МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «**РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ВЕДЕННЯ ОБЛІКУ ГРОМАДСЬКОЇ БІБЛІОТЕКИ З ВИКОРИСТАННЯМ ANGULAR, NODE.JS ТА MONGODB**»

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.П. Тетеря
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тетері Владиславу Павловичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебдодатку для ведення обліку громадської бібліотеки з використанням Angular, Node.js та MongoDB

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та огляд існуючих рішень.
2. Проєктування інформаційної системи.
3. Реалізація та тестування інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.12.2025	
2.	Збір вихідних даних.	20.01.2026	
3.	Обробка методичних та теоретичних	12.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2026	

Студент _____ В.П. Тетеря
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебдодатку для ведення обліку громадської бібліотеки з використанням Angular, Node.js та MongoDB»: 65 с., 18 рис., 9 табл., 22 джерела, 1 додаток.

ANGULAR, DOCKER, JWT, MONGODB, MONGOOSE, NODE.JS, REST API, БІБЛІОТЕКА, БРОНЮВАННЯ, ВЕБРОЗРОБКА, КОНТЕЙНЕРИЗАЦІЯ, ОБЛІК.

Об’єкт дослідження – процес розроблення вебзастосунків для автоматизації облікових операцій.

Предмет дослідження – методи та програмні засоби створення клієнт-серверних вебдодатків із використанням Angular, Node.js та NoSQL бази даних MongoDB.

Мета роботи – скорочення часу на опрацювання облікових операцій та зменшення кількості помилок при веденні бібліотечного обліку шляхом розроблення вебдодатку, що замінює паперовий документообіг у громадській бібліотеці.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення; об’єктно-орієнтоване проєктування; UML-моделювання; методи конструювання та тестування програмного забезпечення.

Реалізовано вебдодаток, що охоплює автентифікацію на основі JWT, процеси позик і бронювання книг із email-сповіщеннями, журнал аудиту та планувальник фонових завдань. Застосунок контейнеризовано за допомогою Docker, а Nginx налаштовано як зворотний проксі-сервер. Результатом роботи є готовий до розгортання вебдодаток, який автоматизує ключові процеси бібліотеки та прискорює виконання облікових операцій.

SUMMARY

Bachelor's qualifying paper "Development of a Web Application for Public Library Management Using Angular, Node.js, and MongoDB": 65 pages, 18 figures, 9 tables, 22 references, 1 supplement.

ANGULAR, DOCKER, JWT, MONGODB, MONGOOSE, NODE.JS, REST API, WEB DEVELOPMENT, LIBRARY, BOOKING, CONTAINERIZATION, RECORD MANAGEMENT.

The object of research is the process of web application development for accounting operations automation.

The subject of research includes methods and tools for client-server web application development using Angular, Node.js, and the MongoDB NoSQL database.

The purpose of the work is to reduce the time required for processing accounting operations and to minimize errors in library management by developing a web application that replaces paper workflow in a public library.

The research methods include software requirements gathering and analysis, object-oriented design, UML modeling, as well as software construction and testing methods.

A web application covering JWT-based authentication, book loan and reservation processes with email notifications, an audit log, and a background task scheduler has been implemented. The application is containerized using Docker, and Nginx is configured as a reverse proxy. The result of the work is a deployment-ready web application that automates key library processes and accelerates accounting operations.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Аналіз предметної області та огляд існуючих рішень 10
1.1 Характеристика предметної області та актуальність автоматизації .. 10
1.2 Огляд та порівняльний аналіз існуючих систем управління бібліотеками 12
1.3 Обґрунтування вибору технологічного стеку 15
1.3.1 Вибір фреймворку для клієнтської частини 16
1.3.2 Вибір платформи для серверної частини 17
1.3.3 Вибір системи управління базами даних 18
1.3.4 Вибір інструментів контейнеризації та розгортання 19
1.4 Постановка задачі, формування вимог та критеріїв якості системи . 20
1.4.1 Функціональні вимоги 21
1.4.2 Нефункціональні вимоги 22
1.4.3 Вимоги до забезпечення якості та тестування 23
1.5 Висновки з розділу 1 23
2 Проєктування інформаційної системи 25
2.1 Обґрунтування архітектури системи 26
2.2 Структурна схема та модульна організація 28
2.3 Проєктування бази даних 31
2.4 Проєктування REST API 33
2.5 Проєктування інтерфейсу користувача та взаємодії 35
2.6 Специфікація бізнес-логіки та життєвих циклів 37
2.7 Висновки з розділу 2 38
3 Реалізація та тестування інформаційної системи 40

3.1 Обґрунтування та опис інструментів розробки 40
3.1.1 Інструменти серверної частини 40
3.1.2 Інструменти клієнтської частини 41
3.1.3 Допоміжні інструменти розробки 41
3.2 Програмна реалізація та архітектура класів системи 42
3.2.1 Структура та реалізація серверної частини 42
3.2.2 Структурна схема та реалізація клієнтської частини 45
3.3 Контейнеризація, розгортання та архітектура оточення 47
3.3.1 Конфігурація середовища Docker 47
3.3.2 Оркестрація локальних сервісів через Docker Compose 49
3.3.3 Налаштування Reverse Proxy у Nginx 49
3.4 Тестування компонентів та модулів системи 50
3.4.1 Модульне тестування 50
3.4.2 Інтеграційне тестування 51
3.4.3 Функціональне тестування системи 52
3.5 Керівництво користувача 54
3.5.1 Рівень підготовки користувача та системні вимоги 54
3.5.2 Вхід до системи 54
3.6 Висновки з розділу 3 57
Висновки 59
Перелік посилань 61
Додаток А Вміст файлу docker-compose.yml 63

ВСТУП

Актуальність теми. Більшість громадських бібліотек в Україні досі ведуть облік книжкових фондів, читачів та видачі літератури у паперовому вигляді. Такий підхід призводить до значних витрат часу на пошук інформації, неможливості оперативно відстежувати терміни повернення книг, складності аналізу статистики та відсутності зручного доступу до каталогу для читачів. В умовах цифровізації суспільних послуг автоматизація бібліотечних процесів є актуальним і практично значущим завданням, що безпосередньо впливає на якість обслуговування читачів та ефективність роботи персоналу.

Аналіз існуючих рішень показує, що комерційні системи управління бібліотеками (АБІС, Koha, OpenBiblio) є складними у розгортанні, потребують значних фінансових витрат або не адаптовані до вимог невеликих громадських бібліотек. Це обумовлює необхідність розроблення спеціалізованого вебдодатку, орієнтованого на конкретні потреби громадської бібліотеки з урахуванням сучасних вимог до безпеки, зручності використання та простоти розгортання.

Мета роботи – скорочення часу на опрацювання облікових операцій та зменшення кількості помилок при веденні бібліотечного обліку шляхом розроблення вебдодатку, що замінює паперовий документообіг у громадській бібліотеці.

Для досягнення поставленої мети визначено такі **завдання**:

- проаналізувати предметну область громадської бібліотеки та виявити основні проблеми паперового документообігу;
- дослідити існуючі системи автоматизації бібліотек та обґрунтувати вибір технологічного стеку;
- сформувати функціональні та нефункціональні вимоги до розроблюваної системи;
- спроєктувати архітектуру вебдодатку та схему бази даних;

– реалізувати серверну частину на основі Node.js, Express та MongoDB;
– розробити клієнтську частину з використанням фреймворку Angular;
– забезпечити контейнеризацію застосунку засобами Docker та Docker Compose;
– провести функціональне тестування розробленої системи.

Об’єкт дослідження – процес розроблення вебзастосунків для автоматизації облікових операцій.

Предмет дослідження – методи та програмні засоби створення клієнт-серверних вебдодатків із використанням Angular, Node.js та NoSQL бази даних MongoDB.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення; об’єктно-орієнтоване проєктування; UML-моделювання; методи конструювання та тестування програмного забезпечення.

Практичне значення результатів. Розроблений вебдодаток може бути безпосередньо впроваджений у громадській бібліотеці замість паперового обліку. Система забезпечує автоматизацію реєстрації читачів, управління фондом книг, видачі та повернення літератури, бронювання видань, а також надсилання автоматичних нагадувань електронною поштою про терміни повернення. Публічний каталог із можливістю онлайн-бронювання підвищує доступність бібліотечних послуг для читачів. Контейнеризація за допомогою Docker спрощує розгортання системи на будь-якій платформі без складного налаштування інфраструктури.

Структура роботи. Кваліфікаційна робота складається з вступу, трьох розділів, висновків, переліку посилань та одного додатку. У першому розділі виконано аналіз предметної області та огляд існуючих рішень, обґрунтовано вибір технологічного стеку і сформульовано вимоги до системи. Другий розділ присвячено проєктуванню архітектури застосунку, бази даних, REST API та інтерфейсу користувача. У третьому розділі описано реалізацію серверної і клієнтської частин, контейнеризацію системи та результати тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Характеристика предметної області та актуальність автоматизації

Громадська бібліотека є культурно-освітньою установою, що відіграє ключову роль у забезпеченні вільного доступу населення до інформації, знань та культурного надбання [1]. З розвитком інформаційного суспільства бібліотеки трансформуються з простих сховищ друкованих видань у сучасні інформаційні центри [2].

Основними бізнес-процесами типової громадської бібліотеки є:

– формування, облік та каталогізація книжкового фонду;
– реєстрація читачів та ведення їхніх карток;
– видача і повернення літератури;
– контроль термінів позики;
– обслуговування інформаційних запитів;
– формування статистичної та аналітичної звітності.

Незважаючи на стрімкий розвиток цифрових технологій в Україні та впровадження концепції «Держава у смартфоні», значна частина невеликих регіональних та місцевих громадських бібліотек досі спирається на застарілі паперові технології. У таких установах облік ведеться за допомогою паперових інвентарних книг, карток читачів (формулярів) та алфавітних чи систематичних каталогів у вигляді дерев’яних ящиків з карточками. Цей традиційний підхід породжує низку критичних проблем та суттєво знижує ефективність роботи установи:

– *надмірні витрати робочого часу бібліотекарів на рутинні операції*: пошук інформації про наявність конкретного видання у фонді може займати від кількох хвилин до півгодини, особливо якщо картка була розміщена неправильно;

- *неможливість оперативного контролю за термінами повернення книг*: виявлення боржників вимагає ручного перегляду сотень читацьких формулярів, що часто призводить до втрати цінних примірників;
- *відсутність зручного механізму бронювання літератури дистанційно*: читач не може дізнатися про наявність книги без фізичного візиту до бібліотеки;
- *складність формування статистичних звітів*: підрахунок кількості виданих книг за місяць, аналіз популярності певних жанрів або авторів вимагає ручної обробки даних, що супроводжується високим ризиком людської помилки;
- *фізична вразливість даних*: паперові документи піддаються зношенню, можуть бути втрачені або пошкоджені внаслідок непередбачуваних ситуацій (пожежа, затоплення), що призведе до повної паралізації роботи бібліотеки;
- *обмеженість доступу*: каталог бібліотеки доступний лише в години її роботи і виключно в її приміщенні.

Автоматизація бібліотечних процесів дозволяє усунути зазначені проблеми шляхом переведення обліку до цифрового формату. Інформаційна система управління бібліотекою (Integrated Library System, ILS) – це комплекс програмного та апаратного забезпечення, призначений для автоматизації внутрішніх бібліотечних процесів та забезпечення доступу користувачів до електронного каталогу.

Вебдодаток як форма реалізації такої інформаційної системи є найбільш оптимальним рішенням на сьогодні. На відміну від десктопних програм, вебдодаток не вимагає встановлення на кожен комп’ютер у бібліотеці, не залежить від операційної системи робочих станцій і легко оновлюється централізовано. Крім того, він надає можливість створити публічний портал (електронний каталог) (Online Public Access Catalog, OPAC), доступний користувачам з їхніх смартфонів чи домашніх комп’ютерів. Це відповідає сучасним вимогам доступності та інклюзивності послуг.

Типова логічна структура даних автоматизованої бібліотечної системи включає такі основні сутності та зв’язки:

– книга (видання) – центральний об’єкт обліку (містить метадані: назву, автора, рік видання, видавництво, унікальний ідентифікатор (ISBN), жанр, мову, а також інформацію про загальну кількість примірників та кількість доступних для видачі одиниць);
– читач (користувач) – містить персональні та контактні дані особи, дату реєстрації, статус активності картки, історію позик та налаштування сповіщень;
– позика (транзакція видачі) – динамічний об’єкт, що фіксує факт передачі певного примірника книги конкретному читачеві (включає дату видачі, планову дату повернення, фактичну дату повернення та статус (активна, закрита, прострочена));
– бронювання – заявка читача на отримання книги, яка наразі відсутня або знаходиться у фонді, з фіксацією дати заявки та статусу обробки адміністратором.

Окрім базового обліку, сучасні системи управління бібліотеками все частіше інтегрують соціальні функції: можливість залишати відгуки, ставити оцінки прочитаним книгам, отримувати персоналізовані рекомендації на основі історії читання. Також критично важливим є ведення детального журналу аудиту (Audit Log), який фіксує всі дії персоналу в системі (хто, коли і які дані змінив), що забезпечує прозорість роботи та захист від зловживань або випадкового видалення важливої інформації.

1.2 Огляд та порівняльний аналіз існуючих систем управління бібліотеками

Ринок програмного забезпечення для бібліотек формувався протягом кількох десятиліть [3]. На сьогодні він пропонує широкий спектр рішень: від

потужних комерційних платформ корпоративного рівня до невеликих open-source проєктів. Для обґрунтування необхідності власної розробки проведемо детальний аналіз найбільш поширених систем, оцінюючи їх за такими критеріями: вартість впровадження та володіння (TCO), складність розгортання, функціональність публічного інтерфейсу, можливості онлайн-взаємодії з читачами та адаптивність до потреб невеликих громадських установ.

Koha – перша у світі вільна (open-source) автоматизована бібліотечна інформаційна система, розробка якої почалася ще у 1999 році в Новій Зеландії. Наразі це один із найпотужніших світових стандартів у галузі. Koha підтримує міжнародні формати бібліографічного опису MARC 21 та UNIMARC, протоколи Z39.50 для обміну даними між бібліотеками, має модулі комплектування, каталогізації, циркуляції фонду та управління періодикою. Незважаючи на те, що система є безкоштовною, вартість її володіння для невеликої бібліотеки може бути високою. Причина полягає у високій складності розгортання та адміністрування: Koha базується на стеку Perl, Apache та MySQL/MariaDB. Вона вимагає глибоких знань Linux-систем для налаштування, оновлення та вирішення проблем. Інтерфейс системи, хоча і розвивається, залишається перевантаженим функціями, які ніколи не знадобляться маленькій місцевій бібліотеці [4].

Evergreen – ще одна потужна open-source АБІС, розроблена в США спеціально для консорціумів бібліотек. Вона чудово справляється з мільйонними фондами та складною ієрархією філій. Однак, як і Koha, Evergreen є занадто «важкою» архітектурно (використовує C, Perl, PostgreSQL, Ejabberd) та абсолютно надлишковою за функціоналом для поодинокої громадської бібліотеки з фондом у кілька тисяч примірників.

АБІС «УФД/Бібліотека» та інші локальні комерційні рішення – системи, що історично набули поширення в українських університетських та великих наукових бібліотеках. Вони адаптовані до вітчизняних стандартів звітності. Проте їх головним недоліком є комерційна модель розповсюдження. Вартість

ліцензій, впровадження та щорічної технічної підтримки є непідйомною для бюджету невеликих територіальних громад. Крім того, багато з цих систем проєктувалися як десктопні застосунки (клієнт-серверна архітектура для локальної мережі) і їхні вебмодулі часто є лише додатковими, не завжди зручними надбудовами, які не забезпечують сучасного користувацького досвіду (UX) для читачів на мобільних пристроях [5].

OpenBiblio – проста та легка автоматизована бібліотечна система з відкритим вихідним кодом, написана на PHP. Довгий час вона була популярним вибором для шкільних та сільських бібліотек завдяки простоті встановлення на звичайний віртуальний хостинг (LAMP стек). Вона містить базовий функціонал: OPAC, циркуляцію, каталогізацію. Проте активна розробка OpenBiblio була зупинена понад десять років тому. Система не відповідає сучасним стандартам веббезпеки, її інтерфейс застарів, вона не є адаптивною (не працює коректно на смартфонах) і не пропонує таких важливих функцій, як автоматичні email-сповіщення, особистий кабінет з історією, або гнучке онлайн-бронювання [6].

Порівняльний аналіз розглянутих систем та цільових характеристик розробленого в рамках даної роботи вебдодатку наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз систем управління бібліотеками

Критерій	**Koha**	**АБІС (Комерційні)**	**OpenBiblio**	**Розроблена система**
Тип ліцензії	Open Source	Комерційна	Open Source	Open Source / Власна
Складність розгортання	Дуже висока	Середня / Вимагає інтегратора	Низька (застарілий стек)	Низька (Docker-контейнери)
Вимоги до сервера	Високі (Linux, Perl)	Залежить від вендора	Мінімальні (PHP 5)	Середні (Node.js, Docker)
Сучасний адаптивний UI/UX	Частково	Зазвичай ні	Ні	Так (Angular Material)
Бронювання онлайн	Так	Залежить від версії	Ні	Так

Продовження таблиці 1.1

Критерій	**Koha**	**АБІС (Комерційні)**	**OpenBiblio**	**Розроблена система**
Публічний онлайн-каталог	Так	Так (як платний модуль)	Базовий	Так
Email-сповіщення	Так (складне налаштування)	Зазвичай ні	Ні	Так (вбудовано)
Особистий кабінет читача	Так	Обмежено	Ні	Так
Журнал аудиту (безпека)	Так	Так	Ні	Так

Як показує аналіз (табл. 1.1), на ринку існує певний вакуум для невеликих бібліотек. Потужні системи (Koha, комерційні АБІС) є фінансово або технічно недоступними. Прості системи (OpenBiblio) безнадійно застаріли. Існує гостра потреба у легкому, сучасному вебдодатку, який можна розгорнути за лічені хвилини за допомогою сучасних технологій контейнеризації, і який би надавав інтуїтивно зрозумілий інтерфейс як для бібліотекаря-пенсіонера, так і для сучасного підлітка-читача зі смартфоном. Саме розробка такого рішення є метою даної роботи.

1.3 Обґрунтування вибору технологічного стеку

Сучасна веброзробка пропонує величезну кількість інструментів, фреймворків та архітектурних підходів. Вибір технологій для розроблення бібліотечного вебдодатку здійснювався на основі аналізу вимог до системи, доступних інструментів, тенденцій ринку праці та їх відповідності завданням проєкту. Архітектура системи побудована за класичним принципом Single Page Application (SPA), де клієнтська та серверна частини є фізично розділеними застосунками, які взаємодіють виключно через REST API. Такий підхід забезпечує високу масштабованість, дозволяє паралельно розробляти фронтенд і бекенд, а в майбутньому легко підключити до того ж API мобільний додаток.

1.3.1 Вибір фреймворку для клієнтської частини

Клієнтська частина (фронтенд) відповідає за взаємодію користувача з системою. Вона повинна швидко реагувати на дії, працювати без перезавантаження сторінки та коректно відображатися на пристроях з різною діагоналлю екрана. Для реалізації клієнтської частини розглядалися три провідні JavaScript-фреймворки: Angular, React та Vue.js. Порівняльний аналіз наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз фреймворків клієнтської частини

Критерій	**Angular**	**React**	**Vue.js**
Тип	Повноцінний фреймворк	UI-бібліотека	Прогресивний фреймворк
Основна мова	TypeScript	JavaScript/JSX	JavaScript
Архітектурний підхід	Модульний (MVC/MVVM)	Компонентний	Компонентний
Управління станом	RxJS, NgRx	Redux, Context API	Vuex, Pinia
Вбудована маршрутизація	Так (Angular Router)	Ні (потрібен React Router)	Ні (потрібен Vue Router)
Масштабованість для Enterprise	Дуже висока	Висока	Середня/Висока
Крива навчання	Висока (крута)	Середня	Низька

Angular обрано як основний інструмент клієнтської частини з низки вагомих причин. По-перше, на відміну від React, який є лише бібліотекою для рендерингу представлень і вимагає самостійного підбору інструментів для маршрутизації, HTTP-запитів чи управління формами, Angular є повноцінним (opinionated) фреймворком. Він пропонує стандартизовану, жорстко структуровану архітектуру «з коробки». Це полегшує підтримку коду та гарантує, що розробник слідує кращим практикам [7].

По-друге, невід’ємною частиною Angular є використання мови TypeScript. Строга типізація даних є критично важливою для застосунків, що працюють зі складними об’єктами (такими як детальні описи книг, вкладені

об’єкти позик та користувачів). TypeScript значно знижує кількість помилок ще на етапі написання коду дозволяє використовувати потужні механізми автодоповнення в IDE та робить рефакторинг безпечним [13].

По-третє, система Dependency Injection (впровадження залежностей) в Angular дозволяє ефективно керувати сервісами (наприклад, сервісом авторизації або HTTP-клієнтом) та створювати слабо зв’язаний код, який легко покривати модульними тестами.

Для забезпечення сучасного, адаптивного дизайну, що коректно працює на мобільних телефонах, планшетах та десктопах, використано бібліотеку Angular Material. Вона надає набір готових, ретельно протестованих UI-компонентів, розроблених за специфікацією Material Design від Google. Це економить час на верстку складних елементів (таких як таблиці з пагінацією та сортуванням, модальні діалогові вікна, форми з валідацією) і гарантує високу доступність (accessibility) інтерфейсу [14].

1.3.2 Вибір платформи для серверної частини

Серверна частина (бекенд) є ядром системи, що відповідає за бізнес-логіку, валідацію даних, безпеку та взаємодію з базою даних. Розглядалися такі технології: Java (Spring Boot), Python (Django/FastAPI), PHP (Laravel) та Node.js (Express/NestJS).

Node.js було обрано як платформу для бекенду. Головною перевагою є можливість використання єдиної мови (JavaScript/TypeScript) у всьому стеку (Full-Stack). Це дозволяє розробнику не перемикати контекст між різними синтаксисами, а також створювати спільні типи даних (інтерфейси) для фронтенду і бекенду, гарантуючи їхню консистентність [8].

Архітектурно Node.js базується на рушієві V8 (розробленому Google) та використовує подієво-орієнтовану, асинхронну модель неблокуючого введення-виведення (Event Loop). Це робить Node.js надзвичайно ефективним

для I/O-bound застосунків (застосунків, де основним вузьким місцем є операції читання/запису до бази даних або мережі, а не складні математичні обчислення). Бібліотечна система, яка в основному обробляє запити на отримання списків книг, збереження нових позик або оновлення статусів, ідеально підходить під цей профіль.

У якості вебфреймворку використано Express.js. Це мінімалістичний, гнучкий інструмент, що став де-факто стандартом для розробки REST API на Node.js. Express надає зручну систему маршрутизації та механізм «middleware» (проміжного програмного забезпечення) [12]. Через middleware дуже зручно реалізовувати наскрізні функції (cross-cutting concerns): перевірку JWT токенів для автентифікації, валідацію вхідних JSON-тіл запитів, обробку помилок та логування (журнал аудиту). Для криптографічного захисту паролів користувачів задіяно алгоритм bcrypt, що робить неможливим відновлення паролів навіть у разі компрометації бази даних [19].

1.3.3 Вибір системи управління базами даних

Бібліотечна система потребує надійного сховища даних. Вибір стояв між традиційними реляційними СУБД (PostgreSQL, MySQL) та документоорієнтованими NoSQL рішеннями (MongoDB). В таблиці 1.3 наведено відповідне порівняння.

Таблиця 1.3 – Порівняльний аналіз MongoDB та PostgreSQL

Критерій	**MongoDB (NoSQL)**	**PostgreSQL (SQL)**
Модель даних	Колекції BSON-документів	Таблиці з жорсткою схемою
Зміна структури	Динамічна, без простоїв	Потребує міграцій (ALTER TABLE)
Зв’язки між сутностями	Вбудовування (Embedding) або посилання (References)	Зовнішні ключі (Foreign Keys), JOIN
Горизонтальне масштабування	Вбудований sharding	Складне

Продовження таблиці 1.3

Критерій	**MongoDB (NoSQL)**	**PostgreSQL (SQL)**
Ідеальне застосування	Швидка розробка, ієрархічні дані	Складні транзакції, фінанси

MongoDB обрано як СУБД, оскільки бібліотечні сутності природно лягають на документоорієнтовану модель [9]. BSON (Binary JSON) формат документів у MongoDB дозволяє зберігати дані саме в тому вигляді, в якому їх використовує клієнтський та серверний JavaScript. Наприклад, масив відгуків до книги або історію її видань можна зберігати безпосередньо у документі книги (вбудовування), що значно пришвидшує операції читання порівняно зі складними SQL JOIN-запитами у реляційних базах.

Крім того, гнучка схема (schema-less) MongoDB полегшує подальший розвиток системи. Якщо в майбутньому виникне необхідність додати до книг специфічні поля (наприклад, посилання на аудіоверсію або особливі примітки для рідкісних видань), це не вимагатиме зупинки бази даних для проведення міграцій схеми.

Для забезпечення цілісності даних на рівні застосунку використовується бібліотека Mongoose (Object Data Modeling). Вона дозволяє описати суворі схеми даних (Schema) у коді бекенду, задати типи полів, встановити правила валідації (обов’язковість, регулярні вирази, унікальність) та використовувати pre/post хуки. Завдяки Mongoose, ми отримуємо переваги гнучкості NoSQL та надійності структурованих даних [11].

1.3.4 Вибір інструментів контейнеризації та розгортання

Однією з найважливіших нефункціональних вимог до сучасної системи є простота її розгортання та відтворюваність середовища. Традиційний підхід, коли адміністратору бібліотеки потрібно вручну встановлювати Node.js, певну

версію MongoDB, налаштовувати змінні середовища та вебсервер, є абсолютно неприйнятним. Це призводить до класичної проблеми «на моєму комп’ютері це працює, а на сервері – ні».

Для вирішення цієї проблеми застосовано технології контейнеризації – Docker. Docker дозволяє запакувати кожен компонент системи (Angular-фронтенд з Nginx, Node.js бекенд, MongoDB) в окремий, ізольований контейнер. Контейнер містить усе необхідне для роботи: операційну систему, бібліотеки, залежності та безпосередньо скомпільований код. Завдяки цьому гарантується, що система працюватиме абсолютно ідентично на машині розробника під Windows, на тестовому сервері під Ubuntu або у хмарному середовищі AWS [10].

Для оркестрації цих взаємопов’язаних контейнерів використовується Docker Compose. Конфігураційний файл docker-compose.yml описує всі сервіси, їхні мережеві взаємодії, змінні середовища (паролі, ключі JWT) та томи (volumes) для постійного зберігання даних бази. Розгортання всієї інфраструктури бібліотеки на новому сервері зводиться до виконання однієї єдиної команди.

Важливо відзначити, що використання Docker створює міцний фундамент для майбутнього масштабування: у разі зростання навантаження, систему можна легко перенести у кластер Kubernetes, використовуючи такі інструменти як Minikube для локального тестування оркестрації.

1.4 Постановка задачі, формування вимог та критеріїв якості системи

На основі детального аналізу предметної області, виявлених недоліків паперового обліку та можливостей обраного технологічного стеку сформульовано технічне завдання на розроблення вебдодатку для ведення обліку громадської бібліотеки. Система проєктується з урахуванням сучасних стандартів архітектури програмного забезпечення та тестування.

1.4.1 Функціональні вимоги

Функціональні вимоги описують, що саме повинна робити система, визначаючи поведінку застосунку під час взаємодії з різними ролями користувачів.

У системі виділено три рівні доступу (ролі): «Адміністратор» (повний контроль, налаштування системи, управління персоналом), «Бібліотекар» (операційна діяльність, робота з фондом і читачами), «Читач» (пошук, перегляд історії, бронювання).

Для неавторизованих відвідувачів («Гостей») доступний лише публічний каталог.

Основні функціональні вимоги наведено на рисунку 1.1.

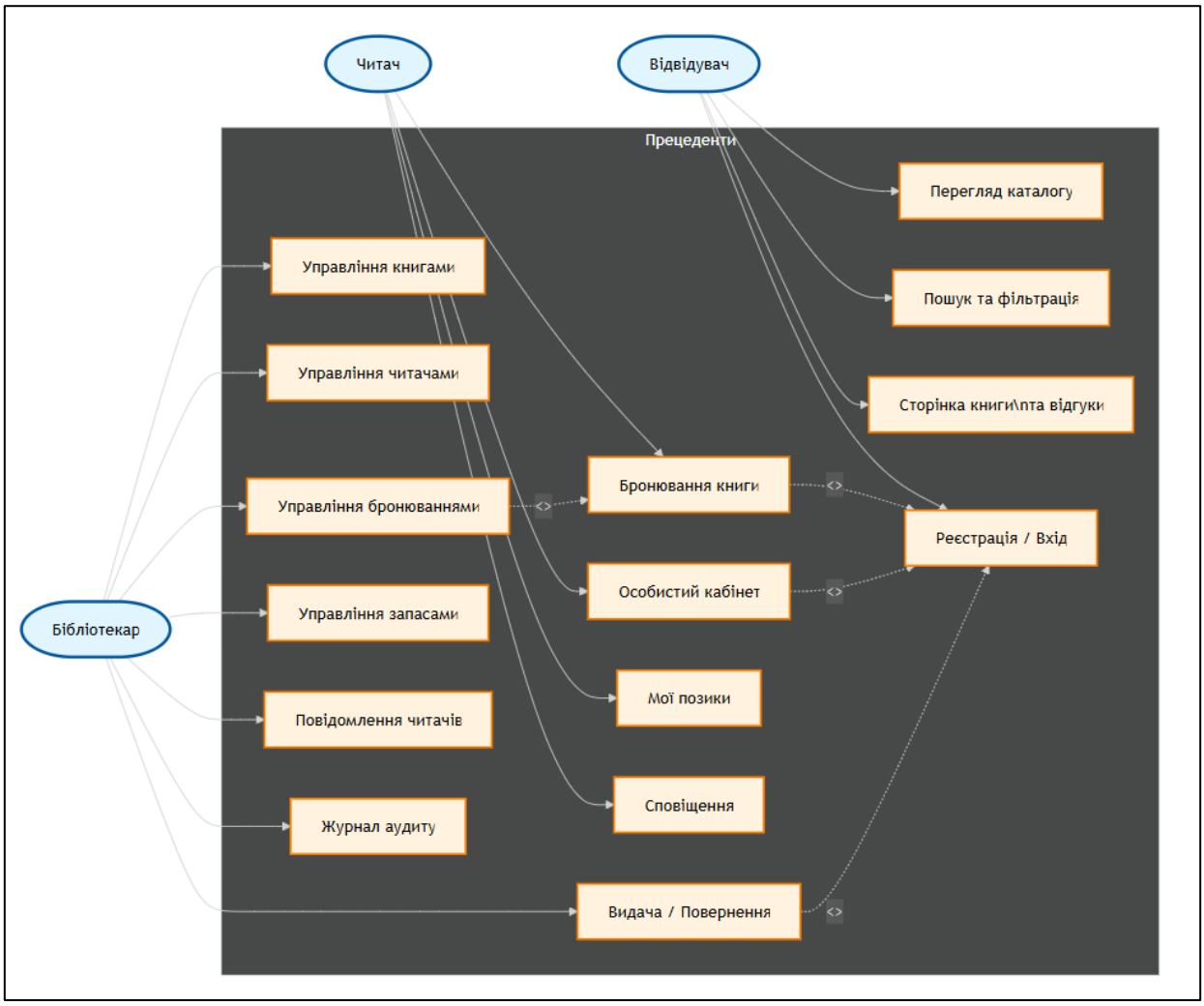


Рисунок 1.1 – Діаграма прецедентів

1.4.2 Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості системи, такі як продуктивність, безпека, надійність та зручність використання. Вони гарантують, що додаток буде не лише виконувати свої функції, але й робити це ефективно та безпечно у реальному середовищі експлуатації.

Продуктивність: архітектура SPA на базі Angular забезпечує миттєву зміну сторінок після початкового завантаження застосунку. Час відгуку серверного REST API на стандартні запити не повинен перевищувати 300-500 мілісекунд при нормальному навантаженні. Пошукові запити до бази даних MongoDB повинні бути оптимізовані за допомогою відповідних індексів на полях назв та авторів.

Безпека: система повинна відповідати базовим вимогам безпеки вебдодатків (OWASP Top 10). Усі паролі в базі даних зберігаються виключно у вигляді хешів (використовується алгоритм bcrypt з автоматичною генерацією ‘солі’). Передача даних між клієнтом і сервером у продакшн-середовищі має здійснюватися лише по захищеному протоколу HTTPS. Захист від міжсайтової підробки запитів (CSRF) та атак типу XSS забезпечується вбудованими механізмами фреймворку Angular. Авторизація реалізована за допомогою короткострокових JWT токенів.

Зручність використання та адаптивність: інтерфейс користувача має бути інтуїтивно зрозумілим. Використання Angular Material гарантує відповідність стандартам доступності. Застосунок є повністю адаптивним (Responsive Design) – верстка автоматично перелаштовується під роздільну здатність екрану, забезпечуючи комфортну роботу на десктопних комп’ютерах бібліотекарів, планшетах та мобільних смартфонах читачів.

Надійність та відновлюваність: контейнеризація Docker забезпечує ізоляцію процесів. Дані бази MongoDB зберігаються на постійному томі Docker (Docker Volume), що гарантує їх збереження навіть при видаленні чи оновленні контейнера бази даних. У разі критичної помилки в роботі Node.js сервера, менеджер контейнерів автоматично перезапустить сервіс.

1.4.3 Вимоги до забезпечення якості та тестування

Для забезпечення високої якості програмного продукту, процес розроблення супроводжується застосуванням сучасних методологій тестування програмного забезпечення. Організація процесу тестування ґрунтується на принципах міжнародного стандарту IEEE 829, що передбачає структурований підхід до складання тестової документації, зокрема розробку комплексних тест-планів та деталізованих звітів про дефекти (bug reports).

Враховуючи вимоги до кросплатформеності та адаптивності, обов’язковим етапом перевірки якості є проведення кросбраузерного тестування. Для цього складаються спеціальні чек-листи (cross-browser checklists), які охоплюють перевірку коректного відображення та функціонування застосунку в основних сучасних веббраузерах (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) як у десктопних, так і в мобільних версіях. Особлива увага під час тестування інтерфейсу приділяється перевірці коректності роботи інтерактивних UI-компонентів, розроблених за допомогою підходів компонентно-орієнтованої архітектури.

1.5 Висновки з розділу 1

Отже, у першому розділі було виконано комплексний аналіз предметної області громадської бібліотеки. Доведено, що перехід від паперового документообігу до цифрових систем обліку є критично необхідним для підвищення якості обслуговування. Детальний огляд існуючих АБІС виявив прогалину на ринку рішень для невеликих бібліотек, що обґрунтовує доцільність розробки легкого, адаптивного та простого у розгортанні вебдодатку.

Проведений аналіз сучасних технологій дозволив сформувати оптимальний стек: Angular для клієнтської частини, Node.js та Express для

швидкого бекенду, документоорієнтована MongoDB для гнучкого зберігання даних, та Docker для надійної контейнеризації. Визначено вичерпний перелік функціональних та нефункціональних вимог, а також встановлено стандарти якості та тестування системи, що є надійною основою для переходу до етапу проєктування архітектури застосунку.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У другому розділі кваліфікаційної роботи здійснюється комплексне, глибоке та системне проєктування архітектурних рішень, компонентної моделі, інформаційного та програмного забезпечення вебдодатку для автоматизації процесів обліку громадської бібліотеки. Проєктування сучасних розподілених систем є визначальним і найбільш критичним етапом життєвого циклу розробки програмного забезпечення. Це зумовлено тим, що закладені на даній стадії структурні та технологічні патерни безпосередньо впливають на показники масштабованості, відмовостійкості, інформаційної безпеки, швидкодії та гнучкості майбутнього програмного продукту, а також визначають логічні та фізичні межі взаємодії між клієнтським програмним кодом, сервером застосунків та базою даних.

Основним об’єктом автоматизації є центральна бібліотека, специфіка діяльності якої вимагає створення інтегрованого середовища для ефективного управління книжковим фондом та взаємодії з користувачами. На основі детального аналізу предметної області та технічних вимог, обґрунтованих у першому розділі роботи, було розроблено концептуальну схему інформаційної системи. Дана схема базується на принципах розмежування прав доступу для різних категорій користувачів, таких як адміністратори системи, бібліотекарі та зареєстровані читачі. Крім того, ключова увага приділена забезпеченню прозорого п’ятикрокового життєвого циклу електронного бронювання примірників та впровадженню інтелектуальних підсистем нечіткого повнотекстового пошуку літератури, стійкого до синтаксичних та друкарських помилок операторів.

Системний підхід до проєктування інформаційного середовища громадської бібліотеки вимагає декомпозиції загальної структури на декілька взаємопов’язаних підсистем. Зокрема, підсистема автентифікації та авторизації повинна гарантувати безпеку даних шляхом застосування сучасних криптографічних протоколів та токенів. Підсистема обліку та

каталожного пошуку має забезпечувати відповідний мінімальний час відгуку при виконання складних фільтрацій та агрегацій даних. Підсистема взаємодії з читачами розробляється з урахуванням сучасних вимог до ергономіки користувацьких інтерфейсів, що є критичним для забезпечення високої доступності сервісів бібліотеки для різних вікових груп населення.

2.1 Обґрунтування архітектури системи

Для практичної реалізації вебдодатку для ведення обліку громадської бібліотеки було обрано трирівневу клієнт-серверну архітектуру з повним логічним та фізичним розділенням клієнтської частини та серверної частини на основі архітурного стилю REST API. Такий вибір є повністю обґрунтованим з інженерної точки зору, оскільки класичні монолітні архітектури, де презентаційний шар та бізнес-логіка розгорнуті в межах одного середовища виконання, мають суттєві обмеження щодо горизонтального масштабування, ускладнюють процес паралельної розробки та суттєво збільшують вартість підтримки системи в довгостроковій перспективі.

Клієнтська частина системи функціонує як односторінковий застосунок, який завантажується в браузер кінцевого користувача одноразово під час першого звернення до вебсервера. Уся подальша взаємодія, навігація між сторінками та оновлення графічного інтерфейсу відбуваються динамічно, без повного перезавантаження сторінок, шляхом асинхронного обміну даними з бекендом за допомогою HTTP-запитів. Це дозволяє суттєво знизити навантаження на мережеві канали зв’язку, мінімізувати трафік та забезпечити плавний інтерфейс, що за своїми характеристиками наближається до нативних настільних застосунків.

Серверна частина розроблюваної інформаційної системи проєктується як безстанова архітектура. Це означає, що сервер застосунків не зберігає інформацію про контекст сесії конкретного користувача між окремими HTTP-запитами. Кожен запит, який надходить від клієнтського застосунку Angular,

є повністю самодостатнім і містить у своєму заголовочному полі Authorization спеціальний криптографічно підписаний токен доступу JSON Web Token [18]. Сервер дешифрує та валідує цей токен безпосередньо в процесі обробки запиту, що дозволяє уникнути необхідності постійного звернення до сховищ сесій та забезпечує можливість легкого горизонтального масштабування системи шляхом додавання нових екземплярів сервера за балансувальником навантаження.

Взаємодія між клієнтським Angular-застосунком та серверною частиною на базі Node.js здійснюється за захищеним протоколом HTTPS із використанням стандартних методів HTTP. Як єдиний формат обміну даними обрано структурований текстовий формат JSON, який є легко читаним як для людини, так і для комп'ютера, і має нативну підтримку в обраному технологічному стеку.

Для наочного відображення логічної взаємодії між усіма рівнями системи розроблено загальну структурну схему архітектури (див. рис. 2.1).

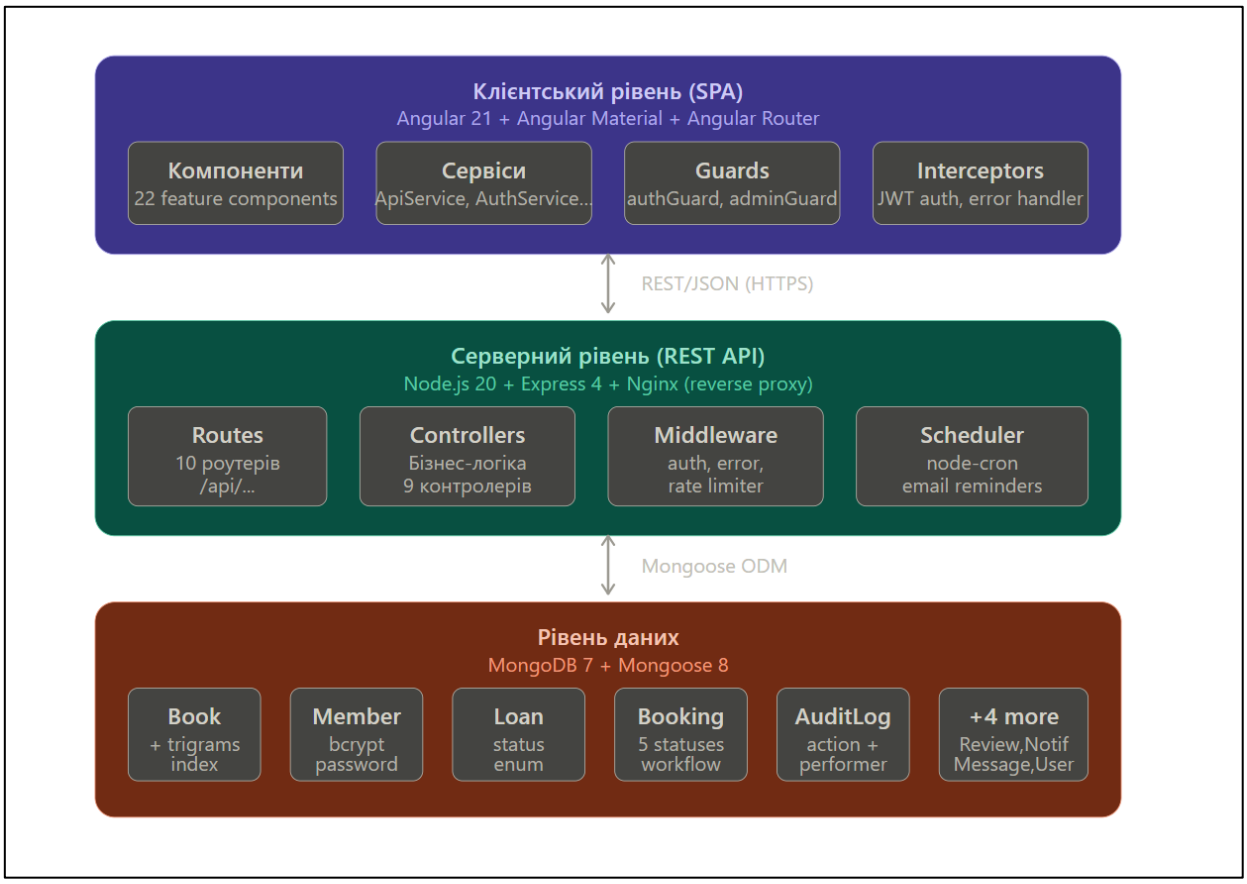


Рисунок 2.1 – Загальна трирівнева архітектура вебдодатку

На наведеній схемі архітектури чітко простежується шлях проходження даних. Запити від користувачів спочатку приймаються високопродуктивним зворотним проксі-сервером Nginx, який відповідає за маршрутизацію трафіку, захист від базових мережевих атак, віддачу статичних файлів клієнтської частини та перенаправлення динамічних API-запитів на внутрішній порт сервера Node.js. Сервер Node.js, використовуючи фреймворк Express, розподіляє запити по відповідних контролерах, виконує необхідні операції бізнес-логіки та взаємодіє з базою даних MongoDB через об’єктно-документне відображення Mongoose ODM [17].

2.2 Структурна схема та модульна організація

Модульна структура вебдодатку для ведення обліку громадської бібліотеки спроєктована на основі фундаментальних принципів об’єктно-орієнтованого проєктування та системної інженерії, а саме: забезпечення високої внутрішньої зв’язності компонентів та низької взаємної зачепленості між окремими модулями. Серверна частина системи, яка функціонує в середовищі Node.js, побудована з чітким розділенням відповідальності за класичним архітектурним шаблоном MVC, адаптованим під реалії розробки сучасних REST API застосунків.

Весь програмний код сервера розподілено за спеціалізованими логічними каталогами на рівні файлової структури проєкту, що дозволяє забезпечити легкість читання коду, спростити процес проведення модульного тестування та автоматизувати розгортання. Папка models/ містить описи схем даних, папка controllers/ відповідає за первинну обробку HTTP-запитів та валідацію вхідних параметрів, а папка services/ виділена під чисту бізнес-логіку системи, що повністю ізольована від особливостей HTTP-протоколу. Для детального ознайомлення з призначенням кожного структурного блоку сервера розроблено специфікацію (див. табл. 2.1).

Таблиця 2.1 – Структура модулів серверної частини (Backend)

Компонент / Папка	**Опис функціонального призначення в архітектурі бекенду**
models/	Визначення схем Mongoose, індексів, зв’язків та тригерів валідації для об’єктів бази даних MongoDB.
controllers/	Приймання HTTP-запитів, валідація вхідних параметрів, виклик сервісів бізнес-логіки, повернення відповідей.
services/	Реалізація бізнес-процесів (алгоритм триграм, обробка замовлень книг, логування дій користувачів, аудит).
middlewares/	Автентифікація користувачів за ролями, перевірка сигнатур JWT, глобальне перехоплення та логування помилок.
routes/	Мапування URL-адрес ендпоінтів на методи відповідних контролерів із застосуванням захисних middleware.
cron/	Конфігурація автоматичних фонових завдань через node-cron (щоденний аналіз термінів та заборгованостей).

Розподіл відповідальності на рівні бекенду дозволяє гарантувати, що зміни в алгоритмах бізнес-логіки, які містяться в каталозі services/ (наприклад, зміна правил розрахунку штрафів чи модифікація алгоритму нечіткого пошуку літератури), жодним чином не вплинуть на код маршрутизації в routes/ чи структури моделей у models/. Це значно підвищує технологічну життєздатність системи та знижує ризик появи регресійних помилок під час тривалої експлуатації вебдодатку в бібліотеці.

Клієнтська частина розроблюваної інформаційної системи базується на передовому фреймворку Angular версії 21. Проєктування архітектури фронтенду виконано з використанням сучасної парадигми Standalone-компонентів, яка дозволяє повністю відмовитися від застарілих та громіздких модулів Angular. Це суттєво спрощує дерево залежностей застосунку та робить структуру проєкту більш прозорою. Завантаження великих функціональних блоків інтерфейсу, таких як панель адміністратора, робоче місце бібліотекаря та особистий кабінет читача, реалізовано за допомогою механізму лінивого

завантаження на рівні Angular Router. Це означає, що користувач завантажує лише той мінімальний обсяг JavaScript-коду, який необхідний для відображення поточної сторінки, що оптимізує швидкість запуску застосунку.

Для управління станом клієнтського застосунку та забезпечення миттєвого реагування графічного інтерфейсу на дії користувача впроваджено інноваційний механізм Angular Signals. Використання сигналів замість класичних асинхронних потоків RxJS дозволяє реалізувати дрібнозернисте оновлення DOM-дерева сторінки [21]. Перерендерування відбуваються виключно для тих елементів інтерфейсу, значення яких безпосередньо змінилися, що мінімізує витрати обчислювальних ресурсів процесора клієнта. Структурна організація та архітектура компонентів фронтенду представлена на рисунку 2.2.

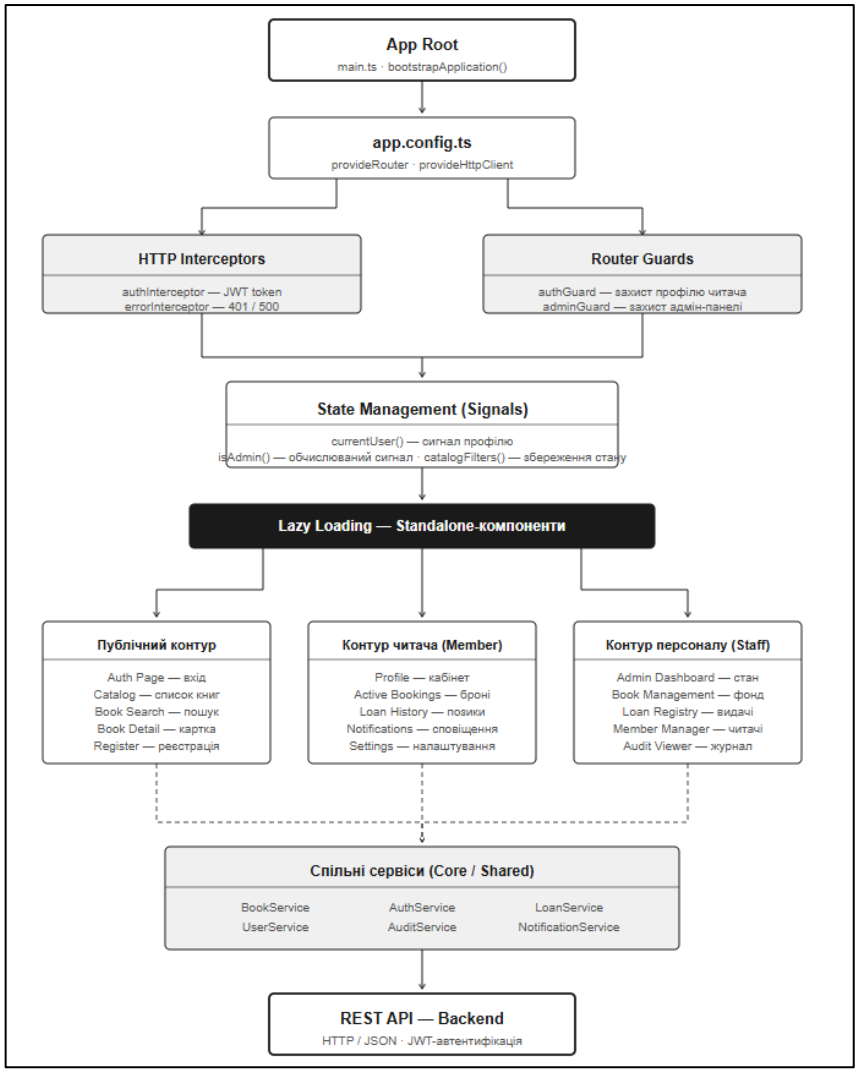


Рисунок 2.2 – Модульна структура Angular застосунку (Frontend)

2.3 Проєктування бази даних

Для забезпечення надійного, гнучкого та високопродуктивного зберігання інформації у вебдодатку для ведення обліку громадської бібліотеки було обрано документоорієнтовану систему керування базами даних MongoDB. Вибір NoSQL рішень замість традиційних реляційних баз даних обумовлений специфікою предметної області бібліотечного обліку. Бібліографічні описи книг та журналів часто мають неструктуровану або слабкоструктуровану природу, містять масиви авторів, динамічні списки жанрів, ключових слів та мінливі метадані, які складно ефективно відобразити у жорсткі реляційні таблиці без створення великої кількості проміжних таблиць зв'язків типу «багато-до-багатьох».

У MongoDB дані зберігаються у вигляді гнучких BSON-документів, що дозволяє природним чином групувати пов'язані дані в межах одного документа, мінімізуючи потребу в коштовних операціях об'єднання. Взаємозв'язки між документами різних колекцій у розроблюваній системі реалізовано за допомогою системних посилань на унікальні ідентифікатори документів ObjectId. Моделювання схем даних здійснюється на рівні сервера за допомогою інструменту Mongoose ODM, який дозволяє декларативно описати типи полів, правила валідації та налаштувати автоматичне розгортання посилань у вкладені об'єкти за допомогою вбудованої високоефективної операції populate() під час виконання аналітичних запитів.

В інфраструктурі бази даних було спроєктовано оптимальний набір з 9 взаємопов'язаних колекцій: User (облікові дані персоналу бібліотеки), Member (профілі зареєстрованих читачів), Author (довідник авторів літератури), Genre (класифікатор жанрів), Book (метадані та бібліографічні описи видань), BookItem (фізичні примірники книг з унікальними інвентарними кодами та штрих-кодами), Booking (заявки на попереднє бронювання), Loan (фактичні видачі книг на руки) та AuditLog (системний журнал аудиту дій користувачів для забезпечення безпеки).

Логічна структура, типи зв’язків та ER-модель NoSQL сховища спроєктовані та представлені на схемі зв’язків бази даних (рис. 2.3).

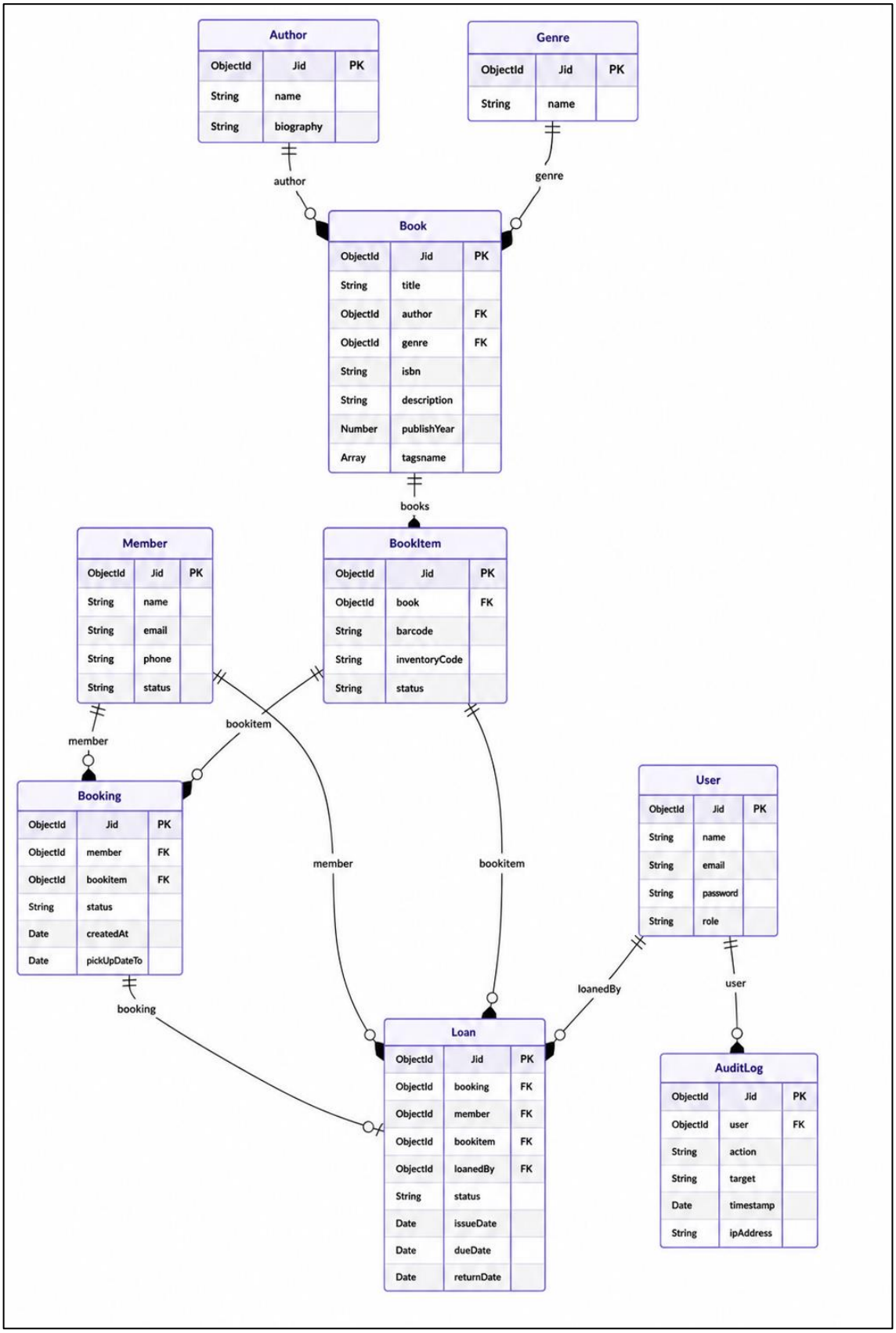


Рисунок 2.3 – Схема зв’язків та ER-структура бази даних MongoDB

Як центральний і найбільш показовий приклад інформаційного забезпечення системи, у таблиці 2.2 наведено детальну специфікацію полів колекції документів Book, яка є стрижневою для формування електронного каталогу бібліотеки. Дана схема містить унікальний індекс за полем isbn для запобігання дублюванню описів видань, а також оптимізований масив рядків trigrams. Масив триграм генерується автоматично через проміжне програмне забезпечення Mongoose безпосередньо перед збереженням документа в базу даних, що є основою для реалізації алгоритму швидкого нечіткого пошуку.

Таблиця 2.2 – Структура колекції документів «Book» (каталог книг)

Назва поля	**Тип даних**	**Обов'язково?**	**Опис призначення та обмеження поля схеми Mongoose**
_id	ObjectId	Так	Унікальний системний ідентифікатор документа в MongoDB за замовчуванням.
title	String	Так	Офіційна назва літературного твору або видання.
author	ObjectId	Так	Посилання на ідентифікатор документа з колекції авторів (ref: 'Author').
genre	ObjectId	Так	Посилання на ідентифікатор документа з колекції жанрів (ref: 'Genre').
isbn	String	Ні	Міжнародний стандартний книжковий номер (унікальний індекс за наявності).
description	String	Ні	Текстова анотація, опис змісту та ключових характеристик видання.
publishYear	Number	Ні	Рік офіційного видання або друку книги.
trigrams	[String]	Так	Масив згенерованих лінгвістичних триграм для розмитого повнотекстового пошуку.

2.4 Проєктування REST API

Усі операції обміну даними між клієнтським Angular-застосунком та сервером Express повністю уніфіковано в межах архітектурного стилю REST API. Кожен ендпоінт системи проєктується з дотриманням семантики методів HTTP та повертає відповідні стандартні коди статусів (наприклад, 200 OK – успішне виконання, 201 Created – успішне створення сутності, 400 Bad Request

– помилка валідації вхідних даних, 401 Unauthorized – відсутність або невалідність токена автентифікації, 403 Forbidden – недостатньо прав доступу для поточної ролі користувача, 404 Not Found – запитуваний ресурс не знайдено в базі даних).

Захист приватних маршрутів забезпечується за допомогою спеціалізованого проміжного програмного забезпечення Auth Middleware. Це middleware перехоплює вхідний HTTP-запит, виділяє рядок токена з заголовка, перевіряє його криптографічний підпис за допомогою секретного ключа сервера, декодує корисне навантаження із зазначенням ідентифікатора та ролі користувача, та динамічно записує ці дані в об’єкт запиту req.user. Якщо роль користувача не відповідає рівню доступу ендпоінту, запит негайно відхиляється. Для повного покриття бізнес-вимог було розроблено вичерпну карту маршрутів REST API інформаційної системи (див. табл. 2.3).

Таблиця 2.3 – Специфікація маршрутів REST API

Метод	**Ендпоінт REST API**	**Доступ**	**Функціональне призначення маршруту в системі**
POST	/api/auth/login	Публічний	Автентифікація персоналу (admin/librarian). Повертає JWT токен.
POST	/api/auth/member/login	Публічний	Автентифікація читачів для входу в особистий кабінет.
GET	/api/books	Публічний	Отримання списку книг з підтримкою розмитого пошуку за триграмами.
POST	/api/books	Admin	Додавання нової книги до каталогу та автоматична генерація триграм.
PUT	/api/books/:id	Admin	Редагування інформації про книгу та оновлення пошукових індексів.
DELETE	/api/books/:id	Admin	Видалення книги з каталогу (можливо, якщо немає активних позик).
GET	/api/books/:id/items	Staff	Перегляд усіх фізичних екземплярів (BookItems) конкретної книги.
POST	/api/books/:id/items	Admin	Додавання нового екземпляра книги з унікальним штрих-кодом.
GET	/api/bookings	Staff	Перегляд усіх замовлень книг у системі (для бібліотекарів).
POST	/api/bookings	Member	Створення нового бронювання книги авторизованим читачем.

Продовження таблиці 2.3

Метод	**Ендпоінт REST API**	**Доступ**	**Функціональне призначення маршруту в системі**
PATCH	/api/bookings/:id/status	Staff	Зміна статусу бронювання в межах 5-крокового життєвого циклу.
GET	/api/loans	Staff	Отримання списку всіх активних та збережених позик літератури.
POST	/api/loans	Staff	Оформлення видачі книги на руки (автоматично при picked_up або вручну).
POST	/api/loans/:id/return	Staff	Фіксація повернення книги, зміна статусу BookItem на available.
GET	/api/admin/audit	Admin	Перегляд логів аудиту безпеки (доступно лише адміністратору системи).

2.5 Проєктування інтерфейсу користувача та взаємодії

Графічний інтерфейс користувача розроблюваної інформаційної системи спроєктовано на основі компонентної моделі бібліотеки Angular Material із суворим дотриманням сучасних принципів адаптивного дизайну. Це гарантує однаково коректне, зручне та візуально привабливе відображення елементів керування як на моніторах стаціонарних робочих станцій персоналу бібліотеки, так і на дисплеях планшетів чи мобільних пристроїв читачів, які користуються електронним каталогом дистанційно поза межами установи.

В основі проєктування інтерфейсу покладено концепцію мінімізації дій користувача для досягнення цілі (принцип трьох кліків). Для цього навігаційну систему розділено на три ізольованих робочих простори: публічний контур (пошук, перегляд деталей книги, реєстрація), кабінет читача (перегляд власних активних бронювань, історії позик та отримання повідомлень про необхідність повернення книг) та внутрішній контур персоналу (керування фондом, видача книг, зчитування штрих-кодів, перегляд системних логів). Логіку переходів та ієрархічну карту екранів представлено на відповідній схемі навігації застосунку (див. рис. 2.4).

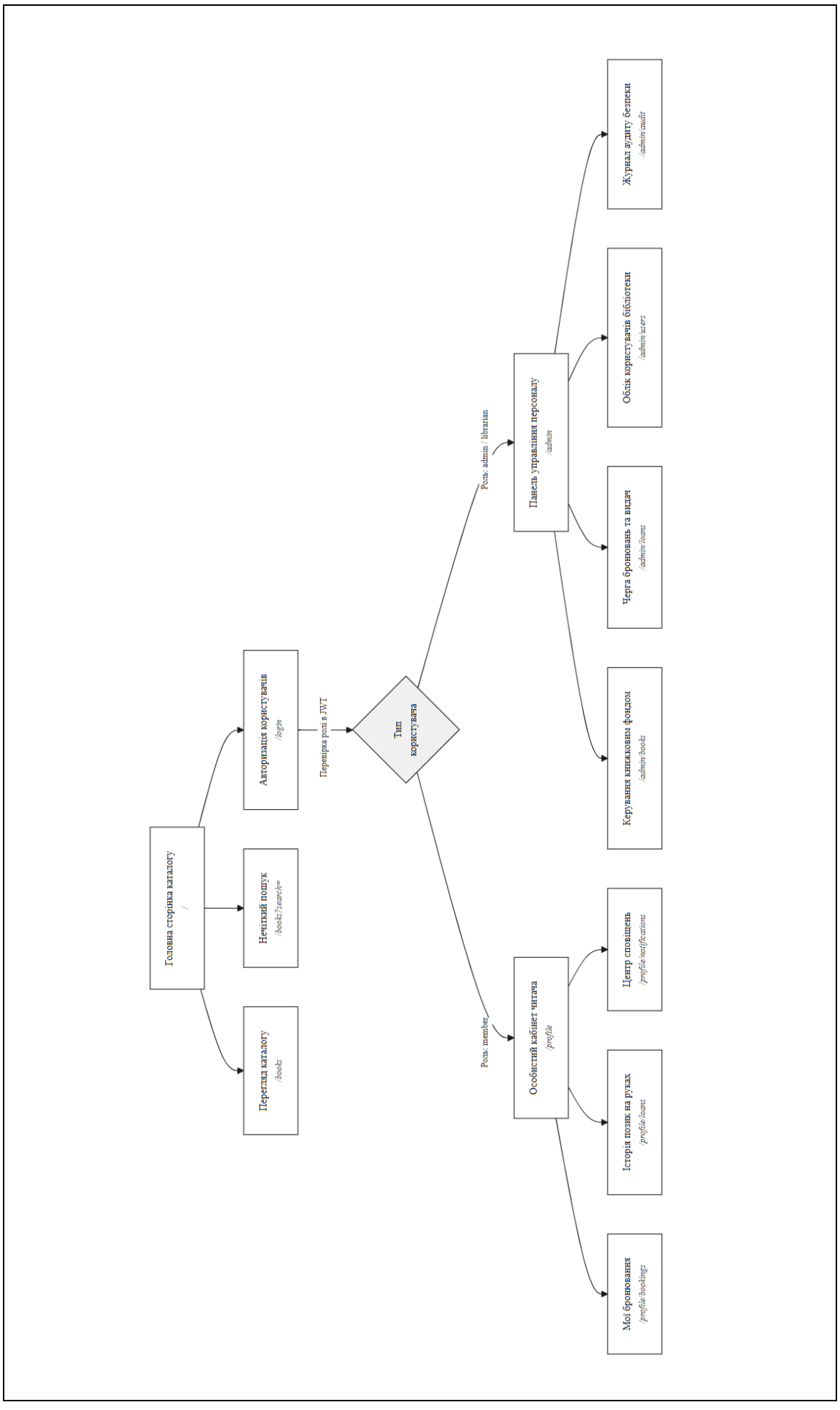


Рисунок 2.4 – Карта навігації та структура сторінок вебдодатку

2.6 Специфікація бізнес-логіки та життєвих циклів

Ключовим та найбільш складним бізнес-процесом розроблюваної інформаційної системи є автоматизоване обслуговування черги замовлень, контроль термінів користування книгами та запобігання втратам книжкового фонду. Для забезпечення високої надійності та виключення логічних конфліктів було спроєктовано строгу математичну модель кінцевого автомата. Дана модель чітко визначає дозволені стани та правила переходів для документа бронювання та зв’язаного з ним примірника.

Модель життєвого циклу бронювання складається з п’яти послідовних станів: pending (заявка створена читачем через сайт, фізичний примірник книги автоматично маркується в базі даних як booked і вилучається з загального пошуку), approved (бібліотекар підтвердив наявність книги на полиці та відклав її, система розраховує дату дедлайну отримання тривалістю 3 дні), rejected (заявку відхилено через пошкодження примірника, статус книги повертається в available), picked_up (читач прибув до бібліотеки та забрав книгу, бронювання закривається, і автоматично генерується новий документ позики зі строком повернення до 30 днів), expired (читач не з’явився протягом 3 днів, броня автоматично скасовується). Графічне представлення кінцевого автомата наведено на діаграмі станів (див. рис. 2.5).

Контроль за дотриманням часових обмежень та автоматичне переведення протермінованих документів у стани expired або overdue покладено на фоновий сервіс-плановувальник на базі модуля node-cron [20]. Цей сервіс запускається безпосередньо на сервері щоденно рівно о 08:00 за київським часом, виконує агрегаційні запити до бази даних MongoDB, виявляє порушників та автоматично змінює статус профілю читача в колекції Member на suspended.

Блокування діє в автоматичному режимі та обмежує можливість дистанційного замовлення нових книг до моменту повернення боргу бібліотеці.

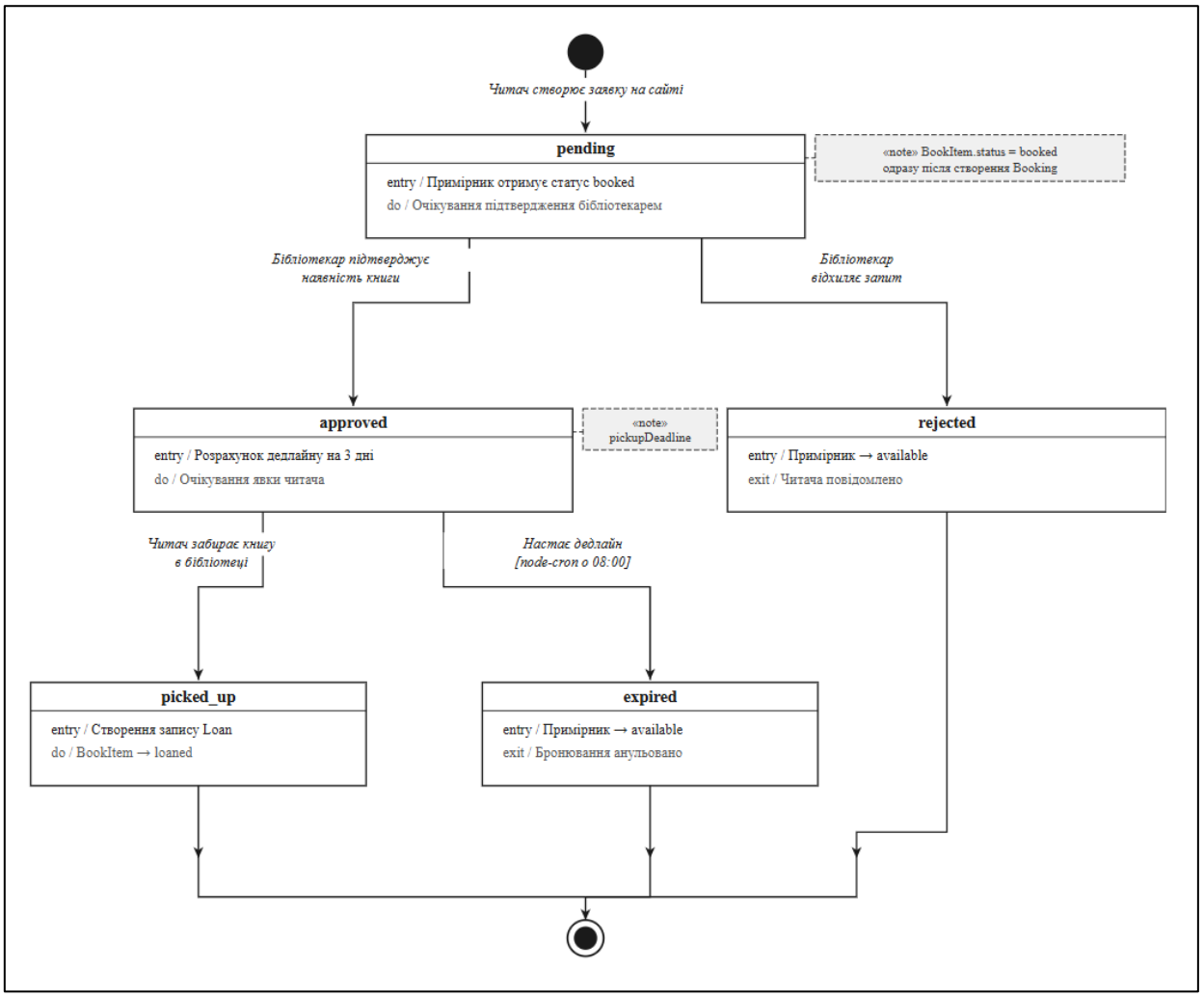


Рисунок 2.5 – Діаграма станів життєвого циклу бронювання книги

2.7 Висновки з розділу 2

У ході виконання другого розділу кваліфікаційної роботи було детально спроєктовано архітектуру, модульну структуру, інформаційне та програмне забезпечення вебдодатку для ведення обліку громадської бібліотеки. На основі проведенного аналізу обґрунтовано вибір трирівневої безстанової архітектури, що взаємодіє через REST API з використанням даних у форматі JSON, що гарантує високу продуктивність та масштабованість.

Спроєктовано логічну структуру бази даних MongoDB, яка складається з 9 оптимізованих колекцій, та наведено вичерпну специфікацію полів центральної колекції Book, включаючи механізм підтримки лінгвістичних

триграм. Розроблено та задокументовано карту маршрутів REST API з розподілом рівнів доступу за допомогою токенів.

Специфіковано модульну організацію Angular-застосунку на базі standalone-компонентів та сигналів управління станом, а також детально розписано логіку функціонування державної машини життєвого циклу бронювання, яка контролюється автоматичним планувальником завдань node-cron.

Усі сформовані архітектурні рішення, проєктні специфікації та структурні схеми повністю відповідають критеріям якості, вимогам методичних вказівка і є завершеною, цілісною теоретичною основою для практичної реалізації та тестування системи у наступному розділі роботи.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Обґрунтування та опис інструментів розробки

3.1.1 Інструменти серверної частини

Для програмної реалізації серверної частини вебдодатку для ведення обліку громадської бібліотеки було обрано програмну платформу Node.js. Дане рішення базується на потребі забезпечення високої швидкодії при обробці великої кількості асинхронних вхідних запитів. Node.js функціонує на базі високоефективного рушія компіляції V8 та використовує подійно-орієнтовану модель проєктування з неблокуючим введенням-виведенням даних. Це дозволяє уникнути значних витрат оперативної пам’яті сервера на створення окремих потоків для кожного користувача, що є характерним для класичних серверних платформ [8].

У ролі базового каркаса для створення архітектури розподілених інтерфейсів програмування додатків було застосовано мінімалістичний фреймворк Express.js. Він надає розробнику гнучкі механізми маршрутизації вхідних викликів, інтеграції проміжних шарів обробки даних та стандартизації відповідей сервера [12].

Для організації взаємодії з документоорієнтованою базою даних MongoDB використано спеціалізовану бібліотеку об’єктно-документного моделювання Mongoose [11].

Mongoose надає можливість декларативного опису строгих схем валідації для документів NoSQL сховища безпосередньо в коді JavaScript. Це дозволяє перенести логіку контролю цілісності інформації на рівень сервера застосунків, забезпечуючи типізацію полів, налаштування унікальних індексів, автоматичне каскадне видалення пов’язаних об’єктів та динамічне розгортання референсних зв’язків між колекціями.

3.1.2 Інструменти клієнтської частини

Розробка клієнтської частини односторінкового застосунку виконана на базі компонентного фреймворку Angular версії 21. Вибір даної технології зумовлений її високою структурованістю та наявністю інструментів для побудови масштабованих інтерфейсів. Проєкт реалізовано з використанням архітектурного підходу Standalone-компонентів, що дозволило повністю відмовитися від традиційних модулів фреймворку. Такий підхід значно спростив дерево залежностей застосунку, зменшив фінальний розмір програмного бандлу та прискорив час його ініціалізації в браузері [7].

Для ефективного управління реактивним станом інтерфейсу та організації безшовного оновлення відображення даних впроваджено механізм Angular Signals. Використання сигналів забезпечує точкове відстеження змін у моделях даних та оптимізує перемальовування виключно тих елементів сторінки, значення яких зазнали модифікації. Це дозволило відмовитися від важких асинхронних потоків та покращити загальну продуктивність вебдодатку на пристроях читачів.

Візуальне оформлення графічного інтерфейсу базується на використанні сучасної бібліотеки готових елементів Angular Material. Компоненти даної бібліотеки спроєктовані відповідно до сучасних стандартів ергономіки вебдодатків, мають вбудовану підтримку адаптивності для мобільних екранів та спрощують створення уніфікованих навігаційних панелей, модальних діалогових вікон і таблиць [17]. Для написання стилів застосовано препроцесор SCSS, який розширює можливості мови стилізації за допомогою впровадження змінних, вкладених правил та міксинів.

3.1.3 Допоміжні інструменти розробки

Основним інтегрованим середовищем розробки для написання, відлагодження та супроводу вихідного коду як серверної, так і клієнтської

частин інформаційної системи було обрано редактор Visual Studio Code. Дане програмне забезпечення має високу швидкість роботи, низькі вимоги до апаратних ресурсів комп'ютера та підтримує розширену інтеграцію з інструментами статичного аналізу коду, системами контролю версій та терміналами виконання команд.

Для забезпечення ізоляції середовища запуска, автоматизації розгортання та виключення проблем сумісності версій залежностей на різних серверах використано платформу контейнеризації Docker. Усі компоненти системи розбито на окремі ізольовані контейнери, управління якими координується через утиліту Docker Compose. Капсуляція бази даних, коду Express та клієнтської збірки дозволяє розгортати систему на будь-якому цільовому сервері за допомогою однієї консольної команди.

Як високопродуктивний вебсервер та зворотний проксі-сервер у системі застосовано Nginx. Його конфігурація забезпечує ефективний розподіл мережевого трафіку, захист внутрішнього порту Node.js від запитів із зовнішньої мережі, безпечну обробку заголовків та коректну маршрутизацію запитів усередині односторінкового застосунку.

3.2 Програмна реалізація та архітектура класів системи

3.2.1 Структура та реалізація серверної частини

Програмний код серверної частини розгалужується на ізольовані шари, де кожен модуль виконує строго визначену архітектурну функцію. На рисунку 3.1 наведено базову деревоподібну структуру каталогів розробленого бекенд-проєкту, яка демонструє фізичний розподіл файлів вихідного коду.

Колекції документів бази даних MongoDB конфігуруються за допомогою схем моделювання Mongoose у каталозі models/. Центральною сутністю системи є модель Book, яка описує метадані літературних творів.

Вона містить текстові поля назви, опису, унікального індексу міжнародного номера, а також референсні зв’язки типу ObjectId на документи авторів та жанрів. Для забезпечення швидкого пошуку схема містить спеціальний індексований масив рядків trigrams.

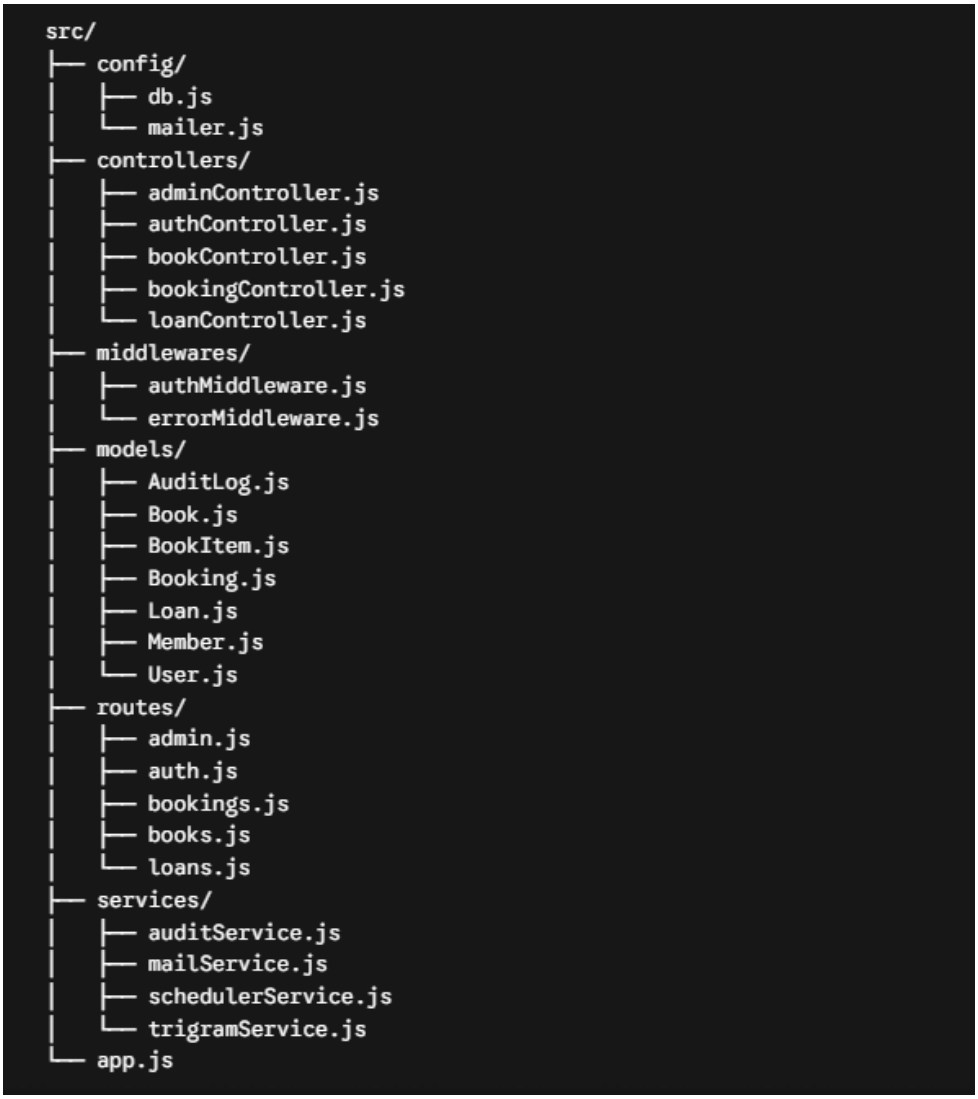


Рисунок 3.1 – Деревоподібна структура каталогів серверної частини

Для управління бізнес-процесами розроблено ряд фонових сервісів у каталозі services/:

– auditService.js – перехоплює ключові модифікації даних та створює незмінні записи дій персоналу в колекції логів;
– trigramService.js – реалізує алгоритм розбиття текстових рядків на лінгвістичні триграми для повнотекстового нечіткого пошуку;

– schedulerService.js – використовує планувальник задач для щоденного автоматичного аналізу термінів утримання броней та формування списків боржників;
– mailService.js – здійснює формування та розсилку електронних листів сповіщень.

Обробка HTTP-запитів координується у файлах каталогу controllers/. Згідно з вимогами до наочності програмних рішень, на рисунку 3.2 наведено реальний фрагмент вихідного коду для ключового сервісу розмитого пошуку та відповідного методу контролера, який здійснює вибірку книг з урахуванням референсних зв’язків.

```
      // Фрагмент файлу src/services/trigramService.js
export const generateTrigrams = (text) => {
  if (!text) return [];
  const normalized = text.toLowerCase().trim().replace(/[^a-za-яєіїґ0-9\s]/g, '');
  const trigrams = new Set();
  for (let i = 0; i <= normalized.length - 3; i++) {
    trigrams.add(normalized.substring(i, i + 3));
  }
  return Array.from(trigrams);
};

// Фрагмент файлу src/controllers/bookController.js
import Book from '../models/Book.js';
import { generateTrigrams } from '../services/trigramService.js';

export const getBooks = async (req, res) => {
  try {
    const { search, genre, author } = req.query;
    let query = {};

    if (genre) query.genre = genre;
    if (author) query.author = author;

    if (search) {
      const searchTrigrams = generateTrigrams(search);
      if (searchTrigrams.length > 0) {
        query.trigrams = { $in: searchTrigrams };
      }
    }

    const books = await Book.find(query)
      .populate('author')
      .populate('genre')
      .exec();

    return res.status(200).json({ success: true, data: books });
  } catch (error) {
    return res.status(500).json({ success: false, message: error.message });
  }
};
```

Рисунок 3.2 – Програмна реалізація пошукової логіки на стороні сервера

В таблиці 3.1 наведено специфікацію методів обробки пошукових запитів каталогу книг.

Таблиця 3.1 – Специфікація методів

Назва файлу та модуля	**Опис методу**	**Вхідні параметри**	**Результат виконання**
trigramService.js	generateTrigrams	Текстовий рядок	Масив унікальних триграм
bookController.js	getBooks	Об'єкт запиту з query	Масив документів із populate

3.2.2 Структурна схема та реалізація клієнтської частини

Клієнтський застосунок на базі Angular 21 організовано за принципом модульного розподілу функціональних контурів на standalone-компоненти. На рисунку 3.3 наведено структуру основних директорій фронтенд-проєкту.



Рисунок 3.3 – деревоподібна структура каталогів серверної частини

Каталог core/ містить загальносистемні елементи, які ініціалізуються один раз при старті системи. Рівень управління станом представлено сервісом state.service.ts, який використовує механізм сигналів для збереження профілю автентифікованого користувача та поточних параметрів фільтрації каталогу. Безпека переходів між сторінками контролюється файлами функціональних захисників у папці guards/.

Взаємодія з сервером за протоколом HTTP координується через два спеціалізовані перехоплювачі:

– auth.interceptor.ts – автоматично виділяє збережений токен доступу з локального сховища браузера та додає його в заголовок Authorization кожного вихідного запиту;
– error.interceptor.ts – здійснює глобальний моніторинг вхідних відповідей сервера.

У разі отримання статусів помилок авторизації сервіс автоматично очищує локальні сигнали стану та перенаправляє інтерфейс читача на сторінку входу.

Для демонстрації взаємодії фронтенду з REST API, у таблиці 3.2 наведено специфікації компонентів шару отримання даних та реальний програмний код Angular-сервісу, що працює з реактивними сигналами.

Таблиця 3.2 – Специфікація сервісів клієнтської взаємодії з бібліографічним каталогом

Назва файлу	**Опис сервісного класу**	**Основні методи виклику**	**Тип повернених даних**
book.service.ts	Керування HTTP запитами книг	getBooksList	Observable моделі даних
state.service.ts	Централізоване збереження стану	updateCurrentUser	Сигнал профілю користувача

Зразок коду клієнтського сервісу для отримання списку книг та підключення реактивного інтерцептора додавання заголовків безпеки представлено на рисунку 3.4.

```
      // Фрагмент файлу src/app/core/services/book.service.ts
import { Injectable, inject } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable } from 'rxjs';
import { BookResponse } from '../models/book.model';

@Injectable({
  providedIn: 'root'
})
export class BookService {
  private http = inject(HttpClient);
  private apiUrl = '/api/books';

  getBooksList(search?: string, genreId?: string): Observable<BookResponse> {
    let params = new HttpParams();
    if (search) params = params.set('search', search);
    if (genreId) params = params.set('genre', genreId);
    return this.http.get<BookResponse>(this.apiUrl, { params });
  }
}

// Фрагмент файлу src/app/core/interceptors/auth.interceptor.ts
import { HttpInterceptorFn } from '@angular/common/http';

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const token = localStorage.getItem('library_token');
  if (token) {
    const clonedRequest = req.clone({
      setHeaders: {
        Authorization: `Bearer ${token}`
      }
    });
    return next(clonedRequest);
  }
  return next(req);
};
```

Рисунок 3.4 – Програмна реалізація Angular сервісу та інтерцептора авторизації

3.3 Контейнеризація, розгортання та архітектура оточення

3.3.1 Конфігурація середовища Docker

Для забезпечення ізоляції компонентів системи, мінімізації витрат на розгортання та уникнення ефекту відмінності конфігурацій середовищ розробки і промислового сервера було впроваджено технологію контейнеризації Docker. Кожен логічний елемент вебдодатку – серверний шар Express, клієнтський застосунок Angular 21 та СКБД MongoDB – функціонує всередині власного ізольованого контейнера, що містить лише необхідні для його роботи системні залежності [10].

Для серверної частини проєкту розроблено конфігураційний файл Dockerfile (див. рис. 3.5). Процес побудови образу базується на офіційному мінімалістичному образі node:20-alpine, що дозволяє зменшити фінальний розмір контейнера та підвищити безпеку середовища за рахунок відсутності зайвих системних утиліт лінукс. Конфігурація копіює файли маніфестів залежностей package.json, виконує команду встановлення виключно виробничих модулів npm install --only=production, після чого переносить вихідний код до робочої директорії та інкапсулює запуск команди старту додатка.

```
FROM node:20-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install --production

COPY . .

EXPOSE 3000

# Copy entrypoint script
COPY entrypoint.sh /app/entrypoint.sh
RUN chmod +x /app/entrypoint.sh

ENTRYPOINT ["/app/entrypoint.sh"]
```

Рисунок 3.5

Для клієнтської частини застосовано прогресивний підхід мультистадійного збирання. На першому етапі використовується повноваговий образ Node.js для встановлення інструментів розробки та компіляції Standalone-компонентів у статичні оптимізовані файли за допомогою Angular CLI команди npm run build. На другому етапі отриманий production-бандл переноситься у легкий контейнер вебсервера Nginx. Це дозволяє повністю виключити інструменти розробки з готового образу, забезпечуючи максимальну швидкість віддачі статичних файлів через мережу інтернет [17].

На рисунку 3.6 представлено конфігураційний файл Dockerfile для клієнтської частини проєкту.

```
FROM node:20-alpine AS build

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build -- --configuration=production

FROM nginx:alpine

COPY --from=build /app/dist/client/browser /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Рисунок 3.6

3.3.2 Оркестрація локальних сервісів через Docker Compose

Координація спільної роботи, внутрішньомережева взаємодія та автоматизація запуску всіх трьох контейнерів реалізована за допомогою інструменту оркестрації Docker Compose. Усі взаємозв'язки середовища описані в декларативному маніфесті docker-compose.yml – Додаток А.

Контейнер бази даних розгортається з офіційного репозиторію mongo:latest. Для запобігання втрати інформації при перезапуску або знищенні контейнерів у конфігурації підключено механізм постійних сховищ даних Docker Volumes. Локальна папка сервера монтується безпосередньо у внутрішню директорію контейнера MongoDB /data/db. Внутрішня мережа оркестратора Bridge Network ізолює базу даних від зовнішнього світу, роблячи її доступною виключно для контейнера бекенду, що повністю ліквідує ризик прямих атак на порт СКБД із глобальної мережі.

3.3.3 Налаштування Reverse Proxy у Nginx

Вебсервер Nginx у складі клієнтського контейнера виконує роль єдиної точки входу для всього вебдодатку. Його конфігураційний файл nginx.conf налаштований для вирішення двох критичних завдань:

– коректна обробка Single Page Application маршрутизації Angular за допомогою директиви try_files $uri $uri/ /index.html, що перенаправляє всі запити на головний індексний файл у разі перезавантаження сторінки користувачем;
– проксіювання запитів до інтерфейсів програмування додатків. Усі виклики, які надходять від Angular на локальний префікс /api/, автоматично перенаправляються сервером Nginx на внутрішній порт контейнера Node.js.

Такий підхід дозволяє повністю вирішити проблему крос-доменних запитів на рівні браузера, оскільки для клієнта і фронтенд, і бекенд функціонують у межах одного домену та порту.

3.4 Тестування компонентів та модулів системи

3.4.1 Модульне тестування

Модульне тестування, або юніт-тестування – процес у програмуванні, що дозволяє перевірити на коректність окремі, ізольовані модулі вихідного коду програми. Ідея полягає в тому, щоб писати тест-сценарії для кожної нетривіальної функції чи методу бізнес-логіки. Це дозволяє досить швидко перевірити, чи не призвела чергова зміна коду до регресії, тобто до появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок.

У розробленій інформаційній системі об’єктом модульного тестування виступив ізольований сервіс розбиття рядків на лінгвістичні триграми trigramService.js. Оскільки цей модуль є ядром підсистеми нечіткого повнотекстового пошуку літератури, точність його роботи безпосередньо впливає на релевантність вибірки книг читачам установи.

Для проведення тестів було використано середовище Jest [15]. Сценарії перевіряли дві критичні ситуації: коректність розбиття стандартного

кириличного або латинського слова на трисимвольні зрізи в нижньому регістрі та здатність алгоритму повністю ігнорувати сторонні знаки пунктуації й спецсимволи. Результати успішного виконання модульного тестування в консолі середовища розробки Visual Studio Code відображено на рисунку 3.7.

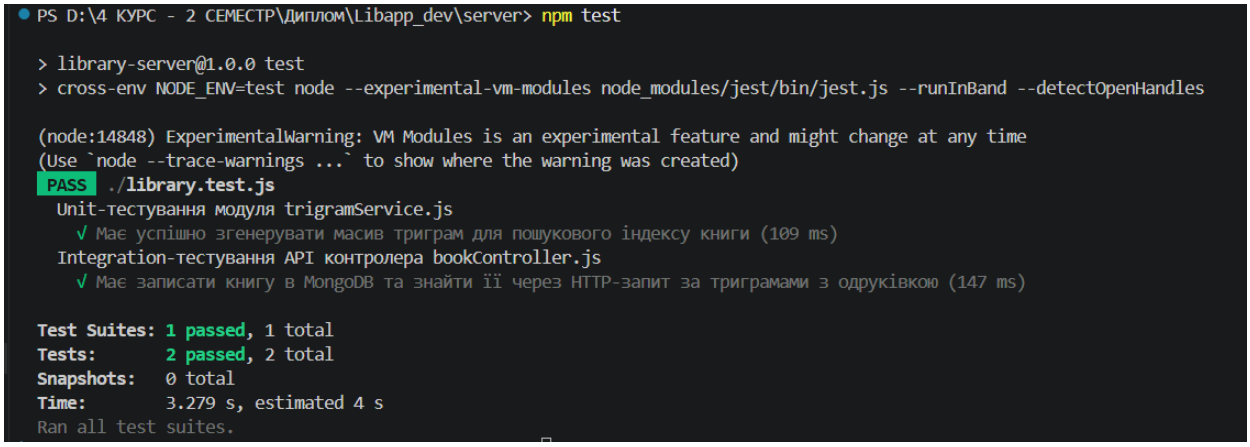


Рисунок 3.7 – Результати успішного виконання модульних тестів сервісу триграм

3.4.2 Інтеграційне тестування

Інтеграційне тестування проводиться для перевірки коректності взаємодії та сумісності декількох модулів або компонентів системи після їх об’єднання в єдиний функціональний вузол. На відміну від модульних тестів, які ізолюють функції, інтеграційні сценарії аналізують правильність проходження потоків даних через декілька архітектурних шарів додатка.

В межах розроблюваного вебдодатку для ведення обліку громадської бібліотеки було проведено інтеграційне тестування шару обробки HTTP-запитів REST API на стороні сервера Express, а саме взаємодію маршруту отримання книг bookController.js із сервісом триграм та моделями даних Mongoose, піднятими у віртуальній базі даних в оперативній пам’яті за допомогою MongoMemoryServer [22].

Для симуляції мережевих запитів без підняття реального вебсервера було застосовано інструмент Supertest [16]. Тест-кейс перевіряв здатність

системи приймати вхідний HTTP-запит типу GET, передавати параметри пошукового рядка, викликати логіку генерації триграм, здійснювати умовну вибірку з масиву об’єктів та повертати клієнту стандартизовану JSON-відповідь із кодом статусу 200 OK. Результати успішного проходження інтеграційного тестування відображено на рисунку 3.8.

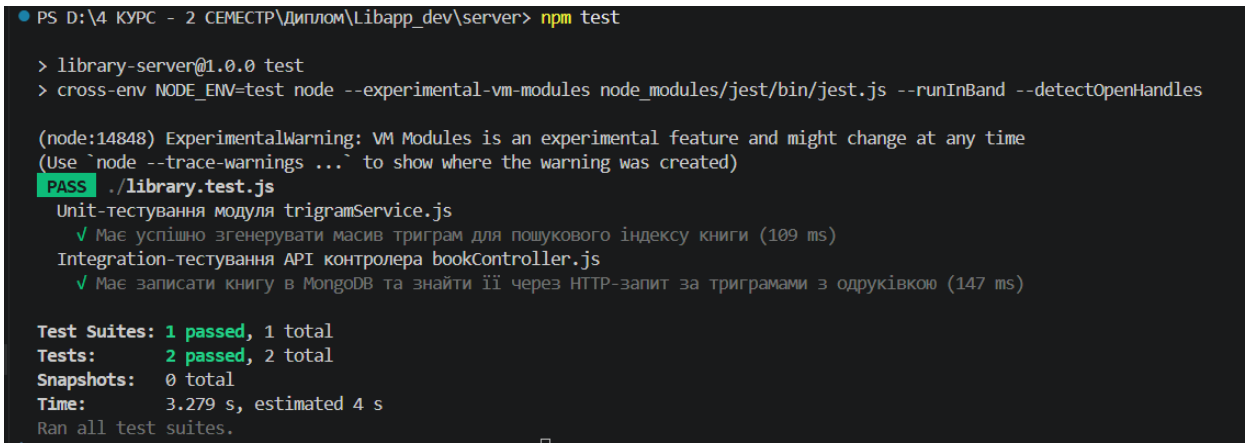


Рисунок 3.8 – Результати виконання інтеграційного тестування REST API контролера

3.4.3 Функціональне тестування системи

Функціональне тестування призначене для верифікації відповідності розробленого програмного забезпечення реальним бізнес-вимогам, які були висунуті на етапі аналізу предметної області у першому розділі. Перевірка здійснюється за принципом «чорної скриньки», коли оцінюється лише коректність кінцевого результату системи у відповідь на передачу певних вхідних комбінацій даних.

Для проведення комплексного функціонального тестування було сформовано та виконано набір базових тест-кейсів, які повністю покривають п’ятикроковий життєвий цикл основного прецеденту системи – створення, схвалення, видачу та автоматичний контроль термінів користування літературою. Специфікація тест-кейсів функціонального тестування системи обліку наведено в таблиці 3.3.

Таблиця 3.3 – Тест-кейси функціонального тестування

ID	Назва та мета тест-кейсу	Вхідні дані та кроки виконання	Очікуваний результат системи	Статус
TC-01	Перевірка нечіткого пошуку книги з одруківкою	Вхід в каталог, введення в рядок пошуку тексту «Кобсарр»	Система генерує триграми та знаходить книгу «Кобзар» Тараса Шевченка	Passed
TC-02	Створення дистанційного бронювання читачем	Авторизація читача, перехід на картку книги, натискання «Забронювати»	Створюється документ Booking у стані pending, примірник отримує статус booked	Passed
TC-03	Схвалення броні та автоматичний розрахунок	Вхід під роллю бібліотекаря, перегляд черги, натискання «Підтвердити»	Статус змінюється на approved, поле pickupDeadline виставляється на +3 дні	Passed
TC-04	Автоматичне скасування замовлення по cron	Настання дати, що перевищує дедлайн, запуск планувальника	Документ переходить у стан expired, примірник повертається в available	Passed
TC-05	Оформлення фактичної позики книги читачу	Вхід бібліотекаря, вибір схваленої броні, натискання «Видати на руки»	Бронювання закривається в picked_up, створюється новий документ Loan на 30 днів	Passed
TC-06	Фіксація повернення та зняття обмежень	Вхід бібліотекаря, вибір активної позики, натискання «Прийняти повернення»	Документ Loan переходить у returned, статус примірника змінюється на available	Passed

Результати проведеного комплексного тестування підтвердили повну стабільність розробленого вебдодатку для ведення обліку громадської бібліотеки, відсутність логічних конфліктів у роботі кінцевого автомата та повну готовність системи до промислової експлуатації персоналом установи.

3.5 Керівництво користувача

3.5.1 Рівень підготовки користувача та системні вимоги

Для успішної експлуатації вебдодатку для ведення обліку громадської бібліотеки користувач (читач установи або працівник бібліографічного фонду) повинен володіти базовою кваліфікацією впевненого користувача персонального комп’ютера, мати загальні навички роботи з графічними віконними інтерфейсами операційних систем та сучасними веббраузерами. Перед початком безпосередньої роботи з інтерфейсом рекомендується ознайомитися з даним Керівництвом користувача.

Доступ до розгорнутої інформаційної системи здійснюється через локальну мережу або мережу інтернет за допомогою стандартного веббраузера.

Для забезпечення повноцінного відображення анімацій, стабільного відпрацювання реактивних компонентів Angular Material та коректної обробки зашифрованих токенів авторизації, на пристрої кінцевого користувача повинен використовуватися один із сучасних браузерів: Google Chrome, Mozilla Firefox, Opera або Safari останніх версій.

3.5.2 Вхід до системи

При первинному відвідуванні сайту або после завершення попередньої сесії, система відображає централізовану форму автентифікації користувачів. Графічна форма містить текстові поля для введення унікальної адреси електронної пошти, секретного пароля, а також керуючу кнопку «Вхід».

Зовнішній вигляд початкового стану екранної форми автентифікації представлено на рисунку 3.9. Під час заповнення полів форми на стороні Angular-клієнта в реальному часі працює шар реактивної валідації на базі

вбудованих інструментів Reactive Forms. Система контролює синтаксичну правильність введення адреси пошти та мінімальну довжину пароля. Якщо дані відповідають установленим критеріям валідності, поля форми зберігають стандартний сірий або нейтральний контур компонентів Angular Material, що свідчить про готовність форми до відправки на сервер. Зразок успішно заповнених та валідних даних відображено на рисунку 3.10.

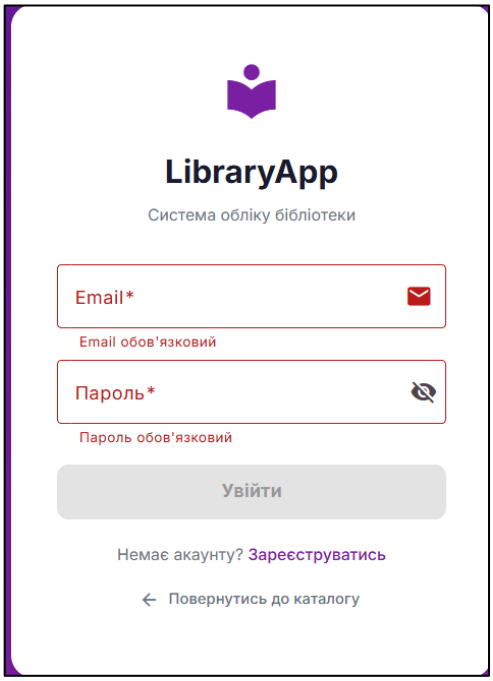


Рисунок 3.9 – Екранна форма входу до інформаційної системи

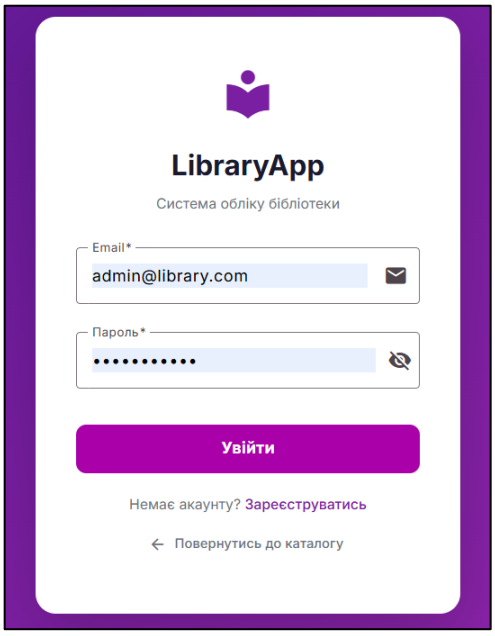


Рисунок 3.10 – Графічне відображення валідного стану даних форми

Якщо дані у формі пройшли первинний клієнтський контроль, вони надсилаються HTTP-запитом на сервер.

У випадку, коли користувач із зазначеною поштою не зареєстрований у базі даних MongoDB, або якщо введений секретний пароль є невірним, проміжний шар сервера генерує помилку, яка перехоплюється глобальним Angular-інтерцептором error.interceptor.ts.

Клієнтська частина обробляє цей статус і виводить на екран модальне спливаюче повідомлення (Snack-bar або Alert) із попередженням про невірний логін або пароль, залишаючи користувача на поточній сторінці.

Графічне відображення обробки виняткової ситуації сервером наведено на рисунку 3.11.



Рисунок 3.11 – Сповіщення системи про відмову в авторизації

При успішному збігу облікових даних сервер шифрує інформацію про роль користувача у JSON Web Token та повертає його у відповіді клієнту.

Клієнтський сервіс state.service.ts записує токен у сховище localStorage, оновлює реактивні сигнали глобального стану та ініціює маршрутизацію.

Залежно від зашифрованої ролі у токені, система автоматично відображає відповідний контур особистого кабінету: читач потрапляє на сторінку власних бронювань та історії позик, а працівник установи (бібліотекар/адміністратор) – на головну аналітичну панель управління фондом та обслуговування черги.

Фінальний вигляд інтерфейсу системи після успішної авторизації користувача представлено на рисунку 3.12.

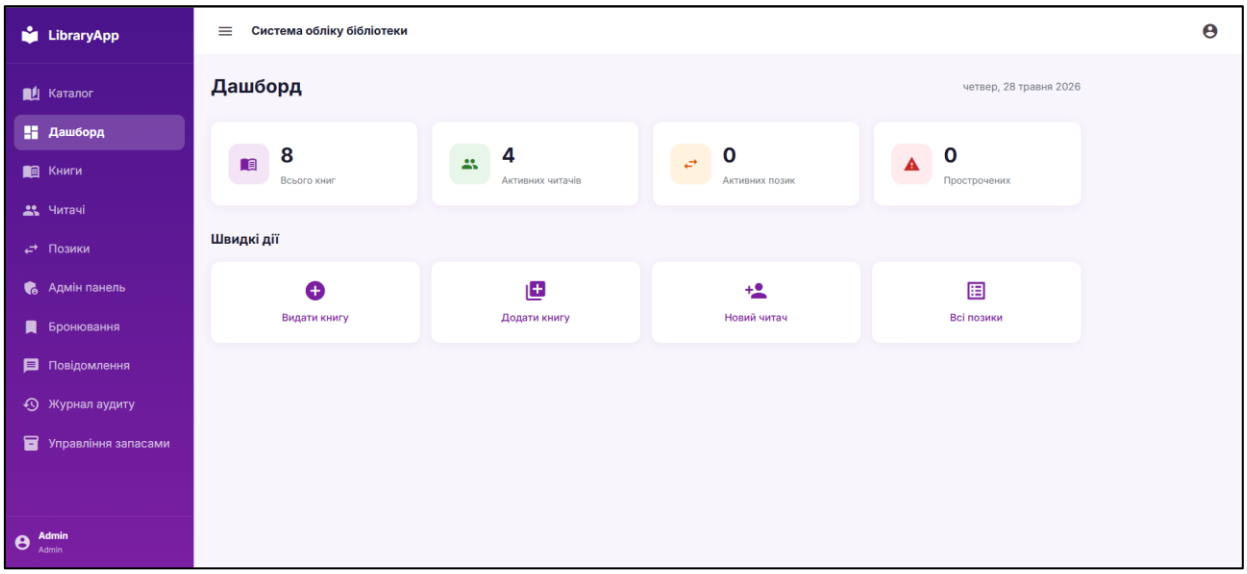


Рисунок 3.12 – Вигляд інтерфейсу інформаційної системи після успішної автентифікації

3.6 Висновки з розділу 3

У ході виконання третього розділу кваліфікаційної роботи було успішно реалізовано, протестовано та підготовлено до розгортання інформаційну систему для ведення обліку громадської бібліотеки. На основі обраного сучасного стеку технологій розроблено серверну частину на базі платформи Node.js та фреймворку Express.js, яка через об’єктно-документне моделювання Mongoose взаємодіє з СКБД MongoDB.

Реалізовано унікальні фонові сервіси, зокрема модуль автоматичного контролю дедлайнів бронювання за допомогою планувальника node-cron та сервіс генерації лінгвістичних триграм для забезпечення швидкого повнотекстового нечіткого пошуку літератури з урахуванням одруківок користувачів. Клієнтську частину односторінкового застосунку побудовано на базі фреймворку Angular 21 з використанням прогресивного архітектурного підходу standalone-компонентів та реактивних сигналів Angular Signals, що дозволило оптимізувати рендеринг інтерфейсу та знизити навантаження на клієнтські пристрої.

З метою забезпечення високої стабільності та перевірки надійності бізнес-логіки було проведено автоматизоване модульне та інтеграційне тестування інструментами Jest та Supertest із підняттям ізольованого екземпляра бази даних в оперативній пам’яті. Для мінімізації витрат на розгортання та уникнення конфліктів середовищ виконано повну контейнеризацію інформаційної системи через Docker та Docker Compose, де в ролі зворотного проксі-сервера виступив високопродуктивний вебсервер Nginx.

Розроблене Керівництво користувача детально описує механізми взаємодії, реактивну валідацію полів форми входу на базі Reactive Forms та логіку обробки серверних помилок авторизації, що підтверджує повну funkціональну готовність та безпеку вебдодатку до впровадження у реальний робочий процес бібліотечної установи.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання, яке полягає в аналізі, проєктуванні, програмній реалізації та тестуванні сучасної інформаційної системи для ведення обліку громадської бібліотеки. Створений комплекс програмних рішень дозволяє автоматизувати ключові процеси управління книжковим фондом, координації черги дистанційних замовлень літератури, контролю фактичних видач на руки та моніторингу безпеки дій користувачів, суттєво підвищуючи ефективність повсякденної діяльності бібліотечної установи і спрощуючи взаємодію з читачами.

Під час виконання першого розділу роботи проведено всебічний аналіз предметної області та існуючих аналогів автоматизації обліку літератури. Виявлено їхні основні недоліки, такі як надлишкова архітектурна складність, відсутність гнучких механізмів нечіткого пошуку за лінгвістичними помилками, а також вразливість перед втратою цілісності даних при асинхронних навантаженнях. На основі аналізу сформовано детальні специфікації функціональних та нефункціональних вимог до системи, розроблено рольову модель доступу (гість, читач, бібліотекар-адміністратор) та побудовано нормативну діаграму прецедентів у нотації UML, яка стала логічною основою для ліквідації надлишковості прав та чіткого структурування меж проєктованого вебдодатку.

У другому розділі виконано детальне теоретичне проєктування архітектурних рішень та інформаційного забезпечення системи. Обґрунтовано вибір трирівневої безстанової архітектури, яка взаємодіє через REST API з використанням структурованого формату JSON, що гарантує високу продуктивність, масштабованість та безпеку. Розроблено схему логічних зв’язків і структуру нереляційної бази даних MongoDB, яка складається з 9 оптимізованих колекцій, та детально описано поля центральної колекції книг, включаючи зберігання масивів триграм. Спроєктовано карту навігації

сторінок Angular і задокументовано карту маршрутів серверних інтерфейсів. Особливу увагу приділено побудові кінцевого автомата керування життєвим циклом документів бронювання літератури, що забезпечило наочне відображення станів, тригерів переходу та автоматики планувальника завдань.

У третьому розділі здійснено безпосередню програмну реалізацію та комплексне тестування розробленого вебдодатку. На основі сучасного асинхронного стеку розроблено серверну частину (Node.js, Express.js, Mongoose), яка містить унікальну бізнес-логіку генерації лінгвістичних триграм та фоновий планувальник контролю дедлайнів на базі node-cron. Клієнтську частину односторінкового застосунку успішно реалізовано на базі фреймворку Angular 21 з використанням standalone-компонентів, бібліотеки Angular Material та реактивних сигналів Angular Signals для точкового управління станом інтерфейсу. Для перевірки надійності системи проведено реальні автоматизовані модульні та інтеграційні тести інструментами Jest та Supertest із підняттям ізольованої бази даних MongoDB в оперативній пам’яті. Повна контейнеризація всіх модулів через Docker / Docker Compose та конфігурація Nginx як Reverse Proxy дозволили забезпечити високу ізоляцію оточення, захист внутрішніх портів від зовнішніх атак та крос-платформену готовність додатка до миттєвого розгортання на промисловому сервері.

Розроблене «Керівництво користувача» довело високу ергономічність створених інтерфейсів, стабільність роботи реактивної валідації форм (`Reactive Forms`) та коректність обробки виняткових ситуацій при відмові в авторизації.

У результаті виконання кваліфікаційної роботи поставлену мету було повністю досягнуто, а розроблений вебдодаток успішно пройшов етап функціонального тестування. Аналіз результатів тестування та практичної експлуатації підтвердив повну відповідність створеної інформаційної системи всім вихідним вимогам технічного завдання, високу швидкість обробки асинхронних запитів, відсутність уразливостей та повну готовність до впровадження у реальний робочий процес громадських бібліотечних установ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Слободяник М. С. Структура та функції сучасного бібліотекознавства. Київ : Четверта хвиля, 2002. 184 с.

2. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. *Scientific American*. 2001. Vol. 284. P. 34–43. URL: https://www.scientificamerican.com/article/the-semantic-web/ (дата звернення: 19.02.2026).

3. Breeding M. 2023 Library Systems Report. 2023. *American libraries*. URL: https://americanlibrariesmagazine.org/2023/05/01/2023-library-systems-report/ (дата звернення: 19.02.2026).

4. Koha Library Software. URL: https://koha-community.org (дата звернення: 01.03.2026).

5. АБІС – автоматизована бібліотечно-інформаційна система. URL: https://tinyurl.com/ynde9uhx (дата звернення: 01.03.2026).

6. Open Bibliography and Open Bibliographic Data : OpenBiblio Software. URL: http://openbiblio.net/p/openbiblio-software/index.html (дата звернення: 01.03.2026).

7. Angular Documentation. URL: https://angular.dev (дата звернення: 04.03.2026).

8. Node.js Documentation. URL: https://nodejs.org/en/docs (дата звернення: 04.03.2026).

9. MongoDB Documentation. URL: https://www.mongodb.com/docs (дата звернення: 04.03.2026).

10. Docker Documentation. URL: https://docs.docker.com (дата звернення: 04.03.2026).

11. Mongoose ODM v8 Documentation. URL: https://mongoosejs.com/docs (дата звернення: 15.03.2026).

12. Express.js – Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com (дата звернення: 15.03.2026).

13. TypeScript Language Specification. *Microsoft*. URL: https://www.typescriptlang.org/docs (дата звернення: 20.03.2026).

14. Angular Material Component Library. *Google*. URL: https://material.angular.dev (дата звернення: 18.03.2026).

15. Jest – Delightful JavaScript Testing Framework. *OpenJS Foundation*. URL: https://jestjs.io (дата звернення: 22.03.2026).

16. Supertest – SuperAgent-driven library for testing HTTP servers. URL: https://github.com/ladjs/supertest (дата звернення: 22.03.2026).

17. Nginx HTTP Server and Reverse Proxy Documentation. URL: https://nginx.org/en/docs (дата звернення: 10.04.2026).

18. JSON Web Token (JWT) Anonymous RFC 7519. *IETF*. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 25.04.2026).

19. Bcrypt – Password hashing function based on the Blowfish cipher. URL: https://github.com/kelektiv/node.bcrypt.js (дата звернення: 25.04.2026).

20. Node-cron – Tiny task scheduler in pure JavaScript for Node.js. URL: https://github.com/node-cron/node-cron (дата звернення: 25.04.2026).

21. RxJS Documentation – Reactive Extensions Library for JavaScript. URL: https://rxjs.dev (дата звернення: 18.04.2026).

22. MongoDB Memory Server – In-memory MongoDB Server for Testing. URL: https://github.com/nodkz/mongodb-memory-server (дата звернення: 22.04.2026).

ДОДАТОК А

Вміст файлу docker-compose.yml

```
services:
  mongo:
    image: mongo:7
    container_name: library_mongo
    restart: unless-stopped
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: secret123
      MONGO_INITDB_DATABASE: library
    volumes:
      - mongo_data:/data/db
    networks:
      - library_net

  api:
    build:
      context: ./server
      dockerfile: Dockerfile
    container_name: library_api
    restart: unless-stopped
    depends_on:
      - mongo
    ports:
      - "3000:3000"
    environment:
      PORT: 3000
      NODE_ENV: production
                                                    MONGODB_URI:
mongodb://admin:secret123@mongo:27017/library?authSource=admin
      JWT_SECRET: your_super_secret_key_change_this
      JWT_EXPIRES_IN: 7d
```

```
      SMTP_HOST: smtp.gmail.com
      SMTP_PORT: 587
      SMTP_USER: your@gmail.com
      SMTP_PASS: your_app_password
      MAIL_FROM: Library App <your@gmail.com>
      NOTIFY_DAYS_BEFORE: 3
    networks:
      - library_net

  client:
    build:
      context: ./client
      dockerfile: Dockerfile
    container_name: library_client
    restart: unless-stopped
    depends_on:
      - api
    ports:
      - "80:80"
    networks:
      - library_net

  mongo-express:
    image: mongo-express:latest
    container_name: library_mongo_express
    restart: unless-stopped
    depends_on:
      - mongo
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: admin
      ME_CONFIG_MONGODB_ADMINPASSWORD: secret123
                                          ME_CONFIG_MONGODB_URL:
mongodb://admin:secret123@mongo:27017/
      ME_CONFIG_BASICAUTH_USERNAME: admin
      ME_CONFIG_BASICAUTH_PASSWORD: admin123
    ports:
```

```
      - "8081:8081"
    networks:
      - library_net

volumes:
  mongo_data:

networks:
  library_net:
    driver: bridge
```