотекою користувача. Серверна частина забезпечує обробку запитів, реєстрацію та авторизацію користувачів, перевірку токена доступу й роботу з базою даних SQLite.

Для реалізації клієнтської частини було використано React, React Router, Context API, Vite, JavaScript, HTML і CSS. React забезпечує побудову інтерфейсу на основі окремих компонентів, React Router відповідає за маршрутизацію між сторінками без повного перезавантаження веб-додатку, а Context API використовується для централізованого збереження стану авторизації та персональної бібліотеки. Vite застосовувався як інструмент запуску та збирання проєкту, що спрощує процес розробки й тестування застосунку.

Серверна частина реалізована на основі Node.js з Express.js. Вона виконує функції обробки API-запитів, перевірки введених користувачем даних, хешування паролів, створення авторизаційних токенів і повернення відповідей клієнтській частині у форматі JSON.

Код, який відповідає за початкову реалізацію структури бази даних міститься у файлі database.js. У ньому здійснюється підключення до SQLite, створення основних таблиць системи та початкове наповнення таблиці книг. Це дозволило перенести ключові дані вебдодатку з локальних структур до окремого файлового сховища бази даних. Фрагмент коду, що відповідає за початкову реалізацію структури бази даних представлений на рисунку 2.8.

Структура frontend-частини організована за принципом розподілу відповідальності між окремими папками та файлами проєкту. Такий підхід дозволяє не змішувати логіку сторінок, компонентів, стилів і даних, що

спрощує підтримку застосунку та робить його структуру зрозумілою для подальшого розвитку.

```
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS users (
      id TEXT PRIMARY KEY,
      login TEXT NOT NULL UNIQUE,
      password_hash TEXT NOT NULL,
      created_at TEXT NOT NULL
    )
  `);

  db.run(`
    CREATE TABLE IF NOT EXISTS books (
      id TEXT PRIMARY KEY,
      title TEXT NOT NULL,
      author TEXT,
      cover TEXT,
      description TEXT,
      genre TEXT,
      year INTEGER,
      publisher TEXT,
      pages INTEGER,
      category TEXT,
      section TEXT
    )
  `);

  db.run(`
    CREATE TABLE IF NOT EXISTS user_books (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id TEXT NOT NULL,
      book_id TEXT NOT NULL,
      book_data TEXT NOT NULL,
      shelf TEXT NOT NULL,
      created_at TEXT NOT NULL,
      updated_at TEXT NOT NULL,
      UNIQUE(user_id, book_id)
    )
  `);
```

Рис. 2.8. Фрагмент коду, що відповідає за початкову реалізацію структури бази даних

(розроблено автором)

У директорії components розміщуються багаторазові елементи інтерфейсу, pages містить основні сторінки веб-додатку, context відповідає за глобальний стан, окремі службові файли забезпечують роботу з даними та API-запитами, а style-файли забезпечують оформлення окремих частин інтерфейсу.

На рис. 2.9 наведено структуру клієнтської частини SPA-додатку, яка демонструє розміщення основних компонентів проєкту та їх функціональне призначення.

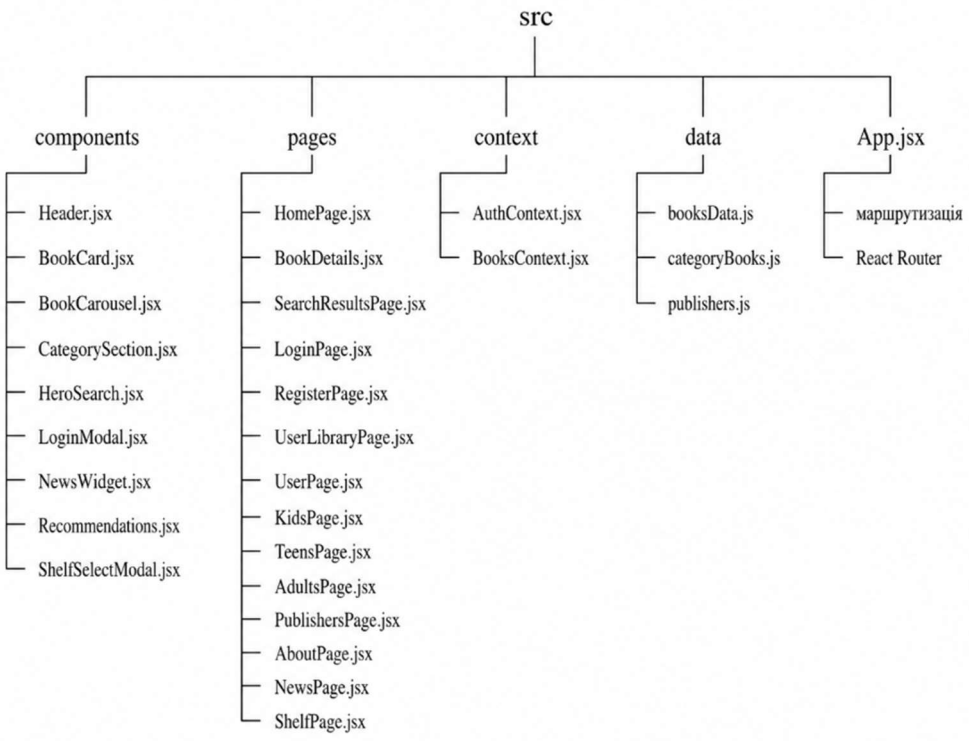


Рис. 2.9. Структура клієнтської частини SPA-додатку

(розроблено автором)

Основою клієнтської частини є файл App.jsx, у якому визначаються основні маршрути застосунку. Саме через нього організовується перехід між головною сторінкою, сторінками авторизації та реєстрації, сторінкою персональної бібліотеки, деталями книги, результатами пошуку та іншими розділами. Завдяки використанню React Router ці переходи виконуються без повного перезавантаження сторінки, що відповідає принципам SPA-архітектури та забезпечує більш плавну взаємодію користувача із системою.

Інтерфейс вебдодатку побудований за компонентним принципом. Окремі частини інтерфейсу реалізовані як самостійні React-компоненти, що дозволяє повторно використовувати їх у різних частинах застосунку. Зокрема, у межах проєкту окремими компонентами реалізовано навігаційну панель, картки книг,

форми авторизації, елементи пошуку, сторінку детального перегляду книги та блоки персональної бібліотеки. Така організація коду робить систему більш гнучкою та зручною для подальшого оновлення.

Для управління станом застосунку використовується Context API. Він забезпечує централізоване збереження інформації про авторизованого користувача, токен доступу та персональну бібліотеку. Завдяки цьому різні компоненти можуть отримувати необхідні дані без складної передачі через багато проміжних рівнів. Після зміни стану, наприклад після входу користувача або додавання книги до бібліотеки, інтерфейс автоматично оновлюється та відображає актуальні дані.

Однією з основних функцій серверної частини є авторизація користувача. Під час реєстрації пароль користувача не зберігається у відкритому вигляді, а перетворюється на хеш за допомогою bcryptjs. Під час входу до системи сервер порівнює введений пароль із хешованим значенням, збереженим у таблиці users бази даних SQLite. Якщо перевірка проходить успішно, сервер створює токен доступу та повертає його клієнтській частині.

На клієнтській стороні робота з авторизацією організована у файлі AuthContext.jsx (код файлу AuthContext.jsx представлений у Додатку А). Він зберігає інформацію про поточного користувача, токен доступу, стан бібліотеки та містить функції входу, реєстрації, виходу, додавання книги на полицю і видалення книги з бібліотеки. Завдяки цьому різні компоненти React-застосунку можуть отримувати актуальні дані без дублювання логіки.

Для отримання книжкового каталогу з серверної частини використовується BooksContext.jsx (код файлу BooksContext.jsx представлений у Додатку Б). Після запуску застосунку він надсилає запит до маршруту /api/books, отримує список книг із бази даних і передає ці дані компонентам інтерфейсу. Це дозволяє використовувати єдине джерело даних для головної сторінки, сторінок категорій, пошуку та детального перегляду книги.

Серверна частина веб-додатку реалізована у вигляді Express-сервера, який містить маршрути для реєстрації, входу до системи та отримання профілю

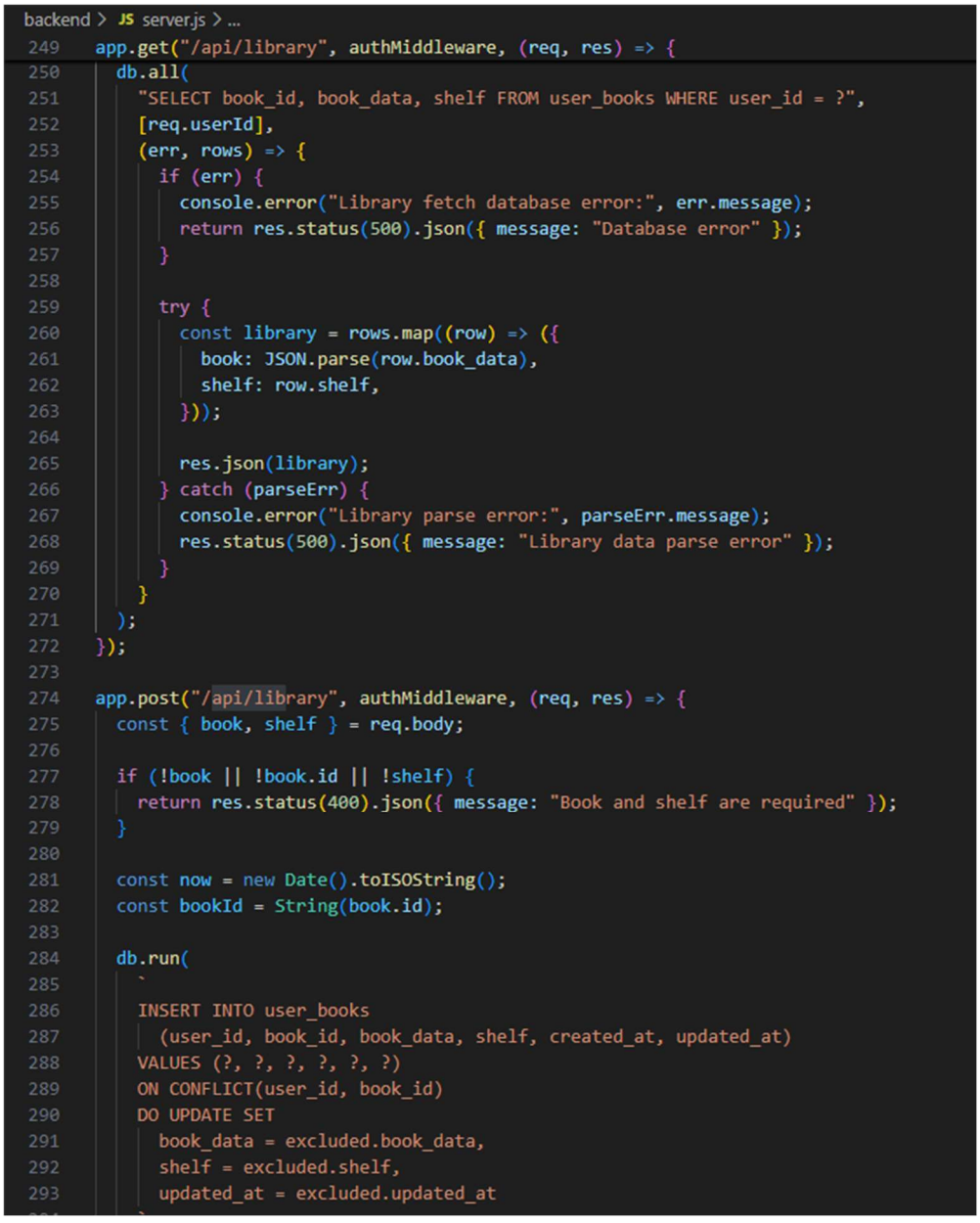
користувача. Для обробки JSON-запитів використовується express.json(), що дозволяє серверу приймати дані з клієнтської частини у зручному для JavaScript форматі. Також у серверній частині використовується механізм CORS (Cross-Origin Resource Sharing), що забезпечує коректну взаємодію між frontend і backend під час локальної розробки.

Для роботи з книжковим каталогом у серверній частині створено окремі API-маршрути (відповідний фрагмент коду представлений на рис. 2.10). Вони забезпечують отримання повного списку книг, пошук за ключовими полями, фільтрацію за секціями та категоріями, а також відкриття конкретної книги за її ідентифікатором. Усі ці операції виконуються через SQL-запити до таблиці books.

Рис. 2.10. Фрагмент коду, який відповідає за роботу з книжковим каталогом у серверній частині

(розроблено автором)

Окремий блок серверної логіки відповідає за персональну бібліотеку користувача (відповідний фрагмент коду представлений на рис. 2.11). Після авторизації користувач може додавати книгу на полицю або видаляти її зі своєї бібліотеки. Дані цих операцій зберігаються у таблиці user_books, що дозволяє пов’язати користувача з обраними книгами та їхнім статусом у бібліотеці.



```
backend > JS server.js > ...
249  app.get("/api/library", authMiddleware, (req, res) => {
250    db.all(
251      "SELECT book_id, book_data, shelf FROM user_books WHERE user_id = ?",
252      [req.userId],
253      (err, rows) => {
254        if (err) {
255          console.error("Library fetch database error:", err.message);
256          return res.status(500).json({ message: "Database error" });
257        }
258
259        try {
260          const library = rows.map((row) => ({
261            book: JSON.parse(row.book_data),
262            shelf: row.shelf,
263          }));
264
265          res.json(library);
266        } catch (parseErr) {
267          console.error("Library parse error:", parseErr.message);
268          res.status(500).json({ message: "Library data parse error" });
269        }
270      }
271    );
272  });
273
274  app.post("/api/library", authMiddleware, (req, res) => {
275    const { book, shelf } = req.body;
276
277    if (!book || !book.id || !shelf) {
278      return res.status(400).json({ message: "Book and shelf are required" });
279    }
280
281    const now = new Date().toISOString();
282    const bookId = String(book.id);
283
284    db.run(
285      `
286      INSERT INTO user_books
287        (user_id, book_id, book_data, shelf, created_at, updated_at)
288      VALUES (?, ?, ?, ?, ?, ?)
289      ON CONFLICT(user_id, book_id)
290      DO UPDATE SET
291        book_data = excluded.book_data,
292        shelf = excluded.shelf,
293        updated_at = excluded.updated_at
```

Рис. 2.11. фрагмент коду, що відповідає за персональну бібліотеку користувача

(розроблено автором)

Для перевірки доступу до захищених маршрутів використовується middleware авторизації. Він аналізує заголовок Authorization, отримує токен і перевіряє його наявність у серверному об’єкті сесій. Якщо токен є коректним, користувач отримує доступ до відповідного API-маршруту. Якщо токен відсутній або недійсний, сервер повертає повідомлення про помилку авторизації.

Таким чином, компонентна структура інтерфейсу, маршрутизація та централізоване управління станом дозволили створити цілісну систему, у якій окремі частини працюють узгоджено.

Висновки до розділу 2

У процесі проєктування інформаційної моделі було визначено ключові сутності системи, серед яких Користувач, Бібліотекар, Книга, Полиця користувача. Для наочного представлення взаємозв’язків між ними були використані UML-діаграми які дозволили відобразити загальну логіку організації даних у системі.

Окрему увагу було приділено взаємодії frontend і backend-частин. Клієнтська частина відповідає за відображення інтерфейсу, навігацію, роботу з каталогом книг, пошук і персональну бібліотеку користувача. Серверна частина реалізує обробку запитів, реєстрацію, авторизацію, перевірку токена доступу та роботу з даними користувачів. Обмін між частинами системи здійснюється через REST API у форматі JSON, що є зручним рішенням для JavaScript-застосунків і дозволяє чітко розділити відповідальність між клієнтом і сервером.

Важливим результатом стало впровадження бази даних SQLite для зберігання основної інформації веб-додатку. У межах реалізації були використані таблиці users, books та user_books, які забезпечують збереження облікових записів користувачів, книжкового каталогу та персональної бібліотеки. Такий підхід є надійним і структурованим, оскільки дозволяє

централізовано зберігати дані, виконувати SQL-запити, уникати дублювання записів і спрощувати подальше масштабування системи.

У результаті було сформовано технічну основу SPA-додатку для керування книжною бібліотекою та визначено, як саме організовуються дані, яким чином користувач взаємодіє із системою, які алгоритми забезпечують роботу її основних функцій.

РОЗДІЛ 3
ФУНКЦІОНУВАННЯ ТА ЕЛЕМЕНТИ ІНТЕРФЕЙСУ SPA ДЛЯ КЕРУВАННЯ КНИЖНОЮ БІБЛІОТЕКОЮ

3.1 Огляд реалізованого функціоналу розробленого SPA

На завершальному етапі розробки було сформовано повноцінний SPA-вебдодаток для керування книжною бібліотекою, який поєднує користувацьку частину, каталог книг, персональну бібліотеку читача та адміністративні можливості бібліотекаря. Розроблений застосунок має назву MyLibr і призначений для зручного перегляду книжкового фонду, пошуку літератури, роботи з персональними книжковими полицями та керування записами про книги через адміністративну панель.

Підсумовуючи, стек технологій розробленого вебдодатку включає React, React Router, Context API, Vite, JavaScript, HTML і CSS для клієнтської частини, а також Node.js, Express.js, SQLite, bcryptjs, cors і multer для серверної частини. Такий набір технологій дозволив реалізувати клієнт-серверну архітектуру, у якій frontend відповідає за інтерфейс і взаємодію з користувачем [17], а backend – за обробку запитів, авторизацію, збереження даних і роботу з базою даних [18].

Головна сторінка вебдодатку виконує роль стартової точки для користувача. Вона містить візуальний блок із пошуком, добірки книг та блок новин бібліотеки. Книжкові добірки поділено на окремі секції: «Гарячі новинки», «Хіти цього місяця» та «Рекомендації» (див. рис. 3.1). Такий підхід робить головну сторінку не просто списком книг, а зручним навігаційним простором, з якого користувач може швидко перейти до зацікавленої книги й орієнтуватися у величезному потоці літературних новинок [19].

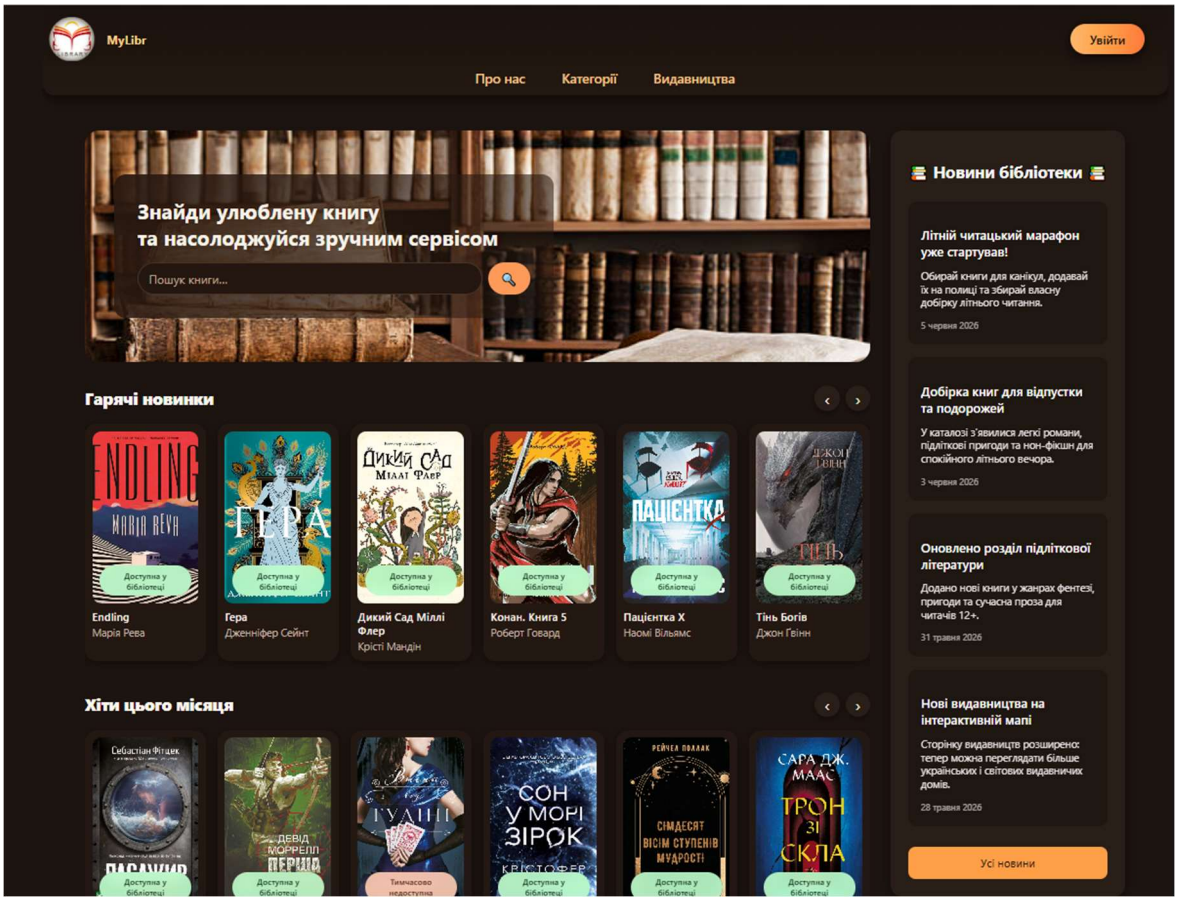


Рис. 3.1. Головна сторінка MyLibr

(розроблено автором)

Кожна книга відображається у вигляді картки (див. рис. 3.2.), що містить у собі обкладинку, назву, автора. Якщо обкладинка відсутня або не завантажується, система відображає резервний візуальний блок, з великою літерою, що запобігає порушенню структури сторінки.

Окремо реалізовано сторінку детального перегляду книги. На ній користувач може побачити повну інформацію про видання: назву, автора, жанр, рік, видавництво, кількість сторінок, опис, категорію та статус доступності.

Також у картці передбачено індикатор доступності книги [20]: «Доступна у бібліотеці» або «Тимчасово недоступна». Це дозволяє користувачу одразу отримати базову інформацію про стан книги у бібліотеці ще до відкриття її детальної сторінки.

Рис. 3.2 Картка книжки на сайті

(розроблено автором)

Важливою частиною вебдодатку є система категорій. У навігації передбачено розділи для дітей, підлітків і дорослих (див. рис. 3.3.). Сторінка дитячої літератури містить навчальні книги та пригодницькі видання (див. рис. 3.4.). Сторінка підліткової літератури поділяє книги на шкільну програму та жанрові добірки (див. рис. 3.5.). Сторінка літератури для дорослих містить сучасну прозу, нон-фікшн і книги для саморозвитку (див. рис. 3.6.). Такий поділ спрощує орієнтацію в каталозі та дозволяє користувачу не переглядати весь книжковий фонд одразу.

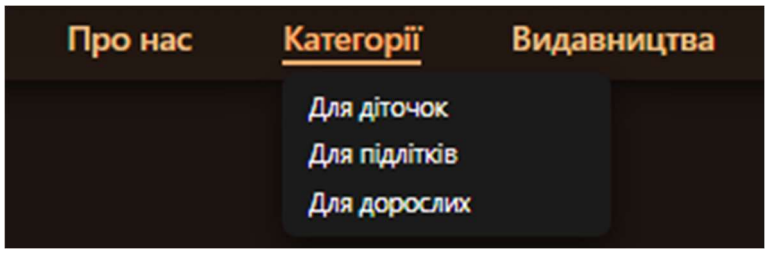


Рис. 3.3. Розділи категорій у вебдодатку

(розроблено автором)

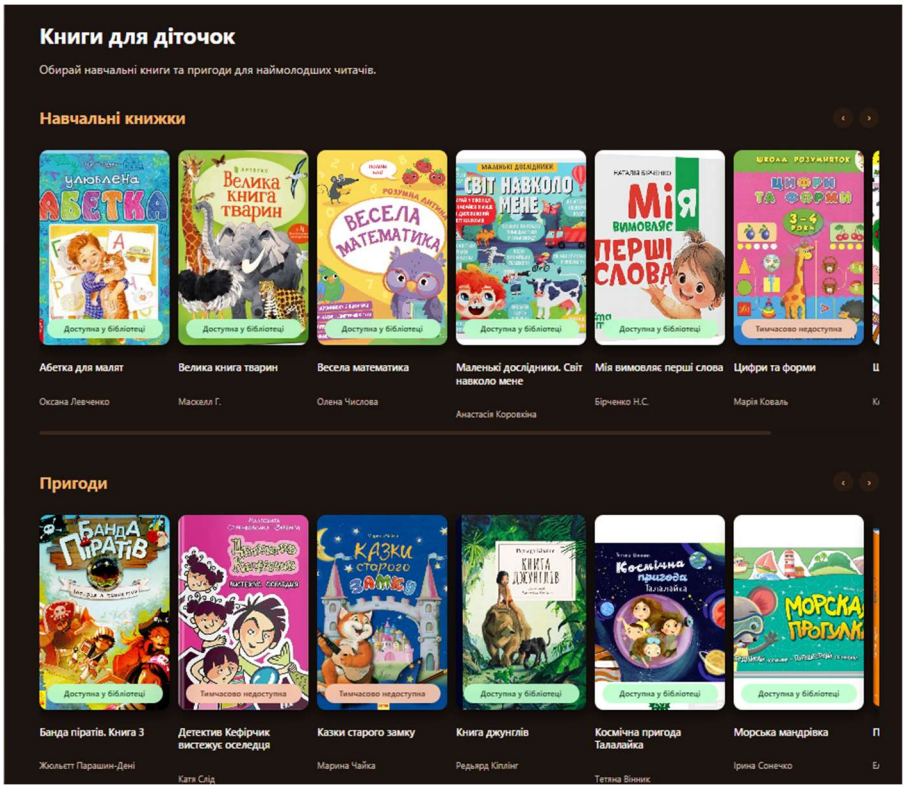


Рис. 3.4. Категорія «Для діточок»

(розроблено автором)

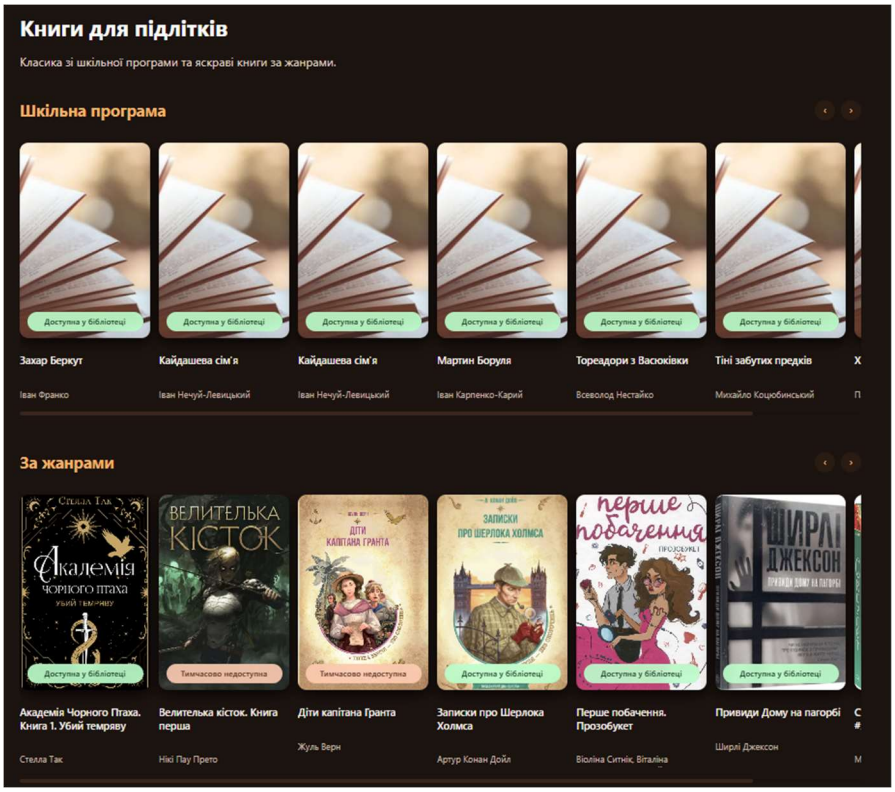


Рис. 3.5. Категорія «Для підлітків»

(розроблено автором)

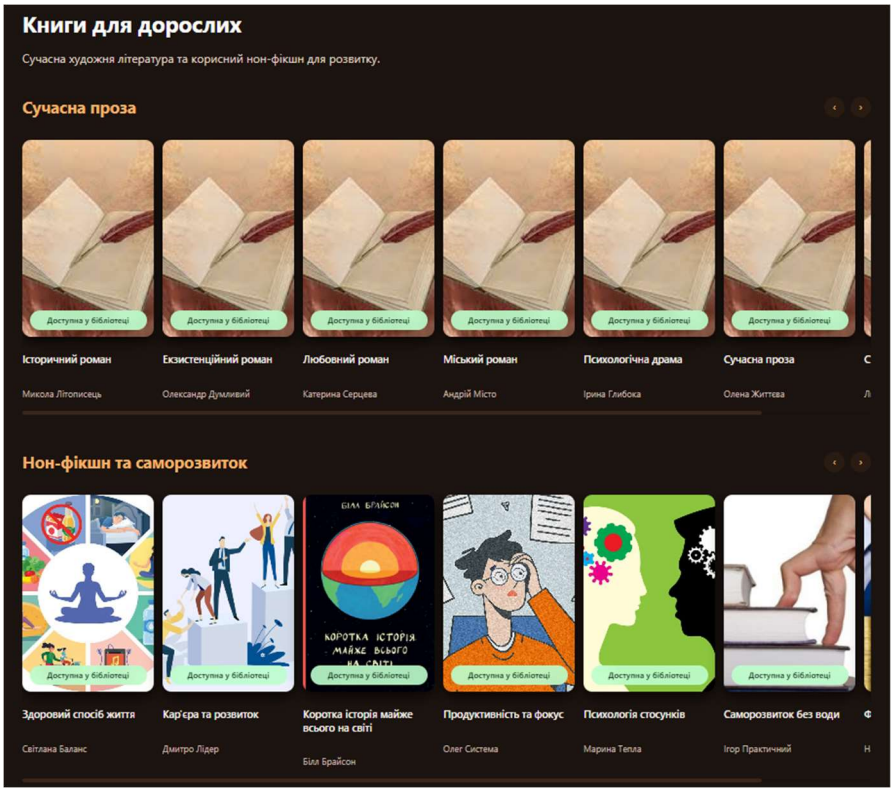


Рис. 3.6. Категорія «Для дорослих»

(розроблено автором)

Не менш значним є реалізація пошуку книг (див. рис. 3.7.) через окрему сторінку результатів.

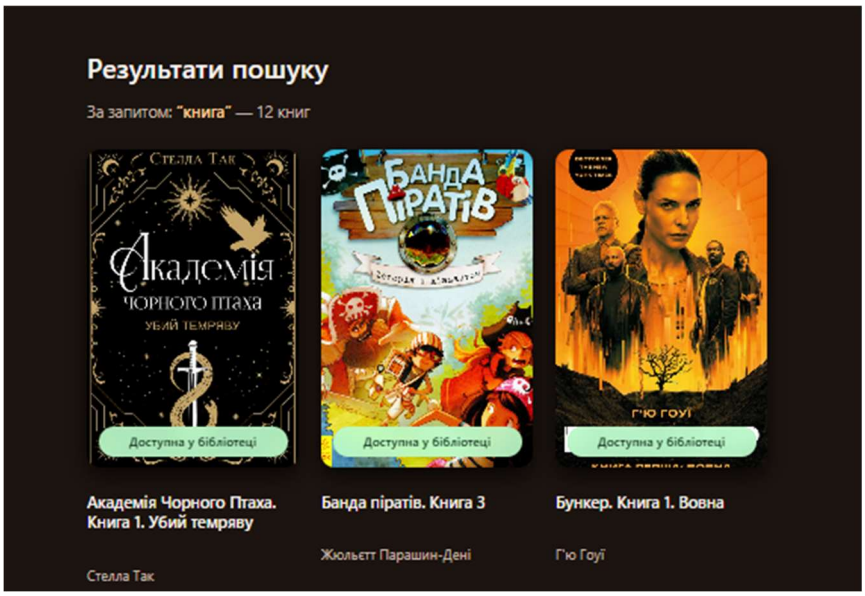


Рис. 3.7. Реалізація пошуку книг у вебдодатку

(розроблено автором)

Користувач вводить запит, після чого система порівнює його з основними характеристиками книг, зокрема назвою, автором, жанром, видавництвом і категорією. Якщо збіги знайдено, результати відображаються у вигляді сітки книжкових карток. Якщо ж збігів не знайдено, користувач отримує повідомлення про відсутність результатів і може повернутися на головну сторінку. Це значно полегшує пошук рішень і робить його більш передбачуваним [21].

У вебдодатку також реалізовано сторінку видавництв (див. рис. 3.8.). Вона містить перелік українських і зарубіжних видавництв, інформацію про країну, місто, адресу, короткий опис і приклади відомих книг. Для зручності користувача додано фільтри за країною, містом і конкретним видавництвом. Після вибору видавництва на сторінці відображається інтерактивна карта Google Maps із точним місцем розташування. Це розширює інформаційну цінність вебдодатку, оскільки користувач може не лише переглядати книги, а й ознайомлюватися з видавничими організаціями.

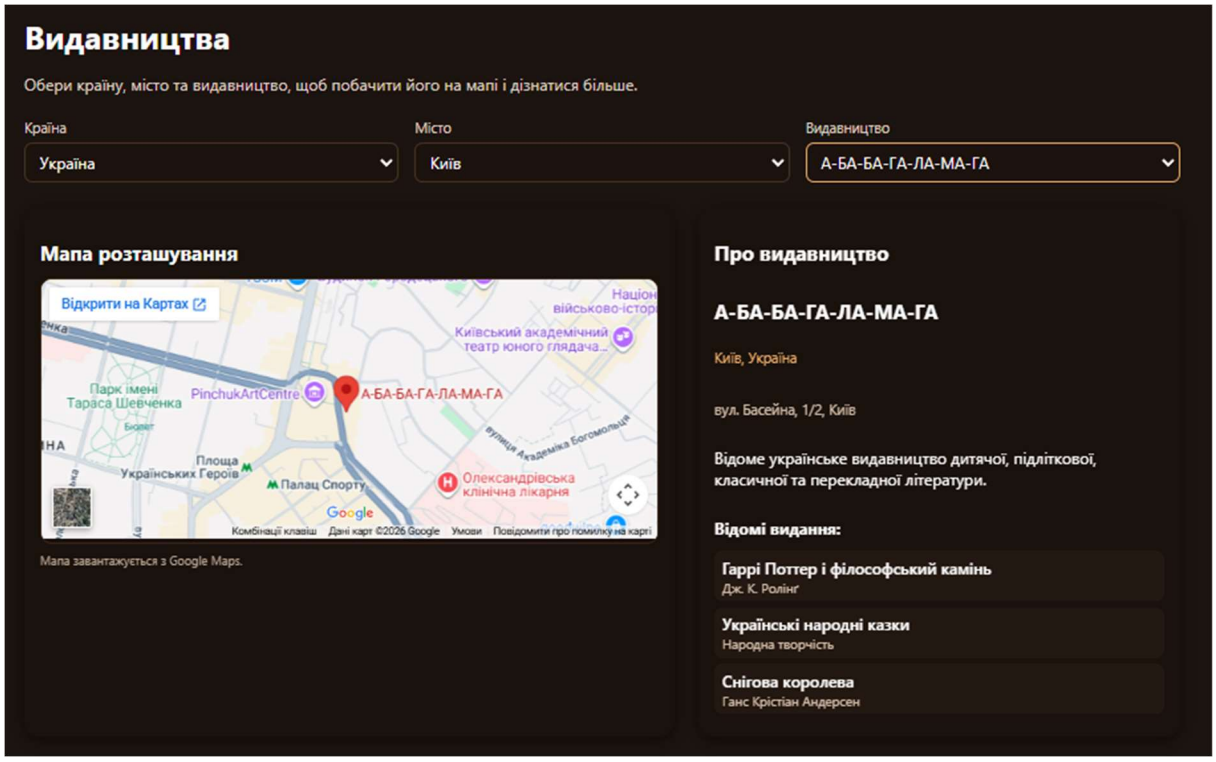


Рис. 3.8. Реалізація сторінки видавництв

(розроблено автором)

Додатковим інформаційним елементом є блок новин бібліотеки. На головній сторінці відображаються останні новини, а повний список доступний на окремій сторінці. Новини містять короткі повідомлення про оновлення каталогу, сезонні добірки, читацькі марафони та зміни у структурі вебдодатку [22]. Це створює ефект живої бібліотечної системи, яка регулярно оновлюється та підтримує комунікацію з користувачами.

Система авторизації включає у себе реєстрацію, вхід і вихід користувача [23]. Під час реєстрації створюється новий обліковий запис, а пароль зберігається у хешованому вигляді. Це дозволяє розмежувати можливості гостя, авторизованого користувача та адміністратора.

Для авторизованого користувача реалізовано сторінку персональної бібліотеки (див. рис. 3.9.). Вона містить набір полиць: «Усі книжки», «Заплановано», «Прочитано», «Улюблені» та «Не зайшли». Користувач може додавати книги до відповідної полиці, змінювати їхній статус і видаляти книги зі своєї бібліотеки. Кількість книг на кожній полиці відображається безпосередньо на сторінці профілю, що дозволяє швидко оцінити стан персональної бібліотеки [24]. Також для авторизованого користувача доступні дії з додавання книги до персональної бібліотеки або видалення її з власного списку.

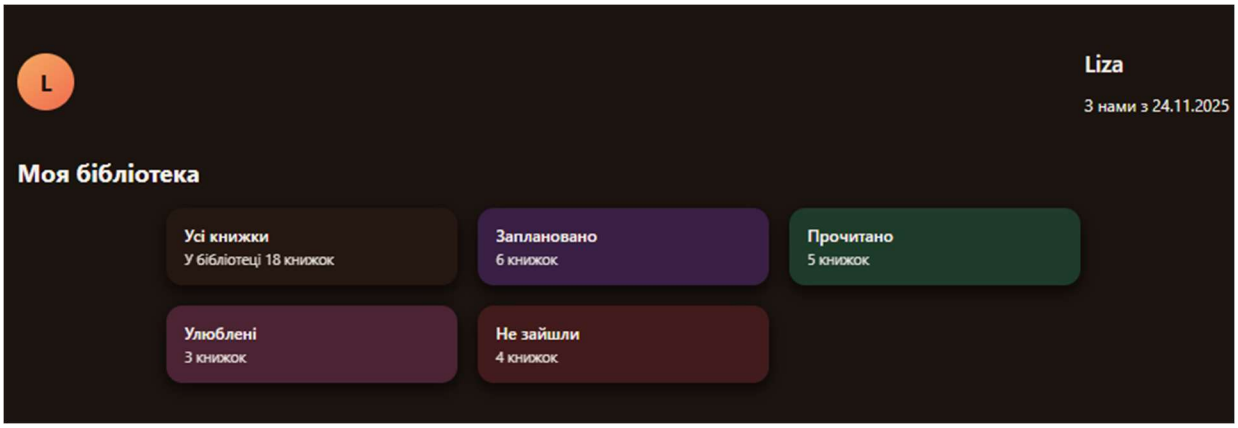


Рис. 3.9. Полички на персональному акаунті авторизованого користувача

(розроблено автором)

Окремим важливим елементом у вебдодатку є адміністративна панель [25]. Вона доступна лише користувачу з правами адміністратора. У цій панелі бібліотекар може переглядати статистику каталогу, додавати нові книги, редагувати наявні записи, видаляти книги, змінювати статус доступності та завантажувати обкладинки (див. рис. 3.10). Для зручності керування реалізовано пошук по книгах у межах адміністративної таблиці та пагінацію, яка виводить по п’ять книг на сторінку, представлено на рис. 3.11. Це робить роботу з великим списком книг більш структурованою. Також для адміністратора замість користувацьких дій на самій картці книжки передбачено можливість перейти до редагування книги або видалення її з каталогу.

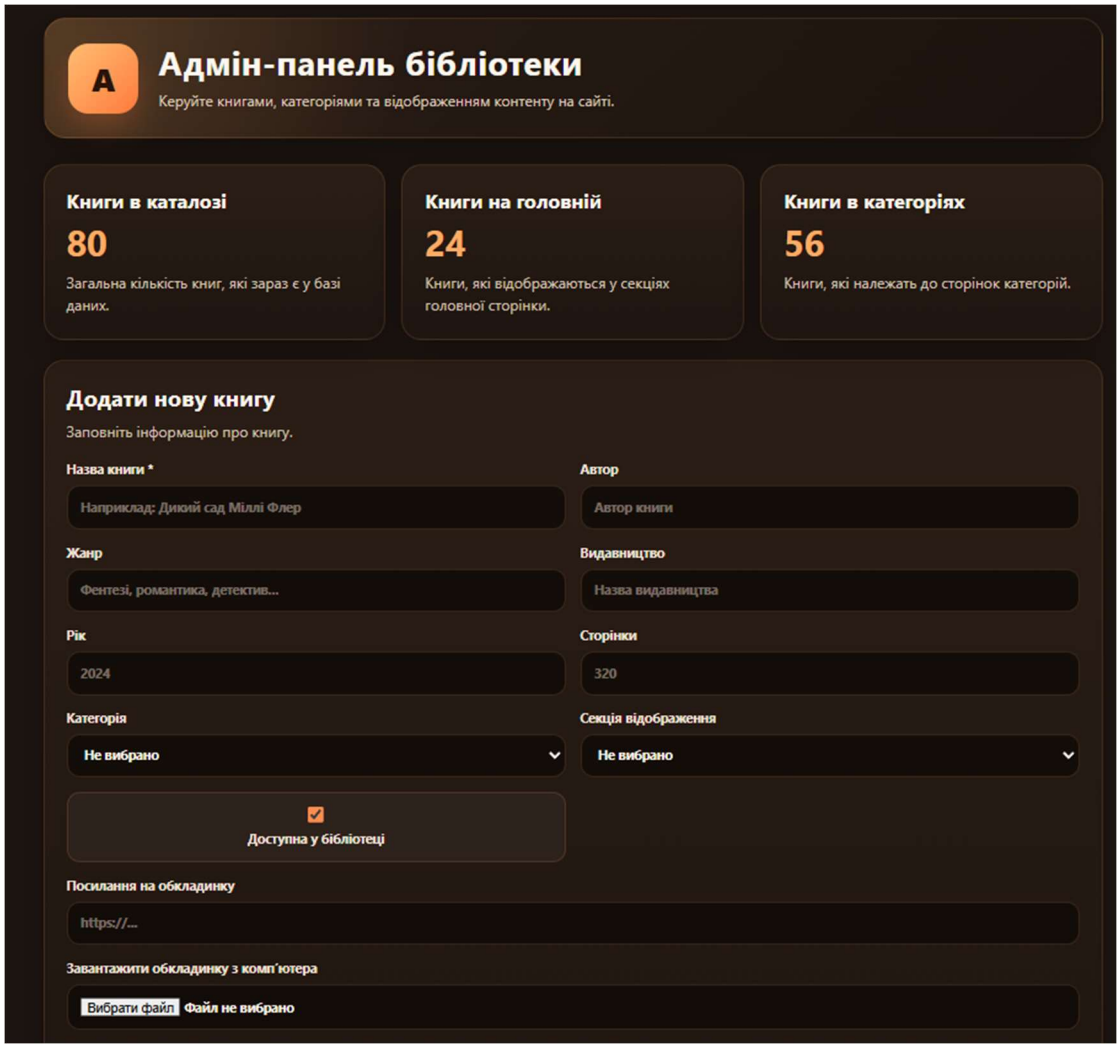


Рис. 3.10. Панель адміністратора вебдодатку MyLibr

(розроблено автором)

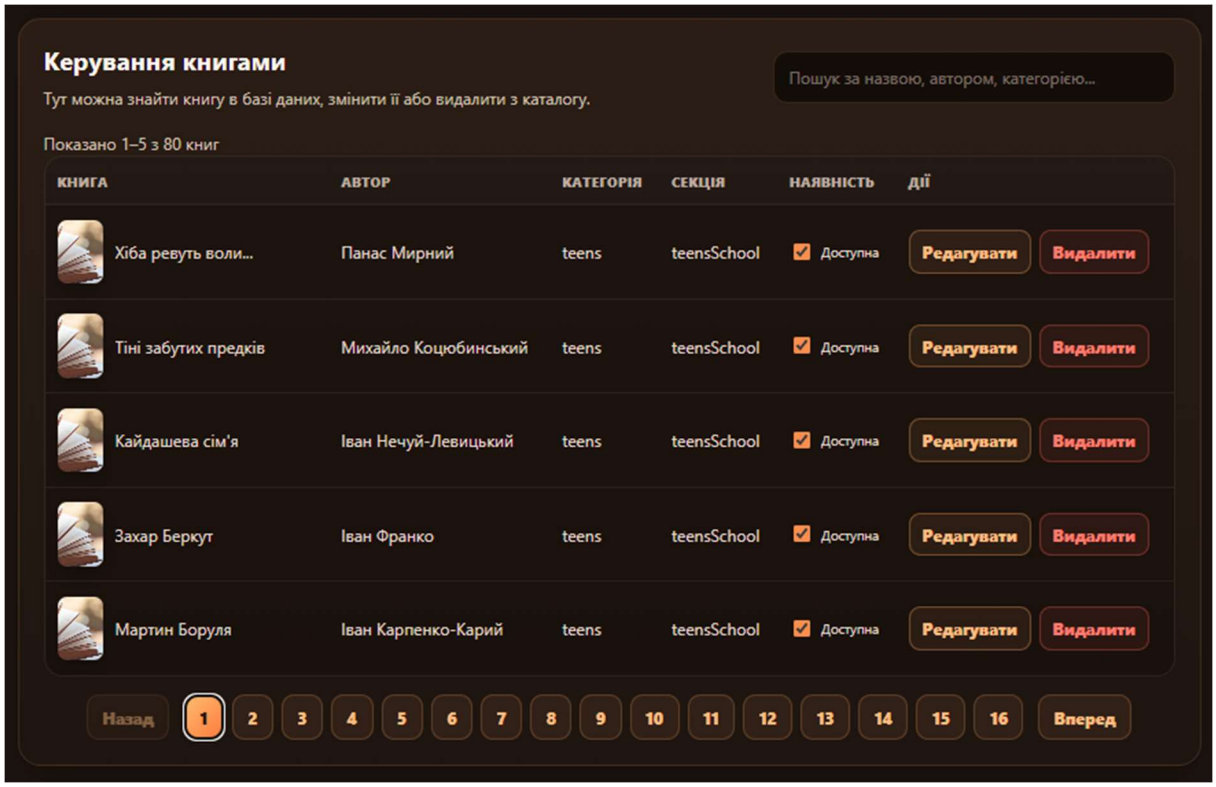


Рис. 3.11. Пошук по книгах у межах адміністративної таблиці

(розроблено автором)

У поточній версії база даних містить таблиці користувачів, книг і персональних бібліотек. Таблиця книг зберігає основну інформацію про книжковий фонд, таблиця користувачів – облікові записи, а таблиця персональної бібліотеки – зв’язок між користувачами, книгами та обраними полицями. На момент аналізу реалізована база містить 80 книжкових записів, що дозволяє перевірити роботу головної сторінки, категорій, пошуку, персональних полиць і адміністративної панелі на достатньому обсязі даних.

3.2 Сценарії взаємодії користувача з розробленим вебдодатком

Функціонування розробленого SPA доцільно розглядати через основні сценарії взаємодії користувачів із системою. У межах вебдодатку передбачено три основні типи взаємодії: робота гостя, робота авторизованого користувача та

робота адміністратора-бібліотекаря. Кожна з цих ролей має власний набір доступних дій, що забезпечує логічне розмежування функціоналу [26].

Перший сценарій – взаємодія гостя з вебдодатком. Гість може відкривати головну сторінку, переглядати добірки книг, переходити до категорій, знайомитися з новинами та переглядати сторінку видавництв. Також йому доступний пошук книг і перегляд детальної інформації про конкретне видання. Такий сценарій дозволяє використовувати сайт як відкритий електронний каталог без обов'язкової авторизації.

Під час перегляду головної сторінки гість бачить основні книжкові добірки, сформовані за секціями. Якщо його цікавить певна книга, він натискає на картку та переходить на сторінку детального перегляду. На цій сторінці відображається опис книги, її автор, жанр, видавництво, кількість сторінок, рік видання та доступність у бібліотеці. Проте дії з додавання книги до персональної бібліотеки для гостя недоступні, оскільки вони потребують авторизації.

Другий сценарій пов'язаний із пошуком книги. Користувач вводить назву, автора, жанр або інший ключовий запит у пошукове поле. Після цього вебдодаток переходить на сторінку результатів пошуку та відображає знайдені книги. Якщо збігів не знайдено, система повідомляє про це користувача і пропонує повернутися на головну сторінку. Такий сценарій є важливим, оскільки пошук є однією з базових функцій електронного каталогу [27].

Третій сценарій – перегляд книг за категоріями. Користувач відкриває навігаційне меню «Категорії» та обирає потрібний розділ: книги для дітей, підлітків або дорослих. Після цього система відображає сторінку з тематичними підрозділами. Наприклад, у дитячому розділі доступні навчальні книги та пригоди, у підлітковому – шкільна програма та жанрові добірки, а в дорослому – сучасна проза та нон-фікшн. Такий сценарій є зручним для користувачів, які ще не знають конкретної назви книги, але хочуть знайти літературу за віковою групою або тематикою.

Четвертий сценарій – реєстрація нового користувача. Користувач переходить на сторінку реєстрації, вводить логін і пароль, після чого система надсилає дані на сервер. Сервер перевіряє, чи не існує вже користувач із таким логіном, хешує пароль і створює новий запис у базі даних. Після успішної реєстрації користувач може авторизуватися та отримати доступ до персональних функцій.

П’ятий сценарій – авторизація користувача. Користувач вводить логін і пароль на сторінці входу. Сервер перевіряє дані, порівнює введений пароль із хешованим значенням у базі та у разі успішної перевірки повертає токен доступу. Після цього в інтерфейсі з’являється профіль користувача, а замість кнопки входу відображається ім’я користувача або позначення адміністратора. Авторизація є необхідною умовою для роботи з персональною бібліотекою [28].

Шостий сценарій – додавання книги до персональної бібліотеки. Авторизований користувач відкриває сторінку детального перегляду книги та натискає кнопку додавання. Після цього відкривається модальне вікно вибору полиці. Користувач може обрати одну з доступних категорій: заплановано, прочитано, улюблене або не сподобалось. Після вибору система зберігає зв’язок між користувачем, книгою та обраною полицею у базі даних. Інтерфейс оновлюється без перезавантаження сторінки, що відповідає принципам SPA.

Сьомий сценарій – перегляд персональної бібліотеки. Авторизований користувач переходить до власного профілю, де бачить блоки з полицями та кількістю книг у кожній із них. Відкривши конкретну полицю, користувач отримує список відповідних книг у вигляді карток. Якщо на полиці немає книг, система відображає повідомлення про порожній список. Це допомагає уникнути неінформативного порожнього екрана та робить інтерфейс більш зрозумілим.

Восьмий сценарій – видалення книги з персональної бібліотеки. Користувач може видалити книгу зі своєї бібліотеки через відповідну кнопку на картці або на сторінці книги. Перед виконанням дії система запитує

підтвердження, що зменшує ризик випадкового видалення. Після підтвердження запис видаляється з персональної бібліотеки користувача, а інтерфейс оновлюється відповідно до нового стану даних [29].

Дев’ятий сценарій – робота зі сторінкою видавництв. Користувач відкриває відповідний розділ у навігації та може обрати країну, місто або конкретне видавництво. Після вибору система відображає інформаційний блок із описом видавництва, прикладами відомих книг і картою розташування. Такий сценарій розширює можливості вебдодатку й робить його не лише каталогом книг, а й інформаційним ресурсом про книжкову сферу.

Десятий сценарій – взаємодія адміністратора з системою. Після авторизації під обліковим записом адміністратора користувач отримує доступ до адміністративної панелі. У ній відображається статистика каталогу, форма додавання або редагування книги та таблиця для керування записами. Адміністратор може створити нову книгу, вказавши її назву, автора, жанр, рік, видавництво, кількість сторінок, категорію, секцію, опис, обкладинку та доступність.

Одинадцятий сценарій – редагування книги адміністратором. Адміністратор знаходить потрібну книгу в таблиці, натискає кнопку редагування, після чого дані книги автоматично підставляються у форму. Після внесення змін адміністратор зберігає оновлений запис, а система відправляє запит на сервер і оновлює дані в базі. Після успішного виконання операції каталог оновлюється, і зміни стають доступними на всіх сторінках вебдодатку.

Дванадцятий сценарій – видалення книги адміністратором. Перед видаленням система запитує підтвердження дії. Якщо адміністратор підтверджує видалення, сервер видаляє книгу з бази даних, а також очищує пов’язані записи з персональних бібліотек користувачів. Такий підхід дозволяє підтримувати цілісність даних і не залишати в системі посилання на неіснуючу книгу [30].

Окремо варто відзначити сценарій зміни статусу доступності книги. Адміністратор може перемикати доступність книги без повного редагування

всього запису. Це зручно для ситуацій, коли книга тимчасово відсутня або знову доступна в бібліотеці. Статус одразу відображається на картці книги та на сторінці детального перегляду.

Таким чином, сценарії взаємодії охоплюють повний цикл роботи з вебдодатком: від відкритого перегляду каталогу до персоналізованого використання системи та адміністрування книжкового фонду. Завдяки розмежуванню ролей і динамічному оновленню інтерфейсу взаємодія з додатком є логічною, послідовною та зручною для різних категорій користувачів.

3.3 Аналіз ефективності розробленої системи та можливих обмежень

Графічний інтерфейс MyLibr побудований із використанням темного візуального оформлення, контрастного тексту, карткової структури та зрозумілої навігації. Це дозволяє зробити вебдодаток візуально цілісним і зручним для сприйняття [31]. Основні елементи інтерфейсу розташовані послідовно: у верхній частині знаходиться навігаційна панель, у центральній – основний контент сторінки, а на головній сторінці додатково використовується бічний блок новин. Така структура не перевантажує користувача та дозволяє швидко знайти потрібний розділ.

Ефективність розробленої системи насамперед проявляється у швидкості взаємодії користувача з інтерфейсом. Завдяки SPA-архітектурі переходи між сторінками відбуваються без повного перезавантаження вебдодатку. Це створює відчуття безперервної роботи та скорочує кількість зайвих дій. Користувач може швидко переходити від головної сторінки до книги, від книги до полиці, від пошуку до результатів або від профілю до персональної бібліотеки.

Ще однією перевагою є централізоване зберігання книжкового каталогу в базі даних SQLite. На відміну від локального зберігання даних у масивах або JSON-файлах, використання бази даних забезпечує більш впорядковану

структуру [32], можливість виконання SQL-запитів і збереження змін після перезапуску застосунку. Це особливо важливо для адміністративних функцій, оскільки додані, відредаговані або видалені книги зберігаються не лише в інтерфейсі, а й у базі даних.

Позитивною стороною є також реалізація персональної бібліотеки користувача. Книги не дублюються повністю для кожного користувача, а зберігаються через зв’язок між користувачем, книгою та полицею. Такий підхід є більш раціональним, оскільки основна інформація про книгу міститься в одному загальному каталозі, а персональна бібліотека лише фіксує належність книги до певного користувача та її статус [33].

Система авторизації підвищує практичну цінність вебдодатку, оскільки дозволяє відокремити гостьовий режим від персоналізованої роботи. Гість може переглядати відкриту інформацію, а авторизований користувач – формувати власну бібліотеку. Адміністратор, у свою чергу, має окремий набір прав для керування каталогом. Таке розмежування ролей забезпечує базовий рівень безпеки та логічну організацію доступу.

Адміністративна панель значно підвищує ефективність керування книжковим фондом. Замість ручного редагування файлів або бази даних адміністратор може працювати через графічний інтерфейс: додавати книги, змінювати інформацію, завантажувати обкладинки, перемикати доступність і видаляти записи. Наявність пошуку та пагінації в адміністративній таблиці спрощує роботу з каталогом, особливо коли кількість книг зростає.

Окремою перевагою системи є її модульність. Сторінки, компоненти, контексти та серверні маршрути розділено за функціональним призначенням [34]. Це полегшує підтримку проєкту, оскільки зміни в одній частині не потребують повного переписування всього додатку. Наприклад, сторінки категорій працюють із загальним контекстом книг, а не з окремими дубльованими масивами даних. Це зменшує ризик розбіжностей і спрощує оновлення каталогу.

Ефективність інтерфейсу також проявляється в обробці помилкових або порожніх станів. Якщо книги завантажуються, користувач бачить повідомлення про завантаження. Якщо виникає помилка, система виводить відповідне повідомлення. Якщо пошук не дав результатів або полиця порожня, користувач отримує зрозуміле пояснення. Такі повідомлення роблять систему більш дружньою та передбачуваною.

Разом із перевагами розроблена система має певні обмеження. По-перше, вебдодаток реалізований як навчальний проєкт і працює у локальному середовищі. Сервер запускається на локальному порту, а база даних SQLite зберігається у файловому форматі. Для повноцінного використання в реальній бібліотеці систему потрібно розгорнути на сервері, налаштувати хостинг, домен, захищене з'єднання HTTPS і стабільне резервне копіювання даних.

По-друге, SQLite добре підходить для невеликих і середніх проєктів, але при значному зростанні кількості користувачів і записів може виникнути потреба в переході на більш масштабовану систему керування базами даних, наприклад PostgreSQL або MySQL. Це дозволило б краще обробляти паралельні запити, розширити можливості аналітики та підвищити надійність при великому навантаженні.

По-третє, механізм авторизації реалізує базову логіку входу за токеном, однак у майбутньому його можна посилити. Доцільно додати повноцінні JWT-токени з терміном дії, оновлення сесій, відновлення пароля, підтвердження електронної пошти та додатковий захист адміністративних операцій. Це зробило б систему більш придатною для реального використання.

По-четверте, у поточній версії роль адміністратора визначається за логіном. У майбутньому доцільно додати окреме поле ролі в таблиці користувачів. Це дозволить створювати не лише адміністратора, а й кілька бібліотекарів із різними рівнями доступу [35].

По-п'яте, пошук реалізовано за основними текстовими полями, проте його можна розширити. У перспективі доцільно додати фільтрацію за роком, жанром, видавництвом, доступністю, віковою категорією та кількістю сторінок.

Також можна реалізувати сортування результатів за алфавітом, роком видання або популярністю.

По-шосте, система поки що не реалізує повний цикл бібліотечної видачі й повернення книги. У вебдодатку є статус доступності та персональні полиці користувача, але немає окремого модуля оформлення видачі книги на конкретний термін, контролю дати повернення та історії операцій. У майбутньому це можна розширити шляхом створення таблиці видач, у якій будуть зберігатися дата отримання, очікувана дата повернення, фактична дата повернення та статус операції.

Висновки до розділу 3

У ході реалізації та тестування розробленого SPA-вебдодатку було підтверджено працездатність основних функцій системи керування книжною бібліотекою. Створений вебдодаток MyLibr забезпечує перегляд книжкового каталогу, пошук літератури, роботу з віковими категоріями, перегляд детальної інформації про книги, авторизацію користувачів, формування персональної бібліотеки та адміністрування книжкового фонду.

Встановлено, що система має логічну структуру навігації, зручне карткове представлення книг, окремі сторінки для категорій, пошуку, видавництв, новин, персонального профілю та адміністративної панелі. Таке розділення функціональних блоків робить інтерфейс зрозумілим і зручним для користувачів.

Було проаналізовано сценарії взаємодії різних ролей із системою. Гість може переглядати каталог і шукати книги, авторизований користувач отримує доступ до персональної бібліотеки та книжкових полиць, а адміністратор має можливість додавати, редагувати, видаляти книги, змінювати їхню доступність і завантажувати обкладинки. Це підтверджує, що у вебдодатку реалізовано базове розмежування прав доступу відповідно до ролей користувачів.

Окрему увагу приділено оцінці ефективності системи. Використання SPA-архітектури забезпечує швидку навігацію без повного перезавантаження сторінок, а застосування SQLite дозволяє централізовано зберігати дані про книги, користувачів і персональні бібліотеки. При цьому, адміністративна панель спрощує керування книжковим фондом, а персональні полиці роблять систему більш корисною для читача.

Разом із тим було визначено можливі напрями подальшого розвитку. До них належать розгортання системи на сервері, посилення механізмів авторизації, розширення пошуку та фільтрації, додавання повноцінного модуля видачі й повернення книг, перенесення новин і видавництв до бази даних, а також удосконалення ролей користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено SPA-вебдодаток MyLibr для керування книжною бібліотекою. Створена система поєднує електронний каталог книг, персональну бібліотеку користувача та адміністративну панель для керування книжковим фондом.

У процесі дослідження було розглянуто сучасні підходи до електронного керування бібліотекою та визначено доцільність використання SPA-архітектури для такого типу систем. Було встановлено, що односторінковий вебдодаток дозволяє забезпечити швидку навігацію, динамічне оновлення інтерфейсу без повного перезавантаження сторінки та зручну взаємодію користувача з бібліотечними даними.

Для реалізації програмного продукту було обрано клієнт-серверну архітектуру. Клієнтську частину розроблено з використанням React, React Router, Context API, JavaScript, HTML, CSS та Vite. Серверну частину реалізовано на основі Node.js і Express.js. Для зберігання даних використано SQLite, що дозволило організувати централізоване збереження інформації про користувачів, книги та персональні бібліотеки.

У ході практичної реалізації створено основний функціонал системи: перегляд книжкового каталогу, пошук літератури, розподіл книг за категоріями, перегляд детальної інформації про книгу, реєстрацію та авторизацію користувачів, додавання книг до персональної бібліотеки, роботу з книжковими полицями, а також адміністрування книжкового фонду. Для адміністратора реалізовано можливість додавання, редагування, видалення книг, зміни їхньої доступності та завантаження обкладинок.

Розроблений вебдодаток має логічну структуру інтерфейсу. До його складу входять головна сторінка, сторінки категорій, сторінка пошуку, детальна сторінка книги, сторінка видавництв, блок новин, персональний кабінет користувача та адміністративна панель. Такий поділ забезпечує зручну навігацію та дає змогу користувачеві швидко знаходити потрібну інформацію.

Взаємодія між клієнтською і серверною частинами організована через REST API. Дані передаються у форматі JSON, а основні операції збереження, оновлення та видалення інформації виконуються через запити до бази даних. Це дозволило розмежувати логіку інтерфейсу, серверної обробки та зберігання даних.

У результаті виконання роботи було досягнуто поставленої мети – розроблено SPA-додаток для керування книжною бібліотекою, який забезпечує автоматизацію основних операцій із книжковим каталогом, персональними полицями користувача та адміністративним керуванням фондом. Розроблена система може бути використана як демонстраційний електронний каталог або основа для подальшого розвитку бібліотечної інформаційної системи.

Подальше вдосконалення вебдодатку може передбачати розгортання системи на сервері, посилення механізму авторизації, розширення фільтрації та пошуку, додавання повноцінного модуля видачі й повернення книг, а також перенесення новин і даних про видавництва до бази даних.

Отже, розроблений SPA-вебдодаток MyLibr відповідає темі кваліфікаційної роботи та демонструє практичне застосування сучасних вебтехнологій для автоматизації процесів керування книжною бібліотекою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гуралюк, А. Г., Углова, О. В. Цифрові тренди розвитку модерних бібліотек світу: майбутнє педагогічної освіти і науки (аналітичний огляд). Київ: Аналітичний вісник у сфері освіти й науки, 2024. 77 с.

2. Розман І. І., Петрова А. І. Сучасні тенденції розвитку інформаційних технологій бібліотечної та архівної справи. Рівне. 2023. №18. С. 129-136.

3. Бережна К. С. Особливості впровадження аібс «koha» в роботу бібліотеки харківського національного університету міського господарства імені о. м. Бекетові. Харків. Видавництво Харківський національний університет. 2017. 19 с.

4. Сюх О. М. Розробка клієнт-серверної системи онлайн-торгівлі з підходом односторінкової архітектури. Луцьк. Видавництво Волинський національний університет імені Лесі Українки. 2025. 65 с.

5. Спірін О. М., Олесюк О. Р. Аналіз програмних платформ для створення інституційних депозитаріїв. Київ. Видавництво Інститут інформаційних технологій і засобів навчання НАПН України. 2014. 101 с.

6. Речкалова Л. М. Цифрові технології в міжнародній бібліотечній співпраці україни: сучасні виклики та можливості. Київ. Видавництво Київський столичний університет імені Бориса Грінченка. 2025. 77 с.

7. Ширяєва, О. Л., Федорченко, М. С. Електронний каталог як інструмент популяризації ресурсів університетської бібліотеки. Харків. Видавництво Харківський національний медичний університет. 2021. 26 с.

8. Ковальчук А. М. та Пелихівський, Л. О. Актуальність та практичне застосування spa-додатків. Житомир. Видавництво Житомирський державний технологічний університет. 2015. 2 с.

9. Пасіка Д. Р. Розробка односторінкового веб-застосунку (SPA) з використанням Vue.js і React. Тернопіль. Видавництво ТНТУ. 2023. 120 с.

10. Безверхий О. С. та Куценко О. В. Ефективність застосування бібліотеки React. Київ. 2022. Інформаційні технології та суспільство. №2 (4) С. 13-19.

11. Данюк В. М. Про фреймворк vue.js. Житомир. Видавництво Житомирський державний технологічний університет. 2017. 38 с.

12. Мухін М. О. Розробка інформаційного веб-сайту криптовалют з використанням технологій ASP. NET Core, Angular та Entity. Одеса. Видавництво Одеський національний університет імені і. І. Мечникова. 2024. 64 с.

13. Сельська Н. П. Дослідження перспектив розробки SPA з використанням TypeScript на прикладі сайту для занять фітнесом з авторизацією користувачів. Тернопіль. Видавництво ТНТУ. 2023. 63 с.

14. Зінов'єва, О. Г. Використання CASE-засобів для проектування інформаційних систем. Запоріжжя. Видавництво Таврійський державний агротехнологічний університет імені Дмитра Моторного. 2023. 72 с.

15. Жежель, А. К. Розробка програмного забезпечення для реалізації методів та алгоритмів трансформації баз даних з ER-моделі в реляційну модель. Дніпро. Видавництво Національний технічний університет «Дніпровська політехніка». 2023.84 с.

16. Байраківська, В. В. Види та властивості алгоритмів. Вінниця. Видавництво Вісник студентського наукового товариства ДонНУ імені Василя Стуса. 2024. 175 с.

17. Сотник, О. А., Марченко, С. В., Єльсуков, А. Ю., Ігнатушко, С. О., & Качур, П. І. Frontend розробка веб-додатків на платформі redpitaya. Львів. Видавництво Національного університету «Львівська політехніка». 2025. 197 с.

18. Ступінь, А. П. Структура backend фрейморків. Луцьк. Видавництво Волинський національний університет імені Лесі Українки. 2023. 122 с.

19. Панасенко М. О. Медіапросування українських видань young-adult літератури. Запоріжжя. Видавництво Запорізького національного університету. 2021. 132 с.

20. Темченко, І. А. Інструменти тестування цифрової доступності сучасних бібліотечних платформ. Полтава. Видавництво Національного університету «Полтавська політехніка імені Юрія Кондратюка». 2025. 68 с.

21. Гуменюк, Д. А. Розроблення інтерактивної інформаційної системи для пошуку та продажу книг. Львів. Видавництво Національного університету «Львівська політехніка». 2025. 57 с.

22. Таланчук, О. Б. Образ бібліотеки в соціальних мережах. Хмельницьк. Видавництво Хмельницького національного університету. 2013. 41 с.

23. Меркулов, А. В. Програмна система для керування сервісами та взаємодії користувачів платформи замовлення послуг. Підсистема платежів та авторизації. Харків. Видавництво Харківського національного університету радіоелектроніки. 2025. 59 с.

24. Уткін, Є. І. Розробка застосунку «Персональна бібліотека» для каталогізації і управління прочитаними книгами. Харківського національного університету радіоелектроніки. 2024. 72 с.

25. Істомін А. О. Створення вебдодатку для компанії EcoBuild Systems з інтегрованою адміністративною панеллю. Харків. Видавництво Харківський національний університет міського господарства імені О. М. Бекетова. 2025. 93 с.

26. Музика, О. Ю. Організаційні схеми і психосоціальні умови ефективності соціального менеджменту. Тернопіль. Видавництво ЗУНУ. 2025. 121 с.

27. Горковчук, М. В. Структура та функції електронного каталогу мір якості геопросторових даних Запоріжжя. Видавництво Інженерна геодезія, 2014. 103 с.

28. Ляшук, Д. А. Персоналізація бібліотечних послуг: роль соціальних мереж, відеоконтенту та цифрових виставок. Київ. Видавництво Київського столичного університету імені Бориса Грінченка. 2025. 57 с.

29. Топчій, Д. Д. Програмна система для організації та управління груповими подорожами. Серверна частина. Харків. Видавництво Харківського національного університету радіоелектроніки. 2025. 49 с.

30. Волошин, О. В. Метод та веб-орієнтована інформаційна система бібліотеки. Хмельницьк. Видавництво Хмельницького національного університету. 2025. 76 с.

31. Пещерьова, В. С. Адаптивний інтерфейс навчальної платформи на основі технологій UX/UI. Миколаїв. Видавництво ЧНУ ім. Петра Могили. 2023. 67 с.

32. Демиденко, М. А. Введення в сучасні бази даних. Київ. Видавництво Інститут інформаційних технологій і засобів навчання НАПН України. 2020. 141 с.

33. Корнійчук, М. В. Пам'ятки про каталоги. Острог. Видавництво Національного університету" Острозька академія". 2011. 129 с.

34. Меженна, Н., Русевич, Т., & Житник, О. Модульність як основа архітектури spa-комплексу. Київ. Видавництво Архітектурний вісник КНУБА, 249 с.

35. Храпчун, Р. С. Інформаційна система контролю бібліотеки та її вмісту. Львів. Вммдавнцтво Національний університет "Львівська політехніка". 2024. 84 с.

36. Дубик, С. О., Онисько, Г. Я., & Шкодзінською, О. К. АБІС Koha як основа для формування віртуального бібліотечного простору. Тернопіль. Видавництво Тернопільського університету імені Івана Пулюя. 2015. 52 с.

37. Лупаренко, Л. С. Еволюція відкритих електронних науково-освітніх систем і їх використання у вітчизняному освітньому просторі. Хмульницьк. Видавництво Національної академії Державної прикордонної служби України. 2021. 272 с.

38. Мінтій, І. Р., Вакалюк, Т. О. Аналіз інституційних репозитаріїв та бібліотечних систем для зберігання й опрацювання fair-даних у галузі освітніх наук. Харків. Видавництво ХНПУ імені Г.С. Сковороди. 2026. 70 с.

39. Воронцова, М. А. Застосування технологій штучного інтелекту у бібліотечних послугах. Київ. Видавництво Київського університету ім. Бориса Грінченка. 2025. 39 с.

40. Кундос, М. М., Герасимчук, І. К., Порівняльний аналіз багатосторінкової та одностроінкової архітектури для освітніх web-додатків: обґрунтування вибору MPA. Рівне. Видавництво Міжнародного економіко-гуманітарного університету імені академіка Степана Дем'янчука. 2025. 146 с.

41. Бондарчук В. О. Проєктування та розробка інтерфесу користувача LMS системи із використанням фреймворку next.js. Луцьк. Видавництво Волинського національного університету ім. Лесі Українки. 2025. 63 с.

42. Карпюк, І. В. Інформаційна система для автоматизованого розгортання веб-додатків з використанням CI/CD процесів. Львів. Видавництво Львівська політехніка. 2024. 57 с.

43. Лемак, В. В. Інтелектуальна система керування завданнями. Суми. Видавництво Сумського державного університету. 2025. 87 с.

Додаток А

Код файлу AuthContext.jsx

```
import { createContext, useContext, useState, useEffect, useMemo }
from "react";
const AuthContext = createContext(null);
const API_URL = "http://localhost:5000";
const initialLibrary = {
  all: [],
  planned: [],
  finished: [],
  disliked: [],
  favorite: [],
};
function normalizeLibrary(raw) {
  if (!raw || typeof raw !== "object") return initialLibrary;
  const normalizeArr = (arr) =>
    Array.isArray(arr) ? arr.map((x) => String(x)) : [];
  return {
    all: normalizeArr(raw.all),
    planned: normalizeArr(raw.planned),
    finished: normalizeArr(raw.finished),
    disliked: normalizeArr(raw.disliked),
    favorite: normalizeArr(raw.favorite),
  };
}
function libraryRowsToState(rows) {
  const result = {
    all: [],
    planned: [],
    finished: [],
    disliked: [],
    favorite: [],
  };
  if (!Array.isArray(rows)) return result;
  rows.forEach((item) => {
    if (!item.book || !item.book.id) return;
    const id = String(item.book.id);
    if (!result.all.includes(id)) {
      result.all.push(id);
    }
     if (item.shelf && result[item.shelf] && item.shelf !== "all")
{
      result[item.shelf].push(id);
    }
  });
  return normalizeLibrary(result);
}
export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
```

Продовження додатку А

```
      const     [token,     setToken]     =     useState(()     =>
localStorage.getItem("authToken"));
  const [library, setLibrary] = useState(initialLibrary);
  const [loading, setLoading] = useState(true);
  useEffect(() => {
    async function fetchProfile() {
      if (!token) {
        setUser(null);
        setLoading(false);
        return;
      }
      try {
        const res = await fetch(`${API_URL}/api/profile`, {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });
        if (!res.ok) {
          localStorage.removeItem("authToken");
          setToken(null);
          setUser(null);
        } else {
          const data = await res.json();
          setUser(data);
        }
      } catch (err) {
        console.error("Profile fetch error:", err);
        localStorage.removeItem("authToken");
        setToken(null);
        setUser(null);
      } finally {
        setLoading(false);
      }
    }
    fetchProfile();
  }, [token]);
  useEffect(() => {
    async function fetchLibrary() {
      if (!user || !token) {
        setLibrary(initialLibrary);
        return;
      }
      try {
        const res = await fetch(`${API_URL}/api/library`, {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });
        if (!res.ok) {
          setLibrary(initialLibrary);
```

Продовження додатку А

```
          return;
        }

        const data = await res.json();
        setLibrary(libraryRowsToState(data));
      } catch (err) {
        console.error("Library fetch error:", err);
        setLibrary(initialLibrary);
      }
    }
    fetchLibrary();
  }, [user, token]);
  async function login(loginValue, password) {
    const res = await fetch(`${API_URL}/api/login`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ login: loginValue, password }),
    });
    const data = await res.json().catch(() => ({}));
    if (!res.ok) {
      throw new Error(data.message || "Помилка входу");
    }
    setUser(data.user);
    setToken(data.token);
    localStorage.setItem("authToken", data.token);
  }
  async function register(loginValue, password) {
    const res = await fetch(`${API_URL}/api/register`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ login: loginValue, password }),
    });
    const data = await res.json().catch(() => ({}));
    if (!res.ok) {
      throw new Error(data.message || "Помилка реєстрації");
    }
    await login(loginValue, password);
  }
  function logout() {
    setUser(null);
    setToken(null);
    localStorage.removeItem("authToken");
    setLibrary(initialLibrary);
  }
  async function addBookToShelf(bookId, shelfId, bookData = null)
{
    if (!bookId || !shelfId || !token) return;
    const id = String(bookId);
    const book = bookData || { id };
    setLibrary((prev) => {
      const inAll = prev.all.includes(id);
```

```
    const newAll = inAll ? prev.all : [...prev.all, id];
    const cleaned = {
      ...prev,
      all: newAll,
      planned: prev.planned.filter((x) => x !== id),
      finished: prev.finished.filter((x) => x !== id),
      disliked: prev.disliked.filter((x) => x !== id),
      favorite: prev.favorite.filter((x) => x !== id),
    };
    if (shelfId !== "all" && cleaned[shelfId]) {
      cleaned[shelfId] = [...cleaned[shelfId], id];
    }
    return cleaned;
  });
  try {
    const res = await fetch(`${API_URL}/api/library`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify({
        book,
        shelf: shelfId,
      }),
    });
    if (!res.ok) {
      console.error("Cannot save book to database");
    }
  } catch (err) {
    console.error("Library save error:", err);
  }
}
async function removeBookFromLibrary(bookId) {
  if (!bookId || !token) return;
  const id = String(bookId);
  setLibrary((prev) => ({
    all: prev.all.filter((x) => x !== id),
    planned: prev.planned.filter((x) => x !== id),
    finished: prev.finished.filter((x) => x !== id),
    disliked: prev.disliked.filter((x) => x !== id),
    favorite: prev.favorite.filter((x) => x !== id),
  }));
  try {
    const res = await fetch(`${API_URL}/api/library/${id}`, {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    if (!res.ok) {
```

Продовження додатку А

```
        console.error("Cannot remove book from database");
      }
    } catch (err) {
      console.error("Library remove error:", err);
    }
  }
  const isAuthenticated = !!user;
  const isAdmin = user?.login === "admin";
  const value = useMemo(
    () => ({
      user,
      token,
      loading,
      isAuthenticated,
      isAdmin,
      login,
      register,
      logout,
      library,
      setLibrary,
      addBookToShelf,
      removeBookFromLibrary,
    }),
    [user, token, loading, isAuthenticated, isAdmin, library]
  );
                    return                    <AuthContext.Provider
value={value}>{children}</AuthContext.Provider>;
}
export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) {
    throw new Error("useAuth must be used within AuthProvider");
  }
  return ctx;
}
```

Додаток Б

Код файлу BooksContext.jsx

```
import { createContext, useContext, useEffect, useMemo, useState }
from "react";
import { useAuth } from "./AuthContext";
const BooksContext = createContext(null);
const API_URL = "http://localhost:5000";
export function BooksProvider({ children }) {
  const { token } = useAuth();
  const [books, setBooks] = useState([]);
  const [booksLoading, setBooksLoading] = useState(true);
  const [booksError, setBooksError] = useState("");
  function getAdminToken() {
    return token || localStorage.getItem("authToken");
  }
  async function refreshBooks() {
    try {
      setBooksLoading(true);
      setBooksError("");
      const res = await fetch(`${API_URL}/api/books`);
      if (!res.ok) {
        throw new Error("Не вдалося завантажити книги");
      }
      const data = await res.json();
      if (Array.isArray(data)) {
        setBooks(data);
      } else {
        setBooks([]);
      }
    } catch (err) {
      console.error("Books fetch error:", err);
      setBooksError("Помилка завантаження книг");
      setBooks([]);
    } finally {
      setBooksLoading(false);
    }
  }
  useEffect(() => {
    refreshBooks();
  }, []);
  function getBookById(id) {
    return books.find((book) => String(book.id) === String(id)) ||
null;
  }
  async function deleteBookById(id) {
    const adminToken = getAdminToken();
    if (!id) {
      throw new Error("Не знайдено ID книги для видалення");
    }
```

Продовження додатку Б

```
    if (!adminToken) {
      throw new Error("Немає доступу для видалення книги");
    }
    const res = await fetch(`${API_URL}/api/admin/books/${id}`, {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${adminToken}`,
      },
    });
    const data = await res.json().catch(() => ({}));
    if (!res.ok) {
        throw new Error(data.message || "Не вдалося видалити
книгу");
    }
     setBooks((prev) => prev.filter((book) => String(book.id) !==
String(id)));
    return data;
  }
  async function updateBookById(id, bookData) {
    const adminToken = getAdminToken();
    if (!id) {
      throw new Error("Не знайдено ID книги для редагування");
    }
    if (!adminToken) {
      throw new Error("Немає доступу для редагування книги");
    }
    const res = await fetch(`${API_URL}/api/admin/books/${id}`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${adminToken}`,
      },
      body: JSON.stringify(bookData),
    });
    const data = await res.json().catch(() => ({}));
    if (!res.ok) {
      throw new Error(data.message || "Не вдалося оновити книгу");
    }
    await refreshBooks();
    return data;
  }
  async function createBook(bookData) {
    const adminToken = getAdminToken();

    if (!adminToken) {
      throw new Error("Немає доступу для додавання книги");
    }
    const res = await fetch(`${API_URL}/api/admin/books`, {
      method: "POST",
      headers: {
```

Продовження додатку Б

```
        "Content-Type": "application/json",
        Authorization: `Bearer ${adminToken}`,
      },
      body: JSON.stringify(bookData),
    });
    const data = await res.json().catch(() => ({}));
    if (!res.ok) {
      throw new Error(data.message || "Не вдалося додати книгу");
    }
    await refreshBooks();
    return data;
  }
  const value = useMemo(
    () => ({
      books,
      booksLoading,
      booksError,
      getBookById,
      refreshBooks,
      deleteBookById,
      updateBookById,
      createBook,
    }),
    [books, booksLoading, booksError, token]
  );
  return (
                                                <BooksContext.Provider
value={value}>{children}</BooksContext.Provider>
  );
}
export function useBooks() {
  const ctx = useContext(BooksContext);
  if (!ctx) {
    throw new Error("useBooks must be used within BooksProvider");
  }
  return ctx;
}
```

ЗГОДА здобувачки вищої освіти

Державного університету економіки і технологій про
перевірку кваліфікаційної роботи на прояви
академічного плагіату
та розміщення в Репозитарії Університету

Я, **Білая Єлизавета Святославівна**,

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

«Розробка SPA для керування книжною бібліотекою»

виконана самостійно та не містить академічного плагіату. Я не надавала і не одержувала недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлена. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформована, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлена.

15.06.26 р  Білая Є. С.