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

Маловічко Олександра Вікторовича
(прізвище, ім'я, по батькові здобувача)

на тему Розробка SPA PWA сервісу електронної бібліотеки
(повна назва теми)

за матеріалами _____
(повна назва бази дослідження)

науковий керівник _____ _____ Шокотько Л.М.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 09.06.2025р. № 12
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг – 2025

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 _Комп'ютерні науки___

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"07" квітня 2025 року

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

_____Маловічко Олександру Вікторовичу_____

1. Тема роботи _Розробка SPA PWA сервісу електронної бібліотеки_____

науковий керівник роботи ___Шокотько Людмила Миколаївна _____,
затверджені наказом вищого навчального закладу від «04» квітня 2025 р. № 224-ст

2. Строк подання здобувачем роботи 31.05.2025р.

3. Зміст кваліфікаційної роботи бакалавра, об'єкт, предмет та мета дослідження:

Розділ 1 __Аналіз предметної області та постановка завдання_____
_____
Розділ 2 __Моделювання процесів управління особистою бібліотекою _____
_____
Розділ 3 __Програмна реалізація проєкту_____
_____

Об'єкт дослідження − технологічні рішення для створення SPA та PWA (Progressive Web App) та Inertia.js, їх поєднання та імплементація_____

Предмет дослідження − технологічні підходи до створення SPA / PWA-застосунків на базі Laravel-бекенду й Vue-фронтенду_____

Мета кваліфікаційної роботи бакалавра − створення SPA (Single Page Application) сервісу для управління електронною бібліотекою книг_____

4. Дата видачі завдання 04.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 28.04.2025р.	25.04.2025
2	Підготовка розділу 2	до 16.05.2025р.	15.05.2025
3	Підготовка розділу 3	до 30.05.2025р.	29.05.2025
4	Реєстрація завершеної дипломної роботи	до 31.05.2025р.	30.05.2025
5	Отримання відгуку від наукового керівника	03-04.06.2025р.	04.06.2025
6	Отримання зовнішньої рецензії	05-06.06.2025р.	06.06.2025
7	Перевірка кваліфікаційної роботи на плагіат	02-09.06.2025р.	04.06.2025
8	Попередній захист кваліфікаційної роботи на кафедрі	03.06.2025р.	03.06.2025
9	Допуск кафедрою кваліфікаційної роботи до захисту	09.06.2025р.	09.06.2025
10	Підготовка студента до захисту в ЕК	до 17.06.2025р.	17.06.2025

Завдання підготував науковий керівник Шокотько Л.М.

Завдання одержав здобувач _____ Маловічко О. В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 54 стор., 23 рисунки, 3 додатка, 28 використаних джерел.

Метою роботи є створення SPA (Single Page Application) сервісу для управління бібліотекою книг.

Об’єкт дослідження – сучасні технологічні рішення для створення SPA та PWA (Progressive Web App) та Inertia.js, їх поєднання та імплементація.

В ході виконання роботи було виконано: визначення моделі процесу розробки програмного забезпечення, аналіз вимог, проєктування інтерфейсу користувача, проєктування структури програмного забезпечення, розробка веб-застосунку.

Під час розробки веб-застосунку було використано такі інструментальні засоби, як Laravel Nova, MySQL, Vue.js та Inertia.js. Ці інструменти забезпечують гнучкість у розробці, надійну серверну логіку та швидкий фронтенд. TailwindCSS дозволяє створити естетичний і простий у використанні інтерфейс, тоді як Redis підвищує продуктивність системи завдяки кешуванню.

Програмний комплекс спрямований на полегшення та покращення процесу використання, сортування і читання електронних книг з власної бібліотеки.

Експлуатаційне призначення програмного комплексу полягає в використанні його в персональних цілях.

Ключові слова: бібліотека, клієнт, облік, сайт, PHP, PWA, SPA, UML, VUE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SPA – Single Page Application

PWA – Progressive Web App

JS – Java Script

HTML – HyperText Markup Language

PHP – Hypertext Preprocessor (Personal Home Page Tools)

DRM – Digital Rights Management

UML – Unified Modeling Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

API – Application Programming Interface

ВСТУП

Сучасний світ вимагає доступних і зручних рішень для управління особистою інформацією. Однією з важливих потреб є можливість ефективного керування бібліотекою книг, яка залишається доступною в будь-який час та з будь-якого пристрою. Актуальність створення веб-застосунку для особистої бібліотеки полягає в інтеграції сучасних технологій, які забезпечують гнучкість, продуктивність і зручність у використанні.

У цифрову епоху обсяг особистих електронних бібліотек зростає швидше, ніж удосконалюються інструменти для їх упорядкування. Більшість популярних сервісів тяжіють до закритих екосистем. Amazon Kindle нав'язує власний формат AZW і DRM-захист, Apple Books обмежений пристроями iOS/macOS. У результаті користувач змушений балансувати між кількома програмами й втрачати час на конвертацію файлів або синхронізацію між пристроями.

Одночасно браузер став повноцінною платформою для складних застосунків: поява Service Worker, IndexedDB і стандартів офлайн-кешування дала змогу прогресивним веб-застосункам (PWA) поєднати переваги нативного програмного забезпечення зі зручністю миттєвого розгортання. Архітектура SPA (Single Page Application) забезпечує плавну навігацію, а сучасний стек Laravel + Vue.js + Inertia.js + Vite спрощує розробку кросплатформних рішень і гарантує швидкий відгук інтерфейсу.

Актуальність теми обумовлена необхідністю універсального інструменту, що дозволяє працювати з особистою бібліотекою незалежно від формату файлів, типу пристрою чи наявності мережі.

Метою цієї роботи є розробка універсального веб-застосунку, що надає користувачу можливість організувати, впорядковувати та взаємодіяти зі своєю бібліотекою книг. Цей застосунок має функціонувати як в режимі онлайн, так і офлайн, використовуючи технології прогресивних веб-застосунків (PWA).

Проєкт побудований на основі технологій Laravel, Laravel Nova, MySQL, Vue.js та Inertia.js. Ці інструменти забезпечують гнучкість у розробці, надійну

серверну логіку та швидкий фронтенд. TailwindCSS дозволяє створити естетичний і простий у використанні інтерфейс, тоді як Redis підвищує продуктивність системи завдяки кешуванню. Інтеграція Vite забезпечує швидкий цикл розробки та оптимізує продуктивність програми, а PWA гарантує доступність даних навіть без підключення до інтернету.

Використання архітектури SPA (Single Page Application) стало ключовим елементом розробки даного застосунку. SPA дозволяє завантажувати сторінку лише один раз, після чого необхідні дані довантажуються динамічно, що значно підвищує швидкість роботи та покращує користувацький досвід. Такий підхід також дозволяє забезпечити плавну навігацію без необхідності перезавантаження сторінок, що є важливим для сучасних веб-застосунків.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі рішення керування електронними бібліотеками та визначити їхні обмеження;
- обґрунтувати вибір технологічного стеку Laravel, Vue.js, Inertia.js, Vite, TailwindCSS і Redis;
- спроєктувати логічну та фізичну моделі даних і структуру API;
- реалізувати модулі аутентифікації, імпорту/експорту книг, пошуку та тегування;
- забезпечити підтримку форматів електронних книжок;
- упровадити сервіс-воркер і кешування для повноцінної офлайн-роботи;
- оптимізувати продуктивність шляхом використання Redis-кешу, lazy-loading і асинхронних черг;
- провести юзабіліті-та стрес-тестування на різних пристроях та мережах;

Об’єктом дослідження є сучасні технологічні підходи до створення SPA / PWA-застосунків на базі Laravel-бекенду й Vue-фронтенду.

Предметом дослідження є методологія розроблення веб-системи для керування особистою бібліотекою, що забезпечує мінімальний час відгуку, офлайн-доступ і кросплатформну роботу.

Результатом кваліфікаційної роботи є повноцінний веб-застосунок-PWA «Особиста бібліотека», реалізований на стеку Laravel + Vue.js + Inertia.js + Vite з TailwindCSS та Redis-кешем. Таким чином, розроблений застосунок буде надавати користувачам можливість зручно додавати, редагувати й переглядати свої книги, створюючи гнучкий інструмент для роботи з особистою бібліотекою. Особливий акцент зроблено на адаптивності та доступності, що дозволить користувачам взаємодіяти із застосунком з будь-якого пристрою та в будь-яких умовах особливо за допомогою офлайн-режиму через Service Worker.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальний огляд управління файлами в умовах цифровізації

Сучасний світ характеризується стрімким розвитком цифрових технологій, які радикально змінюють підходи до обробки, зберігання та передачі інформації. Управління файлами стало важливим компонентом будь-якої інформаційної системи, оскільки від ефективності цього процесу залежить швидкість доступу до даних, безпека та організація роботи.

Управління файлами охоплює широкий спектр завдань: від створення, редагування, сортування до зберігання, архівації та спільного використання файлів. У контексті цифровізації, ця діяльність набуває ще більшого значення, адже кількість інформації, яку необхідно обробляти, постійно зростає. Це вимагає розробки інноваційних підходів та інструментів, які дозволяють ефективно керувати файлами, забезпечуючи водночас високу швидкість та безпеку операцій.

Одним із ключових трендів в управлінні файлами є автоматизація. Завдяки використанню сучасних програмних рішень, компанії та окремі користувачі отримують можливість значно спростити виконання рутинних завдань, таких як систематизація файлів або їх резервне копіювання. Впровадження автоматизованих систем також дозволяє уникнути типових помилок, що виникають через людський фактор, наприклад, втрати важливих даних чи дублювання файлів.

Ще однією важливою тенденцією є хмарні технології. Хмарні сервіси для управління файлами забезпечують доступ до даних з будь-якого пристрою та місця, що особливо актуально для бізнесу, який працює в умовах децентралізованих команд. Крім того, хмарні рішення дозволяють оптимізувати витрати на IT-інфраструктуру, адже користувачі оплачують лише фактично використані ресурси.

Інтеграція інтелектуальних алгоритмів також займає важливе місце в сучасних системах управління файлами. Використання штучного інтелекту дозволяє автоматично класифікувати файли, знаходити дублікати та пропонувати найбільш оптимальні сценарії для організації даних. Завдяки цьому користувачі можуть зекономити час та підвищити продуктивність роботи.

Безпека даних є ще одним критичним аспектом управління файлами. У зв'язку зі збільшенням кількості кібератак, розробка надійних механізмів захисту інформації стала ключовим завданням для розробників програмного забезпечення. Високоякісні системи управління файлами повинні забезпечувати багаторівневий захист, включаючи шифрування даних, автентифікацію користувачів та контроль доступу.

Окремо слід звернути увагу на управління файлами придбаних цифрових книжок. Наразі багато користувачів зіштовхуються з труднощами під час організації своїх бібліотек електронних книг. Відсутність єдиних стандартів зберігання, множинність форматів файлів (ePub, fb2, PDF, mobi тощо) та необхідність використання різних програм для читання створюють незручності. Крім того, користувачам часто доводиться самостійно сортувати книги за жанрами, авторами або іншими критеріями, що займає багато часу. Відсутність інтегрованих рішень для автоматичної організації та синхронізації між пристроями також створює бар'єри для зручного використання таких файлів.

Загалом, розвиток управління файлами в умовах цифровізації вимагає комплексного підходу, який враховує як технічні, так і організаційні аспекти. Впровадження сучасних інструментів дозволяє значно підвищити ефективність роботи з інформацією, забезпечуючи зручний доступ до даних, їх захист та можливість для інтеграції з іншими системами.

1.2 Аналіз існуючих застосунків для читання електронних книжок

Світові лідери ринку електронного читання сформували різноформатне середовище, у якому кожен сервіс намагається поєднати захист авторських прав,

зручність користувача та гнучкість екосистеми. Найпоширенішим прикладом є Amazon Kindle (рис. 1.1), що пропонує тісну інтеграцію власного магазину з хмарним сховищем та сімейством читалок. Стратегія «закритого циклу» забезпечує безшовну синхронізацію й стабільну роботу, але водночас обмежує користувача власним форматом AZW та жорсткою DRM-політикою, унеможливлюючи простий імпорт придбаних у сторонніх магазинах файлів без додаткових конверсій чи втрати захисту.

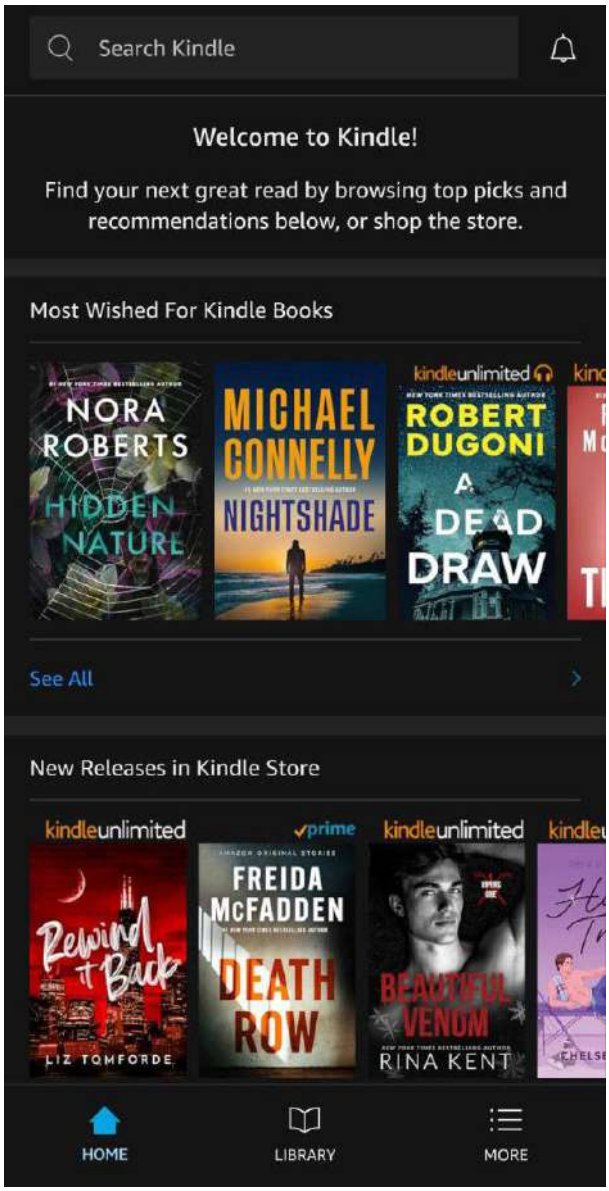

Рис. 1.1. Скріншот застосунку Amazon Kindle

Примітка. Джерело: Розроблено із використанням [24]

Apple Books розвиває подібний підхід усередині екосистеми iOS / macOS. Програма підтримує відкриті формати EPUB і PDF, пропонує відстеження прогресу читання, примітки й синхронізацію через iCloud, проте доступна лише на пристроях Apple. Відсутність веб-версії й сувора прив’язка до одного бренду знижують кросплатформність, що є критичним чинником для користувачів, котрі користуються різними операційними системами.

Google Play Books (рис. 1.2) використовує модель «магазин + читалка + хмара», але робить акцент на підтримці власного магазину книжок. Основним недоліком залишається повільна робота офлайн та обмежений набір інструментів впорядкування, що ускладнює роботу з великими приватними бібліотеками та не зручність читання і налаштування режиму відображення книжки.

Рис. 1.2. Скріншот застосунку Google Play Books

Примітка. Джерело: Розроблено із використанням [25]

Kobo Books (рис. 1.3) позиціонується як відкрита альтернатива Kindle. Підтримка EPUB, інтеграція з сервісами Pocket та OverDrive, а також розвинена мережа партнерських книжкових магазинів надають користувачеві свободу вибору джерел контенту. Водночас десктопна версія має обмежений функціонал, а мобільні застосунки поступаються за швидкодією конкуруючим рішенням, особливо під час відкриття великих ілюстрованих видань.

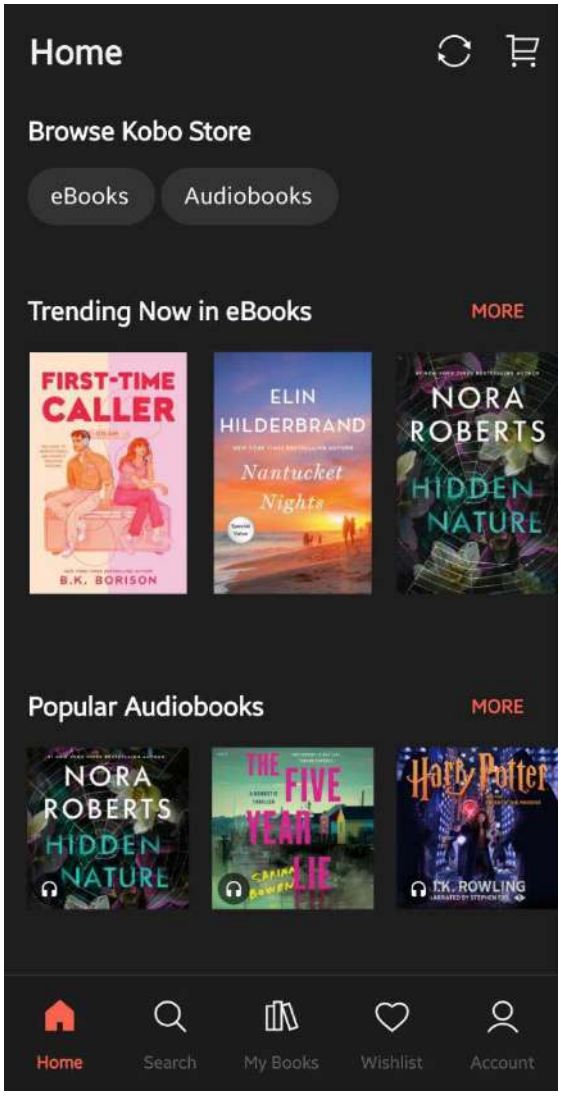

Рис. 1.3. Скріншот застосунку Kobo Books

Примітка. Джерело: Розроблено із використанням [26]

PocketBook Reader, (рис. 1.4) популярний на пострадянському просторі, вирізняється широкою підтримкою локалізацій, великою кількістю форматів. Проте відсутність вбудованого магазину змушує користувачів шукати книги

сторонніми шляхами, а синхронізація між платформами реалізована частково й часто залежить від стороннього хмарного сховища.

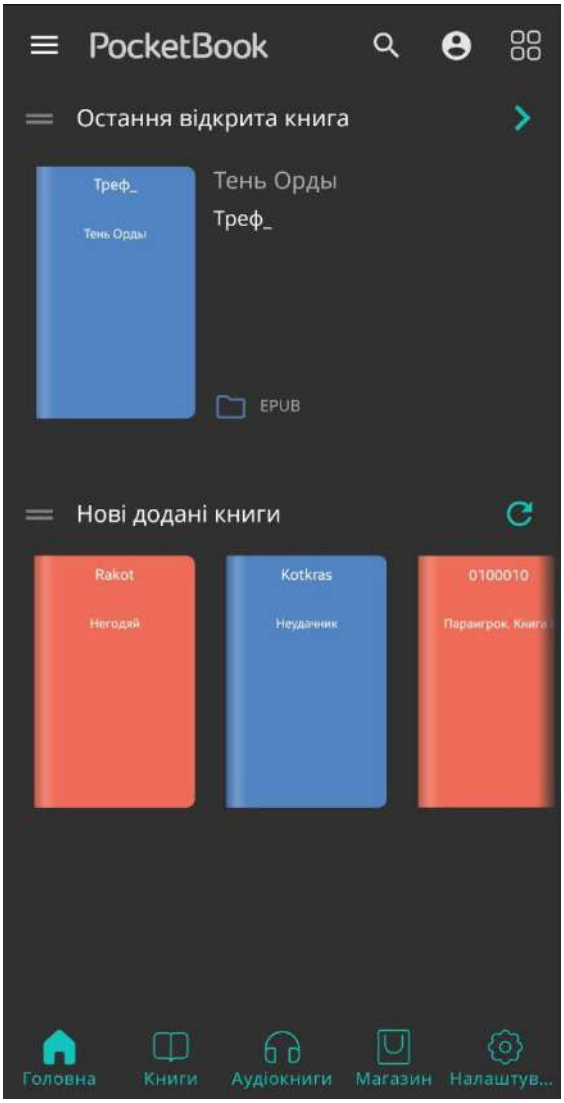

Рис. 1.4. Скріншот застосунку PocketBook Reader

Примітка. Джерело: Розроблено із використанням [27]

FBReader та ReadEra (рис. 1.5) демонструють підхід «легкої» офлайн-читалки. Вони відкривають майже всі поширені формати, займають мінімум ресурсів і не вимагають реєстрації. Головною слабкою ланкою є відсутність централізованого каталогу й сервісів резервного копіювання: бібліотека користувача зберігається локально, тож у разі втрати пристрою контент і метадані зникають.

Calibre виділяється серед настільних застосунків усеосяжним набором

інструментів: організацією бібліотеки, масовою конвертацією, редагуванням метаданих і можливістю налаштовувати власний сервер доступу. Проте інтерфейс програми складний для пересічного читача, а процес налаштування віддаленого доступу вимагає технічної компетентності.

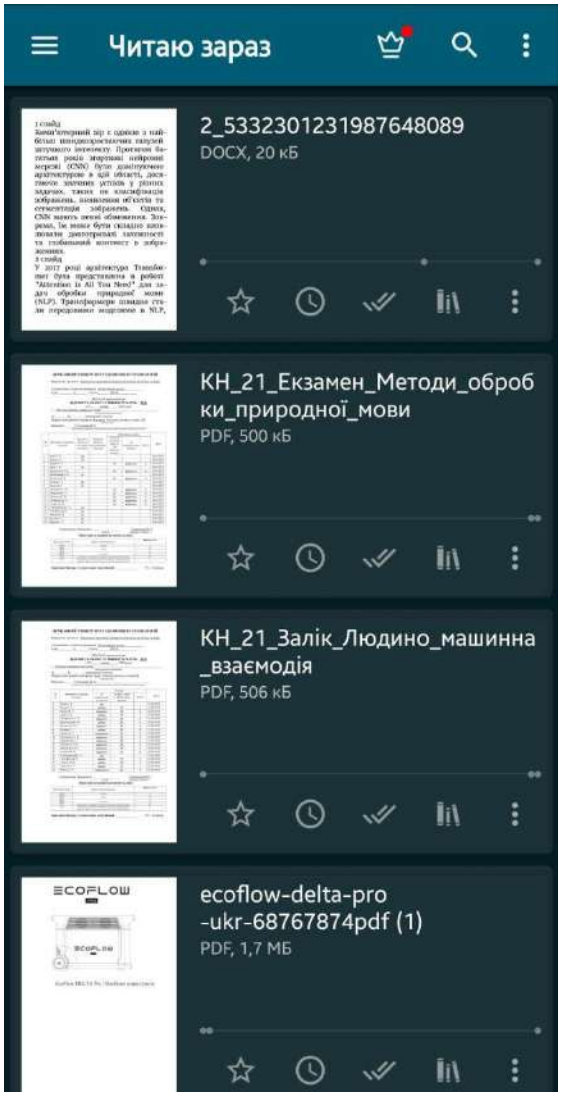

Рис. 1.5. Скріншот застосунку ReadEra

Примітка. Джерело: Розроблено із використанням [28]

Bookmate представляє модель передплати з соціальними елементами, орієнтовану на стримінговий доступ до ліцензійного контенту. Сторонні файли імпортуються у форматі EPUB або FB2, але кількість таких завантажень обмежена тарифом, а DRM-засоби сервісу блокують експорт. Таким чином користувач втрачає контроль над придбаними поза сервісом виданнями.

Підсумовуючи, жоден із розглянутих застосунків не забезпечує одночасно повну кросплатформність, свободу форматів, захист придбаного контенту та зручні інструменти впорядкування великої бібліотеки. Закриті екосистеми обмежують користувача DRM-політикою, відкриті рішення страждають від браку синхронізації й офіційної інтеграції магазину, а універсальні інструменти типу Calibre вимагають високої технічної грамотності. Саме ці недоліки стають ключовими аргументами на користь розробки веб-орієнтованого SPA-застосунку, який поєднає переваги хмарних сервісів і локального керування файлами, зберігаючи контроль за бібліотекою у руках користувача й не зв’язуючи його з певним постачальником контенту чи платформою.

1.3 Визначення функціоналу застосунку і постановка завдання

Цифровий світ характеризується постійним зростанням обсягів особистих даних, які потребують надійного управління, обробки та зберігання. Робота з особистими файлами включає не тільки їх створення і організацію, але також облік доступу, обробку метаданих і забезпечення конфіденційності. У цьому контексті важливим стає аналіз бізнес-логіки, яка керує цими процесами, з урахуванням як технічних, так і організаційних аспектів.

Потреба в автоматизації процесів управління файлами є однією з ключових вимог у багатьох сферах, включаючи корпоративний сектор, освіту, охорону здоров'я та інші. Основними завданнями управління файлами є забезпечення швидкого доступу до необхідної інформації, оптимізація процесів зберігання даних, забезпечення безпеки та конфіденційності інформації, інтеграція з іншими системами для обміну даними.

Бізнес-логіка управління файлами повинна охоплювати такі аспекти, як контроль доступу, система прав та ролей, алгоритми індексації та пошуку, а також механізми синхронізації даних у реальному часі. Для ефективного функціонування системи важливо розробити чіткі механізми аутентифікації користувачів, що забезпечать доступ до файлів лише авторизованим особам.

Система має включати можливості гнучкого налаштування прав доступу залежно від ролі користувача чи політики організації.

Важливим аспектом є процес створення та зберігання файлів. Користувачі повинні мати можливість завантажувати файли, створювати нові документи чи імпортувати дані з інших джерел. Метадані файлів, такі як дата створення, автор, останні зміни, мають зберігатися автоматично для забезпечення організації та пошуку.

Пошук та індексація є ключовими функціями для управління великими обсягами даних. Оптимізовані алгоритми повинні забезпечувати швидке знаходження інформації за ключовими словами, датами або іншими параметрами. Це особливо важливо у випадках роботи з великими бібліотеками документів чи книг, які можуть мати значний обсяг метаданих.

Система також повинна забезпечувати можливість архівування та безпечного зберігання даних у випадку технічних збоїв. Автоматичне резервне копіювання забезпечить відновлення даних у разі їх втрати. Видалення даних має відповідати політикам конфіденційності, зокрема шляхом налаштування процесів видалення, які гарантують захист персональної інформації.

Управління файлами і книгами в межах розроблюваного проєкту вимагає особливої уваги до інтеграції з іншими системами. Це дозволяє об'єднувати дані в єдину екосистему, підвищуючи ефективність обміну інформацією та її використання. Для цього необхідно створити гнучку платформу, яка дозволить адаптувати функціонал під потреби різних типів користувачів та забезпечить масштабованість відповідно до зростаючих вимог.

Висновки до розділу 1

Підсумовуючи, ми можемо підкреслити важливість використання сучасних технологій для управління файлами в умовах цифровізації. Систематизація, автоматизація та інтеграція процесів дозволяють значно підвищити зручність користування і спрощують доступ до інформації.

Автоматизація управління файлами забезпечує користувачам нові можливості, такі як швидке сортування, пошук і резервне копіювання.

Особливу увагу приділяємо забезпеченню конфіденційності та захисту даних. Високий рівень безпеки гарантується за рахунок багаторівневого шифрування, автентифікації та налаштувань контролю доступу.

Одним із ключових аспектів наших досліджень стало управління файлами цифрових книжок. Аналіз показав, що сучасні застосунки для роботи з книжками часто не відповідають сучасним вимогам. Більшість із них застарілі, не мають веб-інтерфейсу для роботи в мережі або доступні лише на одній платформі, що обмежує їхню гнучкість і функціональність. Це створює суттєві незручності для користувачів, які працюють із великими бібліотеками цифрових видань.

Наші дослідження спрямовані на визначення основних підходів для ефективного управління файлами книжок. Ми дійшли висновку, що сучасна система повинна підтримувати кросплатформність завдяки універсальності WEB браузерів, інтеграцію з хмарними сервісами, автоматичне сортування та синхронізацію даних між пристроями. Такий підхід забезпечує легкість у користуванні, підвищує продуктивність та дозволяє уникати втрати даних.

Таким чином, аналіз предметної області та потреб користувачів підтверджує доцільність впровадження системи управління файлами бібліотеки книжок у вигляди WEB SPA застосунку. Це забезпечує зручність, ефективність і безпеку, створюючи базу для інноваційних рішень, які відповідають сучасним викликам.

РОЗДІЛ 2
МОДЕЛЮВАННЯ ПРОЦЕСІВ УПРАВЛІННЯ ОСОБИСТОЮ БІБЛІОТЕКОЮ

2.1 Особливості автоматизації управління файлами книжок

У цифровому середовищі автоматизація управління файлами книжок стає однією з ключових технологічних інновацій, які покращують ефективність і зручність роботи із великими обсягами інформації. Цей процес включає застосування програмного забезпечення, яке дозволяє організувати, зберігати, шукати та обробляти дані про книжки у централізованій системі. Особливу роль відіграє автоматизація у прискоренні доступу до інформації, оптимізації внутрішніх процесів. Важливою перевагою є також можливість швидкого розподілу оновлень даних та їх резервного копіювання, що гарантує збереження інформації навіть у разі технічних збоїв. Таким чином, автоматизація управління файлами книжок не лише спрощує роботу з інформацією, а й створює нові можливості для розвитку.

Одним із ключових аспектів автоматизації управління файлами книжок є вибір інструментів. Наприклад, у нашому проєкті "Library SPA", побудованому на основі Laravel, Vue.js та Vite, забезпечується ефективна взаємодія між серверною і клієнтською частинами. Інтеграція PWA (Progressive Web App) дозволяє користувачам отримувати доступ до даних книжок навіть у режимі офлайн, завдяки кешуванню запитів та статичних ресурсів.

Для забезпечення безпеки та гнучкості управління даними використовуються такі інструменти, як Inertia.js, ZiggyVue, та інші бібліотеки. Зокрема, Laravel Sanctum використовується для автентифікації користувачів, а робота з базами даних виконується через ORM-інструменти, такі як Eloquent.

Service Worker (Workbox) динамічно кешує обкладинки, мініатюри та запити API, а також створює офлайн-репліку з вибраних книг. У результаті користувач може продовжувати читати та переглядати бібліотеку без мережі.

Автоматизація управління файлами книжок включає кілька ключових етапів:

- збір та внесення даних. Застосування форм введення даних із валідацією, що мінімізує людські помилки. Інтеграція з API для автоматичного отримання метаданих книжок;
- сортування та каталогізація. Використання тегів і категорій для структурування даних. Створення деревовидної структури, яка дозволяє швидко знаходити необхідну інформацію;
- пошук і фільтрація. Застосування пошуку по метаданих і вмісту книжок;
- обробка та збереження. Формати файлів, такі як PDF, EPUB, FB2, або MOBI, автоматично конвертуються залежно від потреб користувачів, забезпечуючи універсальний доступ;
- кешування та оптимізація. Інструменти, такі як Workbox, використовуються для кешування часто запитуваних даних, що підвищує швидкодію системи.

Запропонований підхід вирізняється насамперед універсальністю: він однаково підтримує настільні браузери, мобільні пристрої та офлайн-режим завдяки мінімальній зв’язці Laravel + Vue + Service Worker. Така архітектура дає перевагу перед нативними програмами, оскільки усуває потребу в окремих збірках для iOS, Android і Windows, зменшує витрати на підтримку та пришвидшує доставку оновлень.

Незважаючи на переваги, автоматизація управління файлами книжок стикається з низкою викликів. Зокрема, це забезпечення високого рівня безпеки даних, оптимізація для роботи з великими обсягами інформації, а також розробка інтуїтивно зрозумілого інтерфейсу для кінцевих користувачів.

Перспективи автоматизації в цій галузі включають розширення можливостей аналізу даних, впровадження технологій машинного навчання для автоматичної класифікації книжок, а також інтеграцію з хмарними платформами для забезпечення глобального доступу. Поєднання локального сховища й Redis-кешу на сервері забезпечує миттєвий пошук і сортування десятків тисяч записів

без помітних затримок. Це критично для користувачів, що подорожують або працюють у місцях з нестабільним інтернет-з’єднанням. Крім того, відкритий стек технологій і відсутність ліцензійних обмежень знижують бар’єр входу для нових розробників.

2.2 Логічна модель автоматизованої системи управління файлами

При розробці SPA застосунку, створення логічної моделі є важливим завданням, націленим на абстрагування сутностей і взаємозв'язків між ними, спрямованим на формалізацію ключових елементів системи та їх взаємозв'язків. Логічна модель забезпечує концептуальне уявлення про основні функції системи та їхню взаємодію, що дає змогу чітко визначити архітектуру майбутнього програмного продукту.

Основною метою створення логічної моделі є опис функціональних можливостей системи та визначення структур даних, які використовуватимуться для її реалізації. Для досягнення цієї мети система повинна забезпечувати:

- облік та організацію файлів;
- швидкий доступ до інформації через пошукові механізми;
- зручне сортування і фільтрацію файлів;
- забезпечення безпеки даних через функції обмеження доступу.

Цей етап має важливе значення, оскільки зумовлює основу, на якій базуватиметься робота сайту, та допомагає виявити й уточнити функціональні та нефункціональні вимоги.

Логічна модель слугує основою для розроби архітектури застосунку. Шляхом формалізації концептуальних аспектів проєкту вона дає змогу визначити структуру та організацію компонентів системи, а також способи їхньої взаємодії. Це забезпечує архітектурну цілісність і знижує ризик небажаних протиріч і конфліктів на рівні реалізації.

Також розробка логічної моделі дає змогу опрацювати ключові аспекти проєкту на абстрактному рівні, що сприяє виявленню потенційних проблем і

ризиків на ранніх стадіях розробки. Це, зокрема, дає змогу визначити набір тестових сценаріїв і критеріїв перевірки, а також виявити й усунути потенційні вразливості та помилки на ранніх стадіях розробки, а також значно скорочує час, який буде витрачений на їх виправлення. Крім того, логічна модель полегшує процес тестування і налагодження програмного продукту.

Слід зазначити, що створення логічної моделі також сприяє підвищенню прозорості та зрозумілості коду, що спрощує процес його підтримки та супроводу в майбутньому. Розробники, аналізуючи логічну модель, можуть краще зрозуміти структуру та логіку програми, що дає їм змогу ефективніше здійснювати різні види підтримки, включно з додаванням нової функціональності та виправленням помилок.

Для моделювання логічної структури автоматизованого обліку користувачів SPA-застосунку ефективним є використання UML-діаграми прецедентів [8, 6], яка дозволяє наочно визначити основні взаємодії між користувачем та системою. Вона використовується для опису функціональності системи з точки зору її користувачів. Вона допомагає моделювати взаємодію між системою та її оточенням, включно з акторами (користувачами, іншими системами) і прецедентами (функціональними можливостями системи).

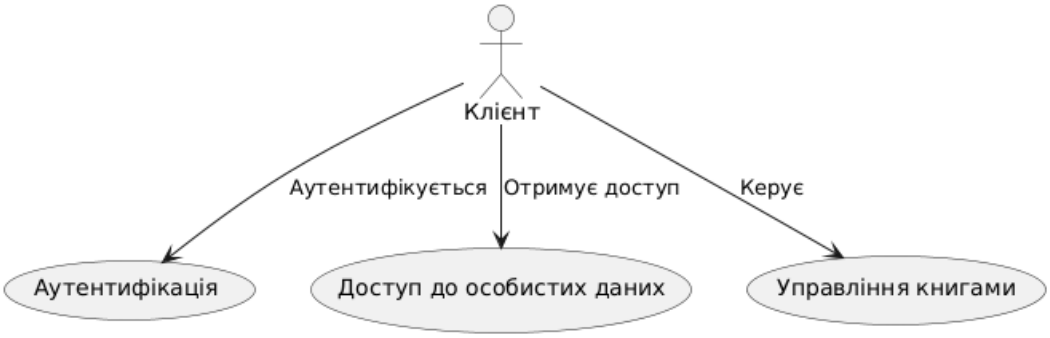

Рис. 2.1. Діаграма прецедентів процесу користування сайтом

(Розроблено автором)

Ця діаграма повинна містити усі основні прецеденти які були виділені при описанні бізнес-логіки у особистій бібліотеці. Зокрема це може бути «Облік

користувачів», «Аутентифікація», «Авторизація» та «Управління книжками», які будуть включати основні дії з відповідними сутностями (додавання, редагування та видалення).

Ця діаграма загалом містить базові прецеденти які можуть бути в подальшому реалізовані у програмному продукті для роботи SPA веб-сайту, та фактично на початковому рівні описує функціональні можливості створюваної системи.

Наступним логічним кроком у процесі проектування автоматизованого обліку відвідувачів особистої бібліотеки є розробка діаграми класів UML [3], що візуалізує структуру об'єктів системи та їхні взаємозв'язки. Вона використовується для графічного представлення структури та відносин між сутностями у створюваній системі. Цей інструмент використовується для візуалізації архітектури програмної системи, полегшення аналізу та проєктування, а також для обміну інформацією між розробниками та зацікавленими сторонами. При цьому класи, зображуються у вигляді блоків, а відносини між класами подаються стрілками та лініями, що показують асоціації.

Базуючись на проведеному аналізу предметної області та бізнес логіки у індустрії застосунків для організації, зберігання а також читання книжок та створеній діаграмі прецедентів можна попередньо передбачити необхідність використання наступних класів: «Книга», «Користувач», «Контролер книг», «Контролер завантажень», «Зчитувач книг». При цьому в якості основного класу може бути вказаний клас «Книга». Розроблена відповідна діаграма класів представлена на рисунку 2.2.

На останньому етапі розробки логічної моделі процесу обліку книжок доцільно розглянути діаграми послідовностей для кожного прецеденту, але враховуючи, що даний проект є навчальним, та те, що діаграми послідовностей для кожного прецеденту будуть мати схожий характер, розглянемо тільки послідовність прецеденту додавання нової книги. Відповідна діаграма представлена на рисунку 2.3.

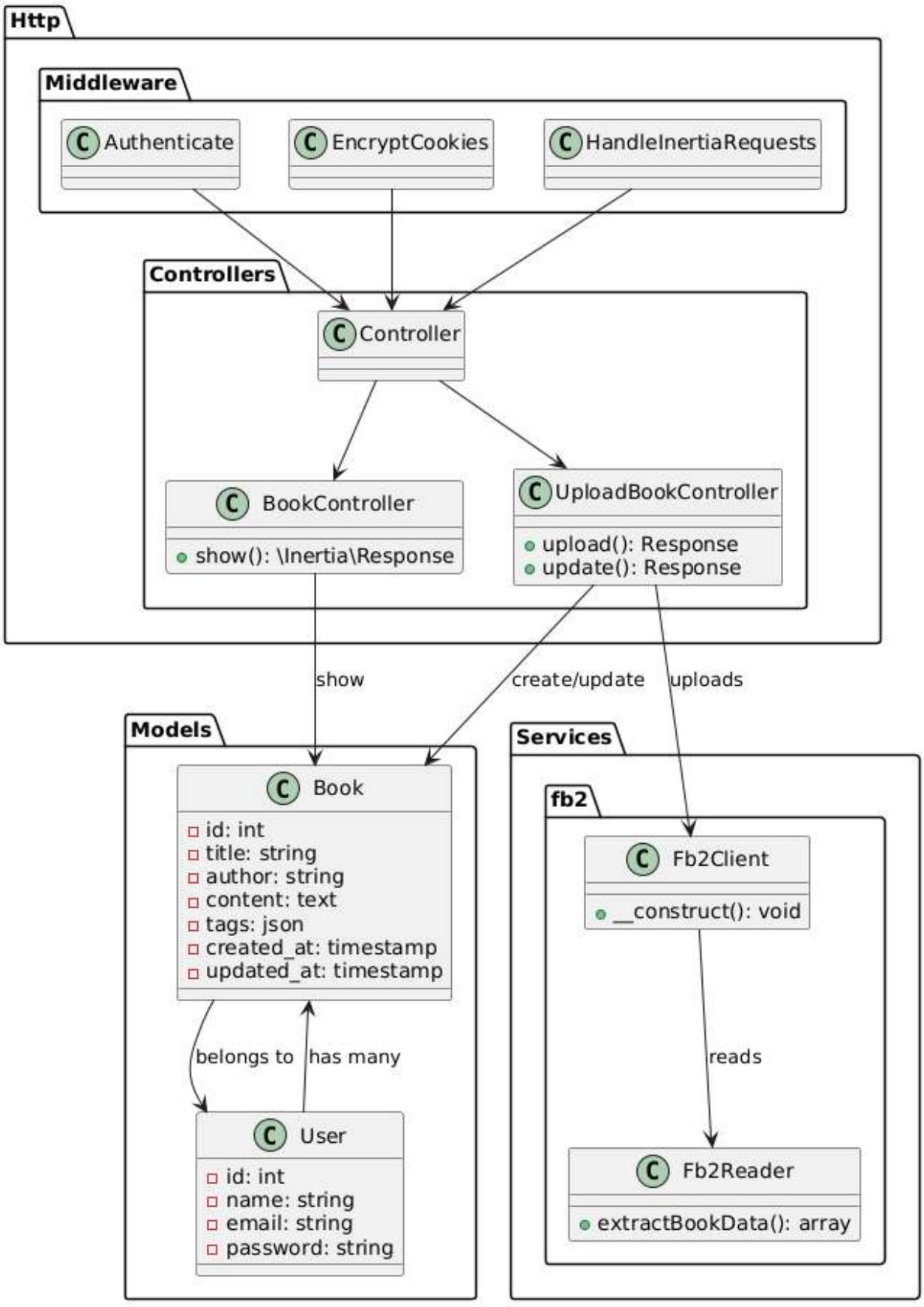

Рис. 2.2. Діаграма класів веб-сайту

(Розроблено автором)

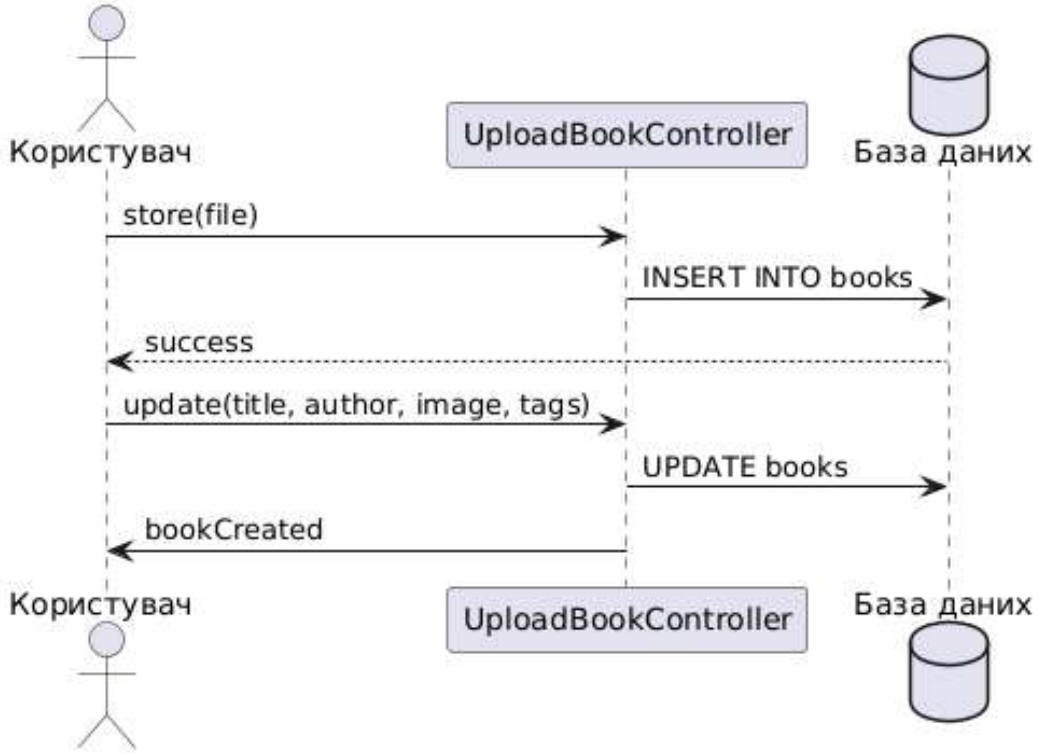

Рис. 2.3. Діаграма послідовності прецеденту додавання нової книги користувачем

(Розроблено автором)

Передбачається необхідність авторизації користувача перед початком роботи з веб-сайтом. Після успішної авторизації користувач може ініціювати додавання нової книги, для чого вочевидь повинна бути передбачена відповідна форма для реєстрації і введення особистих даних користувача, таких як ім’я, адреса електронної пошти, пароль, що загалом відповідає сутності «Користувач», яка описана в діаграмі класів, представленій на рисунку 2.2.

2.3 Фізична модель та компоненти SPA для особистої бібліотеки

Розробка фізичної моделі програмного продукту являє собою один з ключових етапів у циклі розробки інформаційних систем. Цей процес включає в себе детальне проектування архітектури програмного рішення з метою визначення його фізичних компонентів, їх взаємозв'язків і розміщення в

обчислювальному середовищі [16, **Ошибка! Источник ссылки не найден.**].

На самому початку цього етапу команда розробників аналізує високорівневі вимоги до системи і перетворює їх на конкретні специфікації компонентів та інтерфейсів. Потім відбувається декомпозиція системи на окремі функціональні елементи, які згодом стають об'єктами фізичної моделі.

Пророблена фізична модель містить опис кожного компонента системи, його атрибутів, інтерфейсів і взаємозв'язків з іншими компонентами. На основі цієї моделі визначається розподіл обчислювальних ресурсів, а також мережева архітектура, необхідна для функціонування системи.

Мета розробки фізичної моделі програмного продукту, зокрема, полягає в забезпеченні чіткого розуміння архітектурних рішень та оптимізації продуктивності системи. Шляхом аналізу фізичної моделі можливе виявлення потенційних проблем, пов'язаних із продуктивністю або масштабованістю, і вжиття заходів для їх усунення.

Крім того, фізична модель служить основою для планування інфраструктури, необхідної для розгортання і підтримки програмного продукту. Це охоплює вибір апаратного та програмного забезпечення, конфігурацію мережевих ресурсів і механізмів моніторингу та управління.

Фізична модель також відіграє важливу роль у забезпеченні узгодженості та взаєморозуміння між учасниками проекту. Вона слугує основою для комунікації між архітекторами, розробниками, системними адміністраторами та іншими зацікавленими сторонами, забезпечуючи загальне розуміння структури та функціонування системи.

Таким чином, розробка фізичної моделі програмного продукту є невід'ємним етапом у процесі створення інформаційних систем, забезпечуючи основу для подальшої реалізації та успішного функціонування розроблюваного продукту. При цьому розробка фізичної моделі добре вписується в циклічний технологічний процес створення програмного забезпечення. Цей процес зазвичай містить послідовність етапів, починаючи від аналізу вимог і закінчуючи підтримкою та супроводом продукту.

На початкових етапах цього процесу розробки, команда фахівців будує найбільш укрупнені діаграми, які відображають загальну концепцію та архітектуру майбутнього програмного продукту. Ці діаграми на початкових етапах, зазвичай, являють собою досить узагальнене представлення системи, які допомагають визначити основні компоненти системи та їхні взаємозв'язки.

Наступним кроком є поступове уточнення деталей архітектури системи та її компонентів, включно з більш детальною проробкою фізичної моделі. У цьому процесі будуються більш деталізовані діаграми, які відображають фізичний розподіл компонентів системи та їхню взаємодію.

Однак важливо зазначити, що процес розробки програмного забезпечення зазвичай є ітеративним. Це означає, що розробники можуть проводити кілька циклів аналізу, проектування, реалізації та тестування, кожен з яких доповнює й уточнює попередній. У результаті кожної ітерації уточнюється і доповнюється зокрема й фізична модель програмного продукту відповідно до нових вимог, виявлених проблем чи результатів пошуку більш оптимальних рішень.

Таким чином, розробка фізичної моделі програмного продукту є важливою частиною процесу створення програмного забезпечення, та дає змогу команді розробників поступово уточнювати й удосконалювати концепцію та архітектуру продукту впродовж усього процесу розробки.

Тому на початковому етапі доцільно розглянути узагальнену можливу фізичну структуру програмного продукту.

Фізична модель SPA (односторінкового застосунку) для особистої бібліотеки включає детальне описання архітектури, компонентів та їх взаємодії. Ця система створена для забезпечення зручного доступу до інформації, зберігання даних та їх ефективного використання. Вона включає кілька основних компонентів.

Бекендова частина цього застосунку реалізована на основі фреймворку Laravel, який є потужним інструментом для створення сучасних веб-додатків. Laravel забезпечує підтримку RESTful API, що дає можливість інтегрувати різні сервіси та функції в єдине середовище. Це дозволяє фронтендовій частині

застосунку швидко взаємодіяти з сервером, отримуючи необхідні дані або надсилаючи запити для оновлення інформації. Ключова особливість Laravel — це зручність роботи з даними, оскільки фреймворк підтримує ORM (Object-Relational Mapping) через Eloquent. Це значно спрощує взаємодію з базами даних, дозволяючи виконувати складні SQL-запити через зручний об'єктно-орієнтований синтаксис.

Для чіткого розуміння майбутньої архітектури системи варто розробити попередню структуру бази даних, яка послужить фундаментом для кросплатформного застосунку з автоматизованого обліку користувачів SPA-застосунку. Відповідна розроблена схема використовуваної бази даних представлена на рисунку 2.4.

База даних у цій системі організована за допомогою MySQL, яка є однією з найбільш надійних та масштабованих реляційних систем управління базами даних. Архітектура бази містить три основні таблиці: «Книги», «Користувачі» і «Персональні токени доступу».

Таблиця «Книги» структурована так, щоб зберігати ключову інформацію про книжкові ресурси. Вона містить унікальний ідентифікатор книги, ID користувача, який завантажив книгу, назву, автора, шлях до обкладинки, текстовий вміст книги, список тегів у форматі JSON, статус завантаження («очікує», «завершено», «не вдалося»), статус читання («не розпочато», «читається», «завершено»), дату останнього відкриття, позицію читання у відсотках, шлях до файлу книги та розмір файлу у кілобайтах. Ця структура дозволяє відстежувати прогрес користувача у читанні та забезпечувати гнучке керування статусами книги.

Таблиця «Користувачі» зберігає дані про користувачів системи. Вона включає унікальний ідентифікатор користувача, ім'я, електронну пошту, яка є унікальною для кожного облікового запису, дату підтвердження пошти, зашифрований пароль, токен для запам'ятовування сеансу, ID поточної команди (для можливого командного доступу) та шлях до фотографії профілю. Поля для роботи з автентифікацією та підтримки сесій створені для забезпечення високого

рівня безпеки та зручності використання.

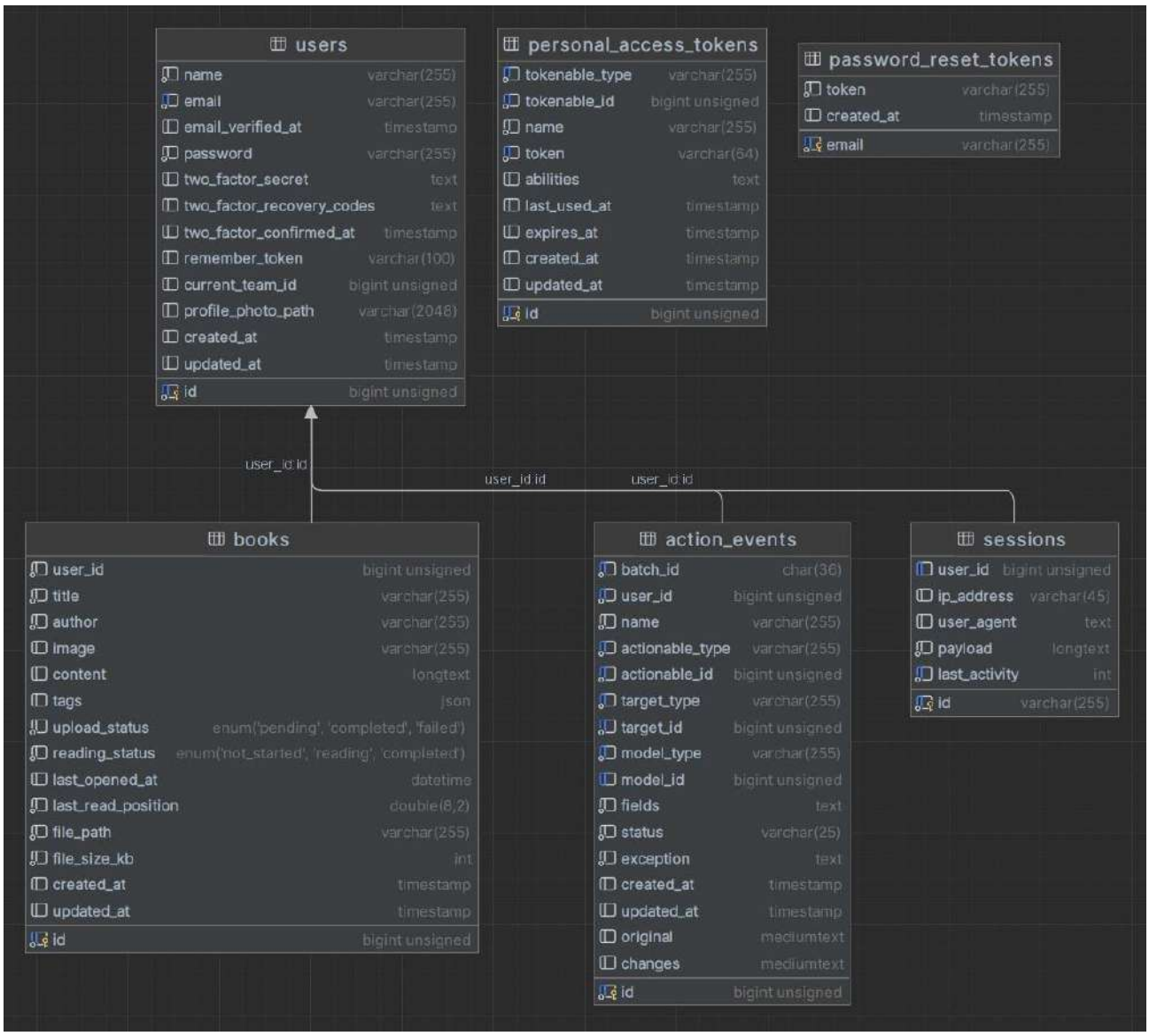

Рис. 2.4. Схема бази даних

(Розроблено автором)

Таблиця «Персональні токени доступу» забезпечує можливість роботи з токенами, які надають користувачам доступ до API. Вона містить унікальний ідентифікатор токена, інформацію про сутність, яка його створила (morphs), назву токена, зашифроване значення токена, набір дозволених дій (у вигляді списку), час останнього використання та термін дії. Це дозволяє забезпечувати гнучке управління доступом та деталізувати права користувачів на виконання різних операцій у системі.

Модуль аутентифікації також реалізовано за допомогою Laravel Sanctum, який забезпечує безпечну роботу користувачів із системою. Sanctum дозволяє працювати з токенами доступу, що гарантує безпеку обробки персональних даних. Завдяки цьому користувачі можуть авторизуватися в системі, отримувати персоналізований доступ до даних та здійснювати операції в межах дозволених прав. Ця функція особливо важлива для бібліотечної системи, яка повинна забезпечувати високий рівень конфіденційності.

Для оптимізації продуктивності системи впроваджено Redis — інструмент для кешування даних. Redis дозволяє зберігати часто запитувану інформацію, наприклад, популярні книги чи списки авторів, у швидкодоступному кеші, що значно зменшує навантаження на сервер і покращує час відгуку API. Крім того, ця технологія дозволяє системі залишатися функціональною навіть за умов високої інтенсивності запитів.

Фронтенд-частина SPA для особистої бібліотеки реалізована на використанні сучасних технологій, які забезпечують максимальну ефективність та зручність у розробці і користуванні. Основу проєкту складають Laravel 10, Inertia.js та Vue.js 3, що разом створюють високопродуктивний інтерактивний центр між серверною і клієнтською частинами. Використання SSR (серверний рендеринг) надає можливість швидкого завантаження сторінок, оптимізації для пошукових систем та плавного користувацького досвіду завдяки усуненню необхідності повного перезавантаження сторінок. Vite виконує роль збирача проєкту, забезпечуючи швидку розробку, зручне тестування і оптимальну загрузку ресурсів.

Inertia.js виконує роль мосту між Laravel та Vue.js, що усуває потребу в традиційному REST API. Це забезпечує передачу даних між сервером та клієнтом у вигляді JavaScript-компонентів, роблячи SPA частиною серверного застосунку. Завдяки цьому кожна сторінка миттєво відображається як Vue-компонент, що значно прискорює взаємодію користувачів із системою. Додатково, Inertia.js дозволяє мінімізувати складність коду, зосереджуючи всю логіку та інтерфейс в єдиній системі.

Vue.js 3 забезпечує гнучкість і реактивність застосунку завдяки компонентному підходу. Використовуючи сучасні можливості JavaScript, Vue.js дозволяє створювати динамічні інтерфейси з легкістю. Його потужний інструментарій та підтримка TypeScript допомагають виявляти помилки ще на етапі розробки, забезпечуючи стабільність і продуктивність застосунку. Завдяки Vue.js інтеграція анімацій, реактивних даних та багатокомпонентних структур відбувається швидко і з мінімальними витратами ресурсів.

SSR (Server-Side Rendering) виконує ключову роль у підвищенні продуктивності та доступності застосунку. Завдяки попередньому рендерингу сторінок на сервері забезпечується миттєве завантаження вмісту, що є особливо важливим для SEO-оптимізації та користувачів із повільним інтернет-з'єднанням. Це також дозволяє зменшити навантаження на клієнтські пристрої, надаючи користувачам оптимальний досвід незалежно від їхніх технічних можливостей.

Vite є інноваційним інструментом для побудови проєктів, який забезпечує неймовірну швидкість завдяки сучасним підходам до бандлінгу. Від використання швидкого сервера розробки до підтримки сучасних стандартів JavaScript, Vite дозволяє розробникам зосередитися на створенні функціоналу без затримок. Його підтримка гарячого перезавантаження і миттєвого оновлення змін забезпечує безперервність роботи і швидкий зворотний зв'язок під час розробки.

Laravel-Vue-PWA інтегрує функціонал прогресивних веб-додатків (PWA) у SPA, забезпечуючи зручний доступ навіть у режимі офлайн. Ця технологія використовує Service Worker для кешування статичних ресурсів, що дозволяє застосунку залишатися функціональним за відсутності інтернет-з'єднання. Завдяки можливості автоматичного оновлення контенту, користувачі завжди працюють з актуальними даними, не помічаючи затримок чи проблем із синхронізацією.

Service Worker відіграє ключову роль у забезпеченні офлайн-доступу. Він відповідає за збереження основних ресурсів і кешування API-запитів. Це

дозволяє застосунку працювати навіть за відсутності інтернет-з'єднання, надаючи користувачам доступ до збережених даних і мінімізуючи затримки. Завдяки цьому технологія ідеально підходить для застосунків, які потребують високої доступності.

TailwindCSS забезпечує швидку і гнучку стилізацію інтерфейсу. Ця технологія використовує утилітарний підхід до класів, що дозволяє створювати адаптивний і сучасний дизайн із мінімальними витратами часу. Завдяки потужним інструментам кастомізації TailwindCSS адаптується до будь-яких потреб проєкту, забезпечуючи візуальну привабливість і зручність користування.

Таким чином виконана загальна попередня розробка логічної та фізичної моделей створюваного SPA застосунку. Розроблені моделі дозволяють послідовно сформувати уявлення про основні концепції та засади, які мають бути покладені в основу створюваного програмного продукту, що дозволяє перейти безпосередньо до його програмної реалізації. При цьому слід зазначити, що створені моделі можуть зазнати значних доповнень та уточнень у процесі подальшої розробки.

Висновки до розділу 2

Розробка фізичної моделі та компонентів SPA для особистої бібліотеки демонструє використання сучасних технологій для створення потужного, адаптивного та продуктивного застосунку. Бекендова частина SPA для особистої бібліотеки розроблена з урахуванням найкращих практик. Laravel, MySQL, Sanctum та Redis утворюють потужну та надійну базу, яка забезпечує високу продуктивність, безпеку та гнучкість у використанні. Завдяки цьому система може обробляти великі обсяги даних і залишатися стабільною навіть під час пікових навантажень, що робить її ідеальною для сучасного використання.

Інтеграція Laravel 10, Inertia.js і Vue.js 3 дозволила створити ефективну взаємодію між серверною та клієнтською частинами, мінімізуючи затримки та

оптимізуючи продуктивність. Застосування SSR (Server-Side Rendering) забезпечило швидке завантаження сторінок і покращену SEO-оптимізацію, що є критично важливим для сучасних веб-додатків.

Використання Vite дозволило значно прискорити розробку завдяки підтримці сучасних стандартів JavaScript, гарячого перезавантаження і миттєвого оновлення змін. Це забезпечило безперервність роботи та підвищило зручність для розробників. Додатково, завдяки інтеграції Laravel-Vue-PWA із Service Worker, додаток отримав можливість роботи в офлайн-режимі, що значно підвищує його доступність для користувачів незалежно від наявності стабільного інтернет-з’єднання.

Інтеграція TailwindCSS забезпечила швидке створення сучасного, адаптивного та естетичного інтерфейсу. Цей підхід дозволяє економити час на розробку стилів і легко адаптувати дизайн під потреби проекту. Потужні інструменти кастомізації TailwindCSS дозволяють швидко вносити зміни та підлаштовувати інтерфейс під змінні вимоги.

Загалом, розроблена фізична модель і компоненти SPA для особистої бібліотеки демонструють успішну інтеграцію провідних технологій для забезпечення стабільності, продуктивності та зручності. Реалізовані підходи та технології створили міцну основу для розробки адаптивного, масштабованого та доступного веб-застосунку.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Опис основних модулів та їх взаємодії

Основні модулі системи та їх взаємодія визначені на основі аналізу бізнес-логіки, вимог користувачів і потреб в автоматизації управління бібліотекою книг.

Модуль авторизації та автентифікації.

Модуль реалізовано з використанням Laravel Sanctum для забезпечення безпечного доступу до системи. Він відповідає за реєстрацію, вхід до системи, вихід та управління сесіями користувачів. Взаємодія між фронтендом і бекендом здійснюється через API-запити з використанням токенів доступу. У разі успішної автентифікації користувач отримує персоналізований доступ до бібліотеки.

Модуль управління книгами.

Цей модуль забезпечує додавання, та перегляд книг. Інтерфейс користувача побудований на Vue.js і дозволяє швидко взаємодіяти з даними без перезавантаження сторінок. Для зберігання інформації використовується MySQL, а ORM-інструмент Eloquent спрощує роботу з базою даних. Користувачі можуть сортувати книги за різними критеріями, додавати теги та змінювати статус читання.

Особливістю функціоналу додавання книг є реалізація процесу через дві послідовні форми. У першій формі користувач вводить файл книги. Після заповнення першої форми дані тимчасово зберігаються і обробляються сервером. Таким чином книга за допомогою написанного нами парсера розкладається на елементи і записується в базу даних. Потім в форму повертається результат обробки і дані про книгу. Далі користувач переходить до другої форми, де отримує попередньо опрацьовані результати і додає за необхідності додаткові деталі, такі як назва, автор, теги та завантажує файл обкладинки. Такий підхід дозволяє структурувати процес внесення даних і забезпечує більш інтуїтивний

інтерфейс для користувача. Всі дані передаються на сервер у вигляді двох запитів, спочатку на завантаження книги і потім на редагування додаткових деталей.

Модуль пошуку та фільтрації.

Функціонал пошуку реалізовано з використанням індексації метаданих книг. Модуль дозволяє швидко знаходити книги за автором, назвою, тегами або іншими параметрами. Для забезпечення швидкої відповіді на запити використовується кешування результатів пошуку за допомогою Redis.

Модуль синхронізації та кешування.

Для роботи застосунку в режимі офлайн впроваджено підтримку PWA (Progressive Web Application) через інтеграцію Service Worker. Цей модуль відповідає за збереження основних даних у кеші, забезпечуючи доступ до бібліотеки навіть без підключення до інтернету. Redis використовується для кешування часто запитуваних даних і зменшення навантаження на сервер.

Модуль управління інтерфейсом.

Графічний інтерфейс побудовано на основі TailwindCSS, що забезпечує адаптивність і сучасний дизайн. Компоненти Vue.js забезпечують динамічну взаємодію користувача із системою, використовуючи механізм реактивності для миттєвого оновлення даних на екрані. Inertia.js виступає мостом між фронтендом і бекендом, зменшуючи складність коду та спрощуючи передачу даних.

Взаємодія між модулями.

Модулі застосунку взаємодіють через API-запити, які обробляються на бекенді за допомогою Laravel. Inertia.js забезпечує передачу даних у вигляді Vue-компонентів, що підвищує продуктивність системи. Service Worker працює у фоновому режимі, синхронізуючи дані між сервером і клієнтом. Redis оптимізує доступ до даних, зберігаючи часто використовувану інформацію в кеші. Завдяки використанню цих технологій забезпечується безшовна робота всіх модулів і висока швидкість реакції системи.

Таким чином, описані модулі та їх взаємодія створюють потужну і гнучку платформу для управління бібліотекою книг, забезпечуючи користувачам

зручний доступ, високу продуктивність і безпеку.

3.2 Створення веб-застосунку

У процесі розробки прикладної частини системи ми дотримувалися традиційної для Laravel схеми "маршрут → контролер → представлення → модель", поступово зводячи каркас, а потім насичуючи його функціоналом.

На початковому етапі роботи необхідно було створити базовий каркас Laravel-застосунку та забезпечити надійний контроль версій за допомогою приватного репозиторію на платформі GitHub. Процес розпочався з виконання команди створення проєкта (рис. 3.1) [4].

```
PS C:\Users\Anrail\PhpstormProjects> composer create-project laravel/laravel book.local
Creating a "laravel/laravel" project at "./book.local"
Cannot use laravel/laravel's latest version v12.0.8 as it requires php ^8.2 which is not
Installing laravel/laravel (v10.3.3)
  - Downloading laravel/laravel (v10.3.3)
  - Installing laravel/laravel (v10.3.3): Extracting archive
Created project in C:\Users\Anrail\PhpstormProjects\book.local
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
```

Рис. 3.1. Створення проєкту

(Розроблено автором)

Дана команда дозволила швидко отримати чисту структуру проекту на базі фреймворку Laravel 10, включаючи стандартний набір залежностей та конфігураційних файлів.

Наступним кроком було встановлення ключа безпеки застосунку (рис. 3.2).

Це забезпечило генерацію унікального ключа шифрування у файлі .env, що критично важливо для безпечного функціонування вебзастосунку (наприклад, захист сесій і шифрування cookies).

```
PS C:\Users\Anrail\PhpstormProjects> cd .\book.local\
PS C:\Users\Anrail\PhpstormProjects\book.local> php artisan key:generate

  INFO  Application key set successfully.
```

Рис. 3.2. Встановлення ключа безпеки

(Розроблено автором)

Далі було необхідно налаштувати підключення до бази даних MySQL у конфігураційному файлі `.env`. В цьому файлі були встановлені параметри конфігурації середовища серверу розробки, назви проєкта, та тип бази даних з логіном і паролем.

Тут було визначено тип підключення (mysql), IP-адресу хоста (127.0.0.1), стандартний порт (3306), ім’я бази даних (book_local). Ці налаштування забезпечили взаємодію Laravel-застосунку з MySQL, дозволивши виконувати команди для створення таблиць та роботи з даними.

Після успішного налаштування середовища було прийнято рішення про використання Git для контролю версій коду та створення приватного репозиторію на GitHub. Для цього було створено новий приватний репозиторій за адресою: https://github.com/Anrail/dilomna_book_spa.

Наступними командами здійснено ініціалізацію Git-репозиторію, додано всі файли у версіонування, створено перший коміт та пов'язано локальний репозиторій із віддаленим (рис. 3.3).

```
git init
git add .
git commit -m "Set up a fresh Laravel app"
git remote add origin https://github.com/Anrail/dilomna_book_spa.git
git branch -M main
git push -u origin main
```

Рис. 3.3. Ініціалізація Git-репозиторію

(Розроблено автором)

Цей процес забезпечив безпечне зберігання коду, контроль за всіма змінами і створив основу для подальшої командної роботи над вебзастосунком. Відтепер усі зміни коду стають доступними через приватний репозиторій GitHub, що гарантує цілісність проєкту.

Під час створення контролерів і моделей проекту, було вирішено зробити розширення функціоналу системи щоб максимально пришвидшити рутинний етап створення моделей, контролерів та Nova-ресурсів. Для цього ми розробили власну консольну команду NovaPageCommand, що стала частиною шару автоматизації. Початковим кроком було виконання команди `php artisan make:command NovaPageCommand`, після чого в app/Console/Commands з'явився відповідний клас. У методі "__construct" ми залишили базову ініціалізацію, оскільки вся користувацька логіка зосереджена в методі "handle". Рядок `protected $signature = 'nova:cp {name} {c?}'` задає коротку й запам'ятовувану назву, обов'язковий параметр "name" для базового імені та не-обов'язковий модифікатор c, що дозволяє пропустити генерацію контролера, коли він уже існує або не потрібен.

```
class NovaPageCommand extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'nova:cp {name} {c?}';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Create Model -m & Controller & Nova Resource';
```

```
/**
 * Create a new command instance.
 *
 * @return void
 */
public function __construct()
{
    parent::__construct();
}

/**
 * Execute the console command.
 *
 * @param $name
 * @return int
 */
public function handle()
{


    if ($this->argument('c') === null)
    {
        Artisan::call('make:controller',[
            'name' => $this->argument('name').'Controller'
        ]);
        $this->comment('Controller: done!');
    }
    Artisan::call('nova:resource',[
        'name' => $this->argument('name').'Resource',
        '-m' => $this->argument('name'),
    ]);
    $this->comment('Resource: done!');
    Artisan::call('make:model',[
        'name' => $this->argument('name'),
```

```
            '-m' => true,
        ]);
        $this->comment('Model: done!');
        return 0;
    }
```

Після запуску команди з аргументом, наприклад `php artisan nova:cp Page`, система послідовно відпрацьовує три виклики Artisan::call. Спершу, якщо другий параметр відсутній, створюється PageController, про що консоль одразу повідомляє фразою "Controller: done!". Далі формується Nova-ресурс PageResource із ключем -m, який одночасно породжує міграцію, а вже потім генерується сама модель Page разом із файлом міграції, забезпечуючи цілісність назв і шляхів. Така послідовність виключає випадкові конфлікти імен і гарантує, що модель існуватиме до моменту використання її в ресурсі.

Метод "handle" повертає 0 при успішному завершенні, що дозволяє інтегрувати команду у скрипти CI/CD й отримувати коректний статус виконання. Реєстрації команди до масиву "$commands" у Kernel.php відбувається автоматично якщо додано клас в директорії app\Console\Commands, після цього виклик доступний у будь-якому середовищі, від локального Docker-контейнера до staging-серверів. На рисунку 3.4 видно, як у консолі пробігає пара "Resource: done! " та "Model: done! ", після чого файли створюються без втручання розробника.

```
PS C:\OSPanel\home\book.local> php artisan nova:cp Library controller
Resource: done!
Model: done!
```

Рис. 3.4. Приклад виклику NovaPageCommand

(Розроблено автором)

Використання NovaPageCommand унеможливлює розходження між назвами класів і таблиць, зменшує час створення типового CRUD-модуля з кількох хвилин до кількох секунд, а також дисциплінує команду, закріплюючи

єдиний шаблон. З практичної точки зору це дало відчутний приріст швидкості розробки: за один робочий день було розгорнуто дванадцять нових ресурсів, кожен із власними міграціями та контролерами, причому жодного разу не виникло колізій або пропущених namespace. На скріншоті 3.4 представлено приклад виклику, де параметр controller вимикає генерацію контролера, що особливо зручно для моделей, які взаємодіють лише через Nova-інтерфейс.

Таким чином NovaPageCommand перетворилася на важливий інструмент внутрішнього конвеєра, органічно доповнивши попередньо описану схему «маршрут → контролер → представлення → модель» і заклавши фундамент для подальших оптимізацій.

Наступним кроком стала розробка моделі Book, що включає в себе необхідні атрибути для опису книг у системі. Для цього було використано команду `php artisan nova:cp Book`, яка одночасно створила модель і файл міграції. У файлі міграції визначено структуру таблиці books, що включає поля id, user_id, title, author, image, content, tags, upload_status, reading_status, last_opened_at, last_read_position, file_path, file_size_kb, created_at та updated_at. Виконання міграцій здійснювалося командою `php artisan migrate`.

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('books', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id');
            $table->string('title');
            $table->string('author');
            $table->string('image')->nullable();
            $table->longText('content')->nullable();
            $table->json('tags')->nullable();
```

```
                $table->enum('upload_status',          ['pending',
'completed', 'failed'])->default('pending');
                $table->enum('reading_status',     ['not_started',
'reading', 'completed'])->default('not_started');
                $table->dateTime('last_opened_at')->nullable();
                $table->float('last_read_position',     8,     2)-
>default(0.0);
                $table->string('file_path');
                $table->integer('file_size_kb');
                $table->timestamps();
            });
        }

        /**
         * Reverse the migrations.
         */
        public function down(): void
        {
            Schema::dropIfExists('books');
        }
    };
```

Для забезпечення функціоналу завантаження книг створено контролер UploadBookController через команду `php artisan make:controller UploadBookController` (рис. 3.11). Метод store цього контролера відповідає за перевірку файлів, зберігання їх у storage/books та передачу до сервісу Fb2Reader. Fb2Reader, у свою чергу, взаємодіє з Fb2Client, який забезпечує обробку та отримання метаданих книг формату FB2, таких як заголовок, автор, анотація, жанр, та інші важливі дані. Це дозволило централізувати логіку обробки різних форматів книжок у майбутньому.

```
    class UploadBookController extends Controller
    {
        protected $fb2Reader;

        public function __construct(Fb2Reader $fb2Reader)
```

```
        {
            $this->fb2Reader = $fb2Reader;
        }

        public function store(Request $request)
        {
            $request->validate([
                'file' => 'required|file|max:5120', // Maximum 5MB
            ]);

            $file = $request->file('file');
            $originalName = $file->getClientOriginalName();
            $filePath = $file->store('books');
            $fileSizeKb = round($file->getSize() / 1024, 2);
            $bookData = $this->fb2Reader->extractBookData($filePath);


            $book = Book::create([
                'user_id' => $request->user()->id,
                'title' => $bookData['title'] ?? 'Unknown Title',
                'author' => $bookData['author'] ?? 'Unknown Author',
                'image' => $bookData['image'] ?? null,
                'tags' => $bookData['tags'] ?? null,
                'content' => $bookData['content'] ?? null,
                'upload_status' => 'completed',
                'file_path' => $filePath,
                'file_size_kb' => $fileSizeKb,
            ]);

            return response()->json([
                'status' => 'success',
                'message' => 'Book uploaded successfully.',
                'bookData' => $book,
```

```
            ], 201);
        }

        public function update(Request $request, $id)
        {
            Book::findOrFail($id)->update($request->all());
        }
    }
```

Доступ до вже завантажених книжок регламентують два маршрути у файлі web.php. Перший приймає POST-запит і потребує санкції Sanctum — таким чином сторонні користувачі не мають можливості завантажувати довільні файли. Другий маршрут /books/{id} має метод GET і повертає конкретний екземпляр. Його обробляє BookController, створений через php artisan make:controller BookController.

```
    Route::middleware([
        'auth:sanctum',
        config('jetstream.auth_session'),
        'verified',
    ])->group(function () {
        Route::get('/dashboard', function () {
            $userBooks = Book::query()->select('id','title','author','image','tags')->where('user_id', auth()->id())->get();
            return Inertia::render('Dashboard', [
                'books' => $userBooks,
            ]);
        })->name('dashboard');
        Route::get('/book/{id}', [BookController::class, 'show'])->name('show_book');
        Route::post('/upload-book', [UploadBookController::class, 'store'])->name('upload_book');
        Route::put('/update-book/{id}', [UploadBookController::class, 'update'])->name('update_book');
    });
```

Метод index формує пагіновану вибірку, відфільтровану за ідентифікатором поточного користувача, а show застосовує Book::findOrFail($id) і передає результат у Vue-компонент Book.vue.

```
<script setup>
import { usePage } from '@inertiajs/vue3';
import { ref, computed } from 'vue';

const props = defineProps({
    book: {
        type: Array,
        default: () => [],
    },
});
const book = computed(() => props.book || {});

const fullScreen = ref(false);
const toggleFullScreen = () => {
    fullScreen.value = !fullScreen.value;
};
const goBack = () => {
    window.history.back();
};
</script>
<template>
    <div class="book-view">


        <div
            @dblclick="toggleFullScreen"
            :class="fullScreen ? 'fixed inset-0 bg-black text-
white p-4 overflow-auto z-50' : ''"
            class="book-content   bg-white   dark:bg-gray-800
rounded shadow-lg p-4"
        >
            <button
```

```
                    @click="goBack"
                    class="mb-4 p-2 bg-gray-300 rounded hover:bg-
gray-400 transition"
                >
                    Назад
                </button>
                <h1 class="text-2xl font-bold mb-4 text-gray-800
dark:text-white">
                    {{ book.title }}
                </h1>
                <div class="text-gray-700 dark:text-gray-300 text-
justify" v-html="book.content"></div>
            </div>
        </div>
    </template>

    <style scoped>
    .book-content {
        //max-width: 800px;
        width: auto;
        padding-left: 50px;
        padding-right: 50px;
        margin: auto;
        text-align: justify;
        border-radius: 0;
    }

    .book-view button {
        cursor: pointer;
    }
    </style>
```

Файл resources/js/app.js (додаток А) виконує роль клієнтського ядра: одразу після імпорту bootstrap та Tailwind CSS ініціалізується PWA-середовище через registerSW із режимом autoUpdate, що дозволяє прозоро доставляти оновлення service-worker без участі користувача. Далі createInertiaApp створює інстанс Vue

3, під’єднує плагін Inertia й бібліотеку Ziggy, яка транспортує маршрути Laravel у фронтенд і забезпечує коректне формування URL-ів у шаблонах. Метод resolvePageComponent з підтримкою import.meta.glob гарантує динамічне завантаження компонентів за схемою Pages/Name.vue, тому бандл не розростається непотрібними частинами, а заголовок кожної сторінки генерується функцією title, що додає суфікс із назвою програми й забезпечує єдину стилістику вкладок браузера. Параметр progress визначає колір індикатора навігації, щоб він узгоджувався з темною палітрою Tailwind, і не потребує сторонніх CSS.

Серверна частина resources/js/ssr.js дзеркально відтворює клієнтську, але використовує createSSRApp і @vue/server-renderer для генерації HTML на боці Node.

```
import { createSSRApp, h } from 'vue';
import { renderToString } from '@vue/server-renderer';
import { createInertiaApp } from '@inertiajs/vue3';
import createServer from '@inertiajs/vue3/server';
import { resolvePageComponent } from 'laravel-vite-plugin/inertia-helpers';
import { ZiggyVue } from '../../vendor/tightenco/ziggy';
import { registerSW } from 'virtual:pwa-register'

const appName = import.meta.env.VITE_APP_NAME || 'Laravel';

createServer((page) =>
    createInertiaApp({
        page,
        render: renderToString,
        title: (title) => `${title} - ${appName}`,
        resolve: (name) => resolvePageComponent(`./Pages/${name}.vue`, import.meta.glob('./Pages/**/*.vue')),
        setup({ App, props, plugin }) {
            return createSSRApp({ render: () => h(App, props)
```

```
})
                    .use(plugin)
                    .use(ZiggyVue, {
                        ...page.props.ziggy,
                        location:                                   new
URL(page.props.ziggy.location),
                    });
            },
        })
    );
    if (typeof window !== 'undefined'){
        import('./pwa')
        registerSW({
            onOfflineReady() {
                console.log("Offline mode is ready");
            },
        })
    }
```

Тут також присутній createInertiaApp, але він приймає об’єкт page і функцію renderToString, що формує повністю гідратований HTML ще до відправки у браузер. Завдяки цьому перший візуальний піксель з’являється значно швидше, а пошукові системи отримують контент без складних JavaScript-евалюацій. У межах setup плагін Ziggy отримує page.props.ziggy, тому посилання, згенеровані на сервері, залишаються дійсними й після гідратації на клієнті. Наприкінці файлу присутня перевірка typeof window !== 'undefined': якщо скрипт виконується у браузері, підключається окремий модуль pwa та повторно викликається registerSW для локального оточення, що уникає подвійної реєстрації під час SSR.

Файл vite.config.js об’єднує великий стек плагінів. laravel-vite-plugin передає шляхи до початкових точок, а також вказує ssr: 'resources/js/ssr.js', тому Vite будує два незалежні бандли — клієнтський і серверний. Підключення vite-plugin-inspect активується лише у production-збірці й записує карту модулів у .vite-inspect, що полегшує аналіз розміру пакунків. @vitejs/plugin-vue

налаштовано на transformAssetUrls: базовий null дає змогу Tailwind-класи кэшувати ресурси без модифікації шляху, а includeAbsolute: false запобігає обробці зовнішніх URL. Найпомітнішим елементом є VitePWA: реєстрація registerType: 'autoUpdate' гарантує, що новий service-worker активується одразу після завантаження й не блокує користувача діалогами. Workbox-секція містить власноруч укладені правила кешування: для Inertia-відповідей із заголовком X-Inertia активується стратегія StaleWhileRevalidate, що миттєво повертає сторінку з кешу, але одночасно підтягує свіжі дані; для звичайних GET-запитів того самого origin застосовується та сама стратегія з окремим cacheName pages-cache. Навмисно відключено navigateFallback, бо Inertia самостійно управляє історією переглядів і не потребує HTML-fallback. Максимальний розмір файлу, який може бути закешований, встановлено 5 MB: цього достатньо для обкладинок і скриптів, але захищає від випадкового потрапляння великих відео. Маніфест описує PWA-метадані: display: 'standalone' іконізує додаток на мобільних пристроях, кольори теми збігаються зі схемою Tailwind, а screenshots допомагають Chrome Web Store попередньо показувати інтерфейс.

У результаті комбінація app.js, ssr.js і vite.config.js формує цілісний ланцюг доставки коду: Vite збирає оптимізовані бандли, серверний рендерінг прискорює перший рендер, а PWA-шар гарантує офлайн-доступ і миттєві оновлення. Така архітектура забезпечує реальну мобільну відчутність навіть на повільних з’єднаннях і робить бібліотеку книг незалежною від якості мережі,

Перехід між книжками здійснюється без перезавантаження завдяки механізму Inertia Response: DevTools демонструє одну-єдину дію X-Inertia-Partial, замість повної перерисовки DOM.

На клієнтській стороні файл app.js ініціалізує InertiaApp, підключає плагін laravel-vue-pwa та реєструє service-worker через virtual:pwa-register (рис. 3.14). Ініціація останнього забеспечує що після першого відвідування кешуються статичні маршрути й обкладинки. В офлайн-режимі Workbox повертає дані з локального сховища, забезпечуючи безперервність читання.

Графічний інтерфейс збудовано на базі Vue 3 та TailwindCSS. Компонент

BookList.vue створює адаптивну сітку карток, причому класи Tailwind автоматично змінюють кількість колонок залежно від ширини дисплея: від однієї на вузьких смартфонах до п’яти на 4K-моніторах.

3.3 Графічний інтерфейс веб-застосунку

Графічний інтерфейс розробленого веб-застосунку "Library SPA" спроектовано з урахуванням принципів зручності, доступності та сучасних дизайн-тенденцій. Основною метою є забезпечення інтуїтивного та зручного досвіду користувача незалежно від його рівня технічних знань. Основні елементи інтерфейсу включають панель аутентифікації, панель навігації, робочу область, форми введення даних та інформаційні панелі.

Головна сторінка застосунку (рис. 3.5) виконана у темній колірній схемі для зниження навантаження на очі користувача під час тривалого користування. У центрі головної сторінки розташовано іконку книги та короткий опис функціоналу сервісу.

Рис. 3.5. Головний екран сайту

(Розроблено автором)

На цьому екрані обираємо посилання на реєстрацію або аутентифікацію.

На рисунку 3.6 ми бачимо форму для входу в дешборд, де необхідно ввести

дані для аутентифікації нашого користувача. Додатково передбачено можливість запам'ятати користувача, а також функція відновлення пароля.

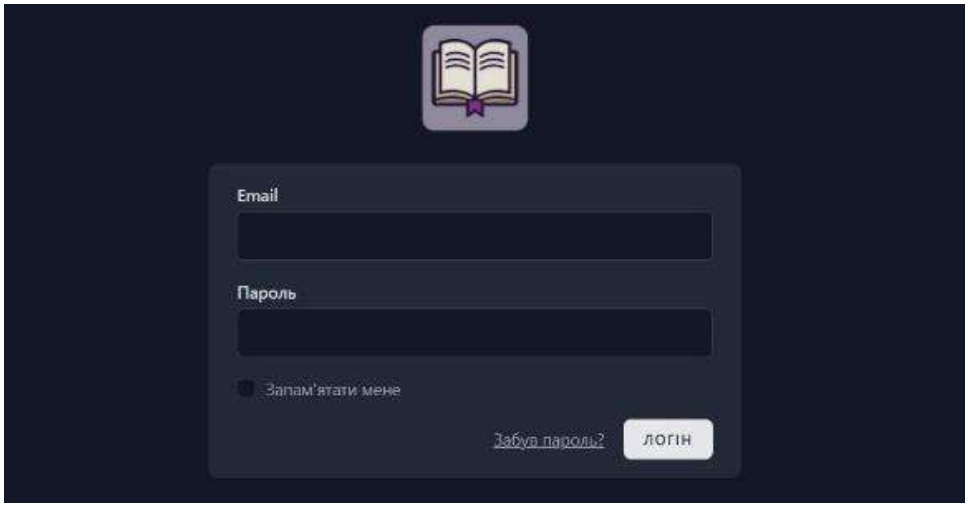

Рис. 3.6. Екран аутентифікації сайту

(Розроблено автором)

Структура проста й інтуїтивно зрозуміла для будь-якого користувача. Або якщо користувач ще не створений, то нам потрібно створити користувача в формі реєстрації (рис. 3.7).

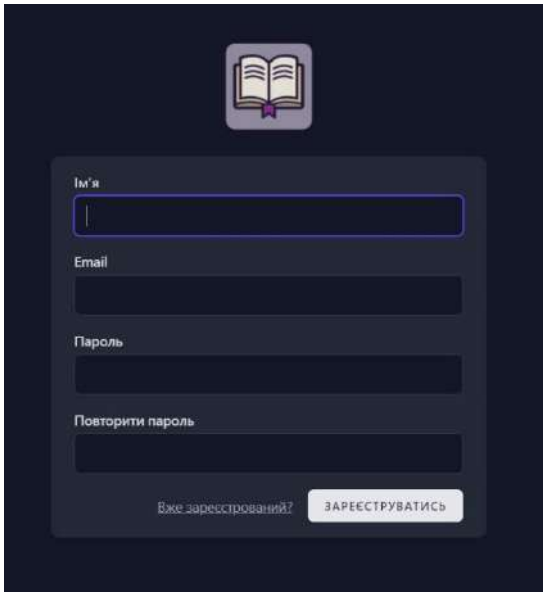

Рис. 3.7. Форма реєстрації користувача

(Розроблено автором)

Після успішної аутентифікації відбудеться перехід на сторінку дешборд.

Основний вміст (рис. 3.8), складається з карток книг, які відображають обкладинку, назву, автора та кнопку "Читати книгу". Картки структуровані у вигляді сітки для зручності перегляду великої кількості контенту. Там представлені всі основні елементи сайту.

Дешборд є основною сторінкою для взаємодії користувача з бібліотекою. Інтерфейс складається з панелі пошуку, де користувач може знайти книги за назвою або автором. Додатково є список тегів для швидкої фільтрації, за жанрами (рис. 3.9-3.10).

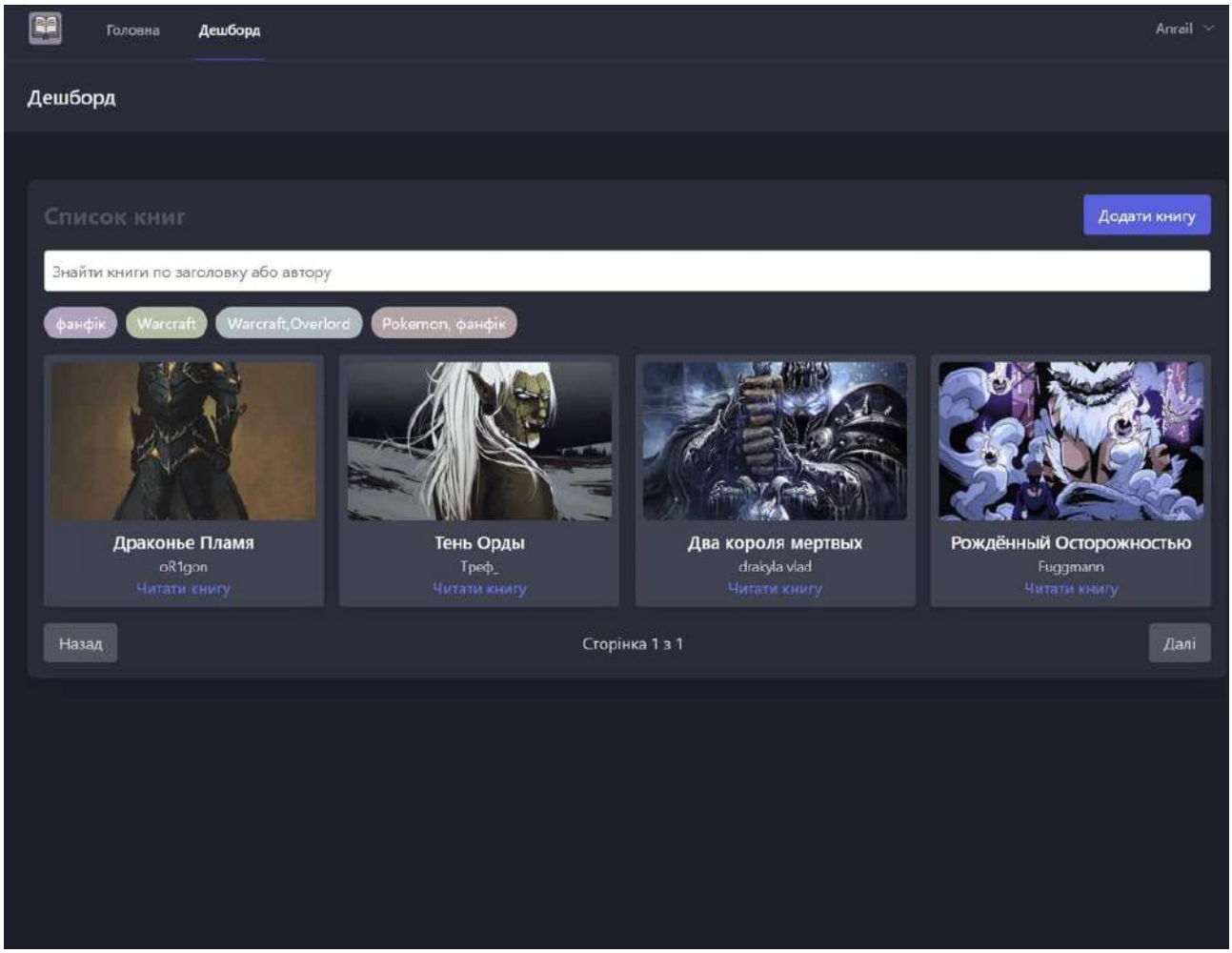

Рис. 3.8. Сторінка дешборд

(Розроблено автором)

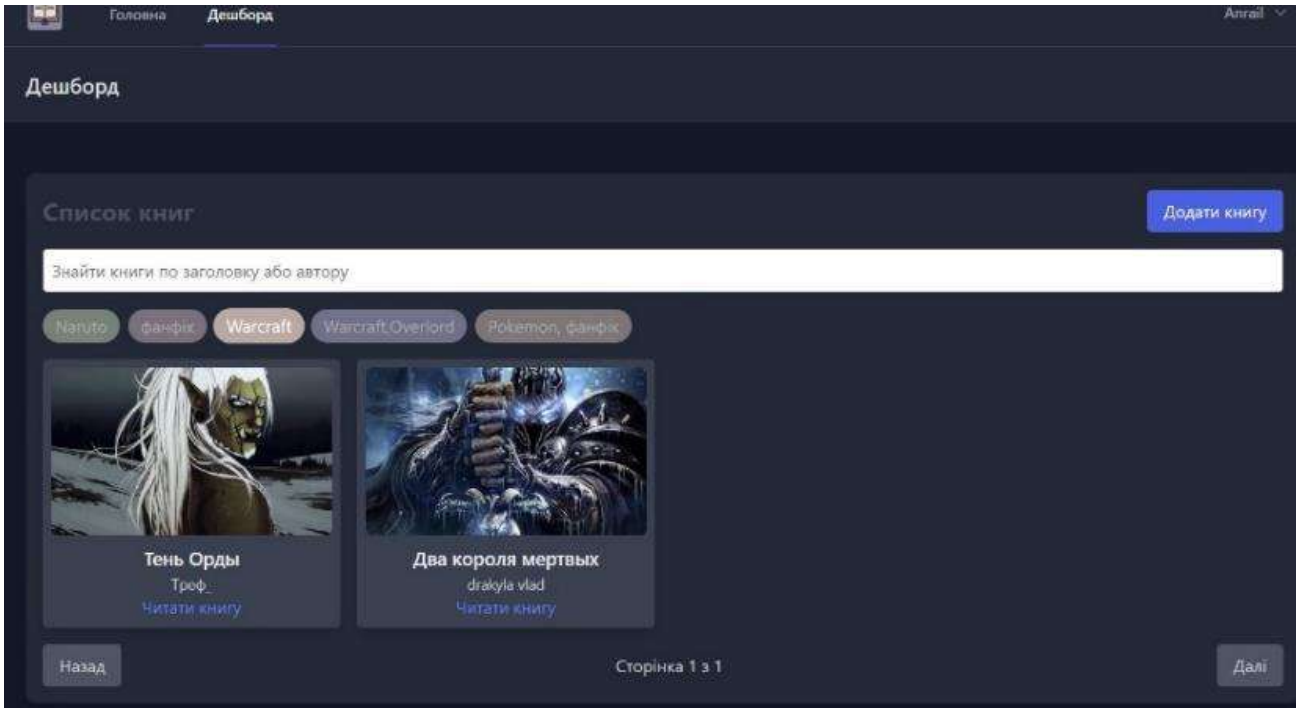

Рис. 3.9. Сортування за тегами

(Розроблено автором)

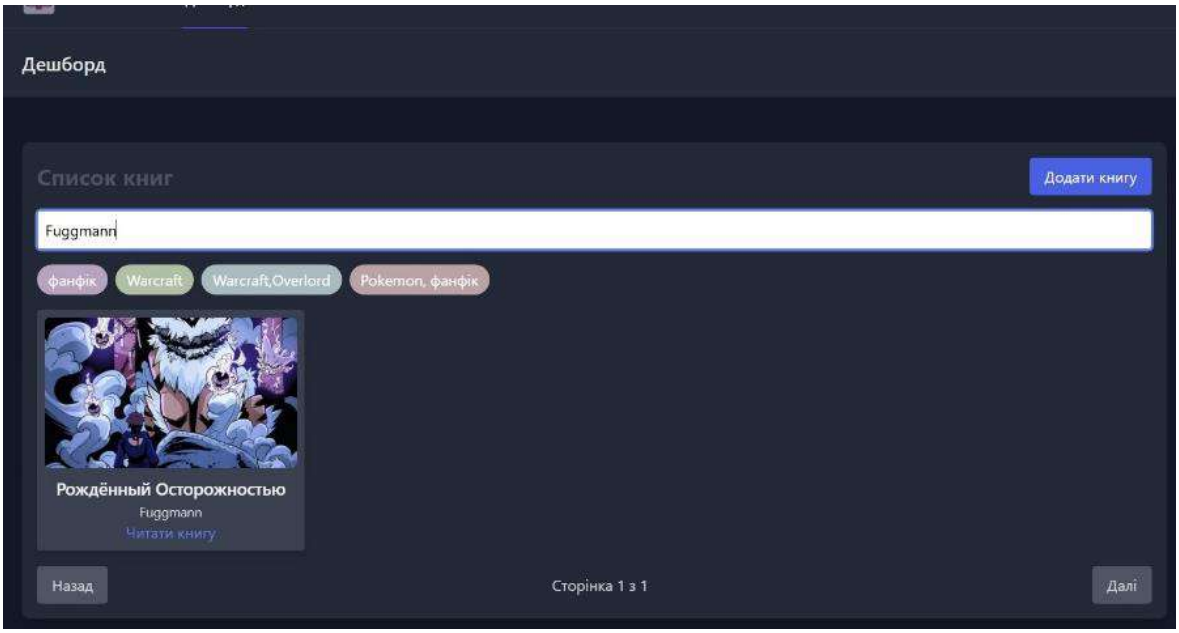

Рис. 3.10. Пошук по автору

(Розроблено автором)

Також реалізовано додавання книг через завантаження файлу. Відображається модальне вікно з відповідними полями для введення (рис. 3.11).

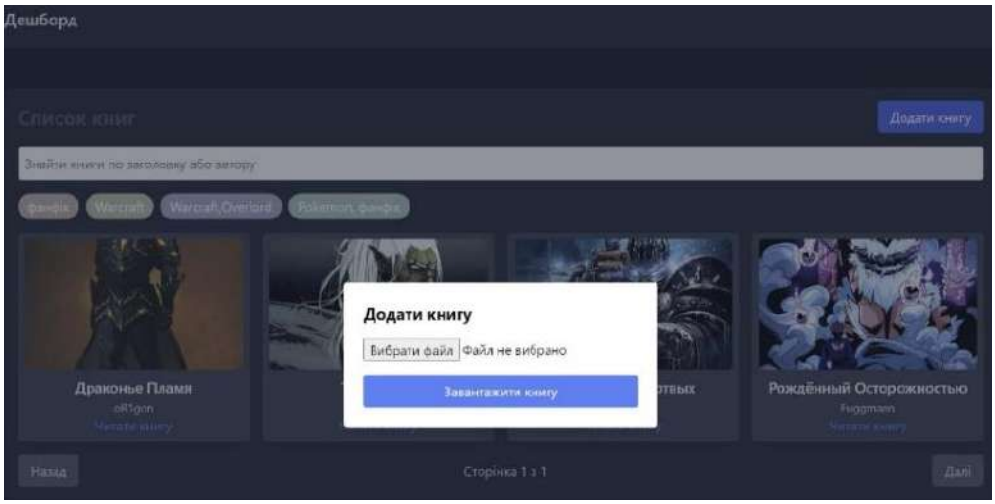

Рис. 3.11. Форма додавання книги

(Розроблено автором)

Після завантаження книги вона обробляється на сервері і відкривається друга модальна форма для додаткового зміну назви, автора, картинки та тегів за необхідності (рис. 3.12). Після додавання книги користувач може миттєво переглядати її в списку доступних ресурсів.

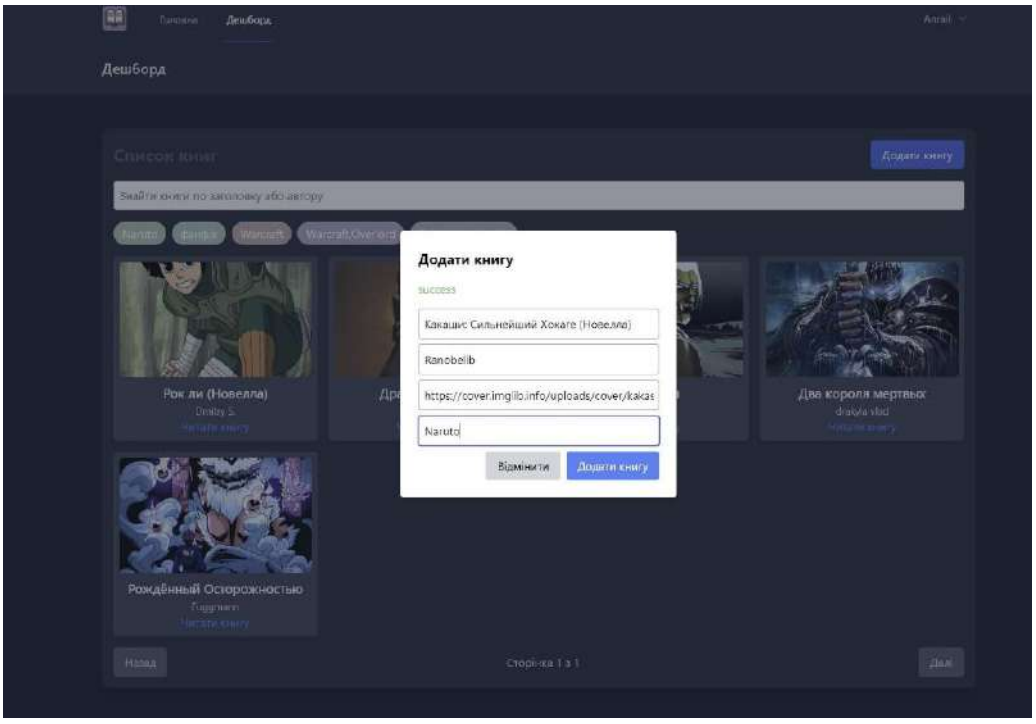

Рис. 3.12. Форма додавання книги

(Розроблено автором)

Інтерфейс для читання книги виконано у форматі повносторінкового режиму з можливістю повернення до списку книг. Текст книги структурований за абзацами для зручності сприйняття (рис. 3.13).

Рис. 3.13. Інтерфейс для читання

(Розроблено автором)

Інтерфейс адаптований для роботи на мобільних, планшетах і десктопах. Елементи дизайну та їх розташування змінюються залежно від розміру екрана для забезпечення максимально комфортного користування (рис. 3.14).

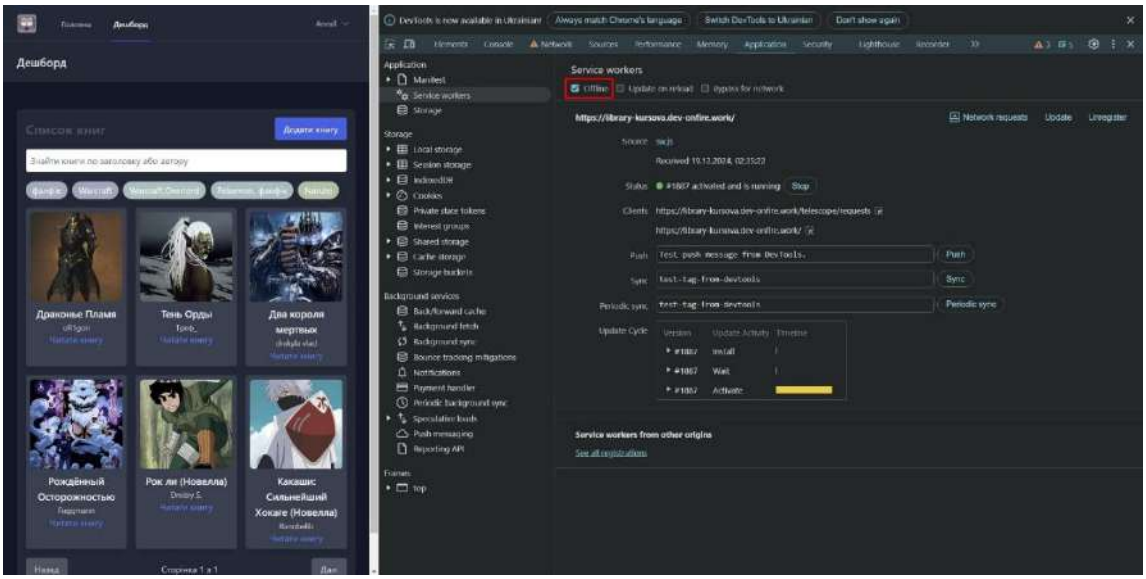

Рис. 3.14. Адаптивність та офлайн режим браузера

(Розроблено автором)

На цьому ж рисунку можна побачити, що веб-сайт працює в режимі офлайн.

Висновки по розділу 3

У третьому розділі роботи було детально описано функціональні можливості розробленого SPA веб-сайту, його графічний інтерфейс, а також надано рекомендації щодо ефективного використання.

Розроблений програмний продукт демонструє успішне поєднання сучасних технологій для створення ефективного односторінкового застосунку (SPA), орієнтованого на управління бібліотекою книг. Основні модулі системи, такі як авторизація, управління книгами, пошук, синхронізація та управління інтерфейсом, виконують ключові завдання для забезпечення зручності та продуктивності користувачів.

Модуль авторизації та автентифікації гарантує безпеку доступу та персоналізацію користувацького досвіду. Управління книгами реалізовано з урахуванням багатокрокового підходу, що дозволяє автоматизувати обробку даних і забезпечити гнучкість у внесенні інформації. Використання двох послідовних форм для додавання книг підвищує точність та інтуїтивність роботи з бібліотекою.

Модуль пошуку та фільтрації дозволяє швидко знаходити необхідну інформацію завдяки ефективним алгоритмам та індексації метаданих. Синхронізація через PWA забезпечує доступність застосунку навіть у режимі офлайн, а Redis оптимізує роботу з даними через механізми кешування.

Інтерфейс користувача побудовано з використанням сучасних підходів, що гарантує адаптивність та високу швидкість взаємодії. Використання таких інструментів, як Vue.js, Inertia.js та TailwindCSS, сприяє покращенню продуктивності та створенню зручного візуального середовища.

Взаємодія між модулями забезпечує цілісність системи, а використання

API-запитів та Service Worker гарантує стабільність роботи навіть за умов високого навантаження. Завдяки цьому система відповідає сучасним вимогам до продуктивності, гнучкості та безпеки.

Таким чином, розроблений програмний продукт є ефективним інструментом для управління бібліотекою книг, що забезпечує користувачам інтуїтивний, швидкий та безпечний досвід роботи.

ВИСНОВКИ

На завершення роботи було проведено повний аналіз, моделювання, розробку та тестування SPA (Single Page Application) для управління бібліотекою книг. Цей проєкт охоплював комплексні завдання, які забезпечили повноцінний цикл розробки сучасного веб-застосунку.

У вступі було визначено актуальність розробки SPA для управління особистою бібліотекою, об’рунтовано вибір використаних технологій, таких як Laravel, Vue.js, Inertia.js, Vite, TailwindCSS та Redis. Обрані інструменти дозволили забезпечити високу продуктивність, зручність у використанні, а також адаптивність до сучасних потреб користувачів. Особливу увагу приділено впровадженню PWA (Progressive Web App), що гарантує доступність даних навіть у режимі офлайн.

У першому розділі проаналізовано предметну область і визначено потреби користувачів. Основні акценти зроблено на автоматизації управління файлами, інтеграції хмарних технологій і забезпеченні безпеки даних. Особливу увагу приділено управлінню цифровими книжками, що включає класифікацію, сортування, зберігання та пошук книжок у різних форматах. Було також розроблено рекомендації для оптимального впровадження цих функцій у системі.

Другий розділ сфокусувався на моделюванні системи. Було створено логічну та фізичну моделі, які забезпечують прозоре уявлення про архітектуру, компоненти та їхню взаємодію. Розроблено структуру бази даних, що включає таблиці для управління книжками, користувачами та токенами доступу. Інтеграція Redis забезпечила високу продуктивність системи через кешування, а використання Laravel Sanctum гарантує безпеку персональних даних.

У третьому розділі детально описано функціональні модулі та інтерфейс розробленого застосунку. Інтерактивний дизайн, створений на основі TailwindCSS, забезпечує зручний користувацький досвід. Використання Inertia.js як мосту між серверною та клієнтською частинами значно спростило розробку,

дозволивши інтегрувати серверний рендеринг (SSR) для покращення SEO-оптимізації та швидкості завантаження сторінок.

Одним із ключових елементів, що забезпечують стабільність і доступність застосунку, є використання Service Worker. Завдяки йому реалізовано кешування основних ресурсів і API-запитів, що дозволяє застосунку функціонувати навіть за відсутності інтернет-з'єднання. Цей підхід не лише покращує швидкодію, а й гарантує безперервний доступ до бібліотеки. Service Worker також виконує роль проміжного шару між мережею та користувачем, що дозволяє більш гнучко управляти оновленнями, забезпечуючи прозору доставку нових версій застосунку без необхідності втручання користувача. Така архітектура є критично важливою для сучасних PWA-рішень і суттєво підвищує надійність роботи системи.

Для ефективного використання SPA для особистої бібліотеки рекомендовано дотримуватись наступних вимог:

- по-перше, важливо забезпечити доступ до стабільного інтернет-з'єднання, оскільки основні функції, такі як завантаження та синхронізація даних, оптимально працюють у режимі онлайн. Проте інтеграція PWA (Progressive Web App) дозволяє використовувати застосунок і в офлайн-режимі. Це забезпечується завдяки кешуванню даних та використанню Service Worker;
- по-друге, для оптимального користування рекомендується встановлювати застосунок на пристрої через браузер, який підтримує функції PWA. Це дозволить зменшити час доступу до бібліотеки, оскільки застосунок відкриватиметься безпосередньо з робочого столу чи головного екрана пристрою.

У результаті виконаної роботи було створено надійну, продуктивну та сучасну платформу для управління бібліотекою книг, яка відповідає вимогам сучасних користувачів і викликам цифрової епохи. Запропоноване рішення демонструє потенціал для подальшого вдосконалення та інтеграції з новими технологіями, забезпечуючи високий рівень функціональності, безпеки та

адаптивності. Цей проєкт слугує відмінним прикладом того, як можна ефективно використовувати сучасні технології для вирішення реальних потреб.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eloquent JavaScript: A Modern Introduction to Programming / Marijn Haverbeke. – 3rd ed. – No Starch Press, 2018. – 472 с.

2. Duckett J. HTML & CSS: Design and Build Websites / Jon Duckett. – Wiley, 2011. – 512 с.

3. Боровик В. М., Труш О. І., Крива О. М. Програмні компоненти проектування діаграм UML. *Problems of Informatization and Management*. 2010. Т. 3. №. 31. С. 14-19.

4. Laravel Team. Laravel Documentation. Laravel Framework. 2024. [Електронний ресурс]. https://laravel.com/docs/10.x (дата звернення 15.11.2024).

5. Vue.js Developers. Vue.js Official Guide. Frontend Development. 2024 [Електронний ресурс]. https://vuejs.org/guide/introduction.html (дата звернення 11.10.2024).

6. Vue.js Up and Running: Building Accessible and Performant Web Apps / Callum Macrae. – O'Reilly Media, 2018. – 174 с.

7. Єремєєв В. С., Курлянський С. С. Можливості мови UML для проектування програмного забезпечення. *Інформаційні технології в освіті та науці: зб. наук. пр*. 2018. №. 10. С. 88-91.

8. Єфремов М. Ф., Єфремов Ю. М., Єфремов В. М. Проектування програмного забезпечення з використанням UML. 2016.

9. REST API Design Rulebook / Mark Masse. – O'Reilly Media, 2011. – 118.

10. Mastering Modular JavaScript / Nicolas Bevacqua. – O'Reilly Media, *2018. – 472 с*.

11. CSS: The Definitive Guide / Eric A. Meyer, Estelle Weyl. – O'Reilly *Media, 2017. – 1096 с*.

12. Fullstack Vue: The Complete Guide to Vue.js / Hassan Djirdeh, Nate *Murray, Ari Lerner. – Fullstack.io, 2018. – 544 с*.

13. Луцков М. П., Новицький М. О., Римар П. В. Забезпечення безпеки даних у BIG DATA. *Прикладні інформаційні технології*. 2020. С. 152-154.

14. Марковець О. В., Синько А. І. Формування якісної технічної документації до програмного забезпечення. *Вісник Вінницького політехнічного інституту*. 2021. №. 2. С. 98-106.

15. Inertia.js Team. Inertia.js Documentation. SPA Framework. 2024 *року* [Електронний ресурс]. https://inertiajs.com/ (дата звернення 5.11.2024).2024.

16. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення: науково-методичний посібник. 2015.

17. Пилипчук В. Г., Брижко В. М. Інформаційна безпека та приватність у сфері захисту персональних даних. *Інформація і право*. 2016. №. 4 (19). С. 60-70.

18. Розумей С. Б., Горбонос Є. С. Процес вибору стратегій управління взаємовідносинами з клієнтами. *Науковий вісник Міжнародного гуманітарного університету. Серія: Економіка і менеджмент*. 2015. №. 12. С. 119-123.

19. Харченко О., Яцишин В. Розробка та керування вимогами до програмного забезпечення на основі моделі якості. *Вісник Тернопільського національного технічного університету*. 2009. №. 66, № 1. С. 201-207.

20. Шалапай Р. І. Технології виявлення функціональних та нефункціональних вимог до комп'ютерних систем. 2023.

21. Шевченко С., та ін. Обґрунтування попереднього вибору архітектури системи обробки даних з використанням нечіткої логіки. *Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології*. 2019. №. 2. С. 81-87.

22. Яковлєва С. О., Міхайлуца О. М., Пожуєв А. В. Дослідження впливу функціональних вимог на якість та тестування програмного забезпечення. *Збірник наукових праць Дніпродзержинського державного технічного університету. Технічні науки*. 2014. С. 82-85.

23. Ярмолюк О. Я. Ремаркетинг та система взаємовідносин з клієнтами. *Проблеми системного підходу в економіці*. 2016. №. 2. С. 86-89.

24. Amazon Kindle. Kindle for Web. 2024 [Електронний ресурс]. https://read.amazon.com (дата звернення 11.10.2024).

25. Google Play Books. Library. 2024 [Електронний ресурс]. https://play.google.com/books (дата звернення 12.10.2024).

26. Kobo Books. 2024 [Електронний ресурс]. https://www.kobo.com (дата звернення 12.10.2024).

27. PocketBook Reader. 2024 [Електронний ресурс]. https://pocketbook.com.ua/uk-ua/app-ua (дата звернення 12.10.2024).

28. ReadEra. 2024 [Електронний ресурс]. https://readera.org (дата звернення 12.10.2024).

ЗГОДА здобувача вищої освіти

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

Я, _Маловічко Олександр Вікторович_(ПІП),

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

Засвідчую, що кваліфікаційна бакалаврська робота

Розробка SPA PWA сервісу електронної бібліотеки

(назва роботи повністю) виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував 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 ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований(на), 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 ознайомлений.

01.06.2025 р.		О.В. Маловічко
Дата	підпис	ініціали, прізвище