МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

ГРИГОРІВ Марія-Вероніка Віталіївна

Вебсайт роботи бібліотеки / Website of the library's operations

спеціальність: 123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Кваліфікаційна робота

Виконала: студентка групи КІ–41
ГРИГОРІВ Марія-Вероніка
Віталіївна

Науковий керівник
к.т.н., доц. Піцун О.Й.

ТЕРНОПІЛЬ – 2026

АНОТАЦІЯ

Григорів М.-В. В. Веб-сайт онлайн-бібліотеки SmartLib з інтегрованим інтелектуальним помічником (AI advisor). – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2026.

Робота написана обсягом 70 сторінок і містить 8 рисунків, 16 таблиць, 3 додатки та 30 джерел за переліком посилань.

Метою кваліфікаційної роботи є розробка веб-сайту онлайн-бібліотеки SmartLib для зручного пошуку, зберігання та читання книг в онлайн-режимі. Спроєктовано клієнт-серверну архітектуру системи на основі сучасного технологічного стеку (React, NestJS, PostgreSQL) із реалізацією реєстрації та авторизації користувачів, каталогізації, зберігання та пошуку книг. Окрему увагу приділено інтеграції штучного інтелекту: реалізовано інтелектуального помічника (AI advisor) на базі зовнішніх API-сервісів (Claude, OpenAI), який забезпечує пошук, персональні рекомендації книг та консультування користувачів у діалоговому режимі. Виконано порівняльний аналіз розробленого продукту з аналогами та техніко-економічне обґрунтування доцільності його розробки.

Ключові слова: ОНЛАЙН-БІБЛІОТЕКА, ВЕБ-ДОДАТОК, ШТУЧНИЙ ІНТЕЛЕКТ, AI-ПОМІЧНИК, REACT, NESTJS, POSTGRESQL.

ANNOTATION

Hryhoriv M.-V. V. SmartLib online library website with an integrated intelligent assistant (AI advisor). – Manuscript.

Qualification work for the Bachelor degree in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2026.

The work is written in 70 pages and contains 8 figures, 16 tables, 3 appendices and 30 references.

The purpose of the qualification work is to develop the SmartLib online library website for convenient online search, storage and reading of books. A client-server system architecture is designed based on a modern technology stack (React, NestJS, PostgreSQL), implementing user registration and authorization, cataloguing, storage and search of books. Particular attention is paid to the integration of artificial intelligence: an intelligent assistant (AI advisor) based on external API services (Claude, OpenAI) is implemented, providing search, personalized book recommendations and interactive user consulting. A comparative analysis of the developed product with analogues and a technical and economic feasibility study of its development were carried out.

Keywords: ONLINE LIBRARY, WEB APPLICATION, ARTIFICIAL INTELLIGENCE, AI ASSISTANT, REACT, NESTJS, POSTGRESQL.

ЗМІСТ

ВСТУП 9

1 АНАЛІЗ ВЕБ-САЙТІВ ОНЛАЙН-БІБЛІОТЕК 11

1.2 Програмні засоби розробки веб-сайтів 14

1.3 Особливості веб-сайтів бібліотек 18

1.4 Висновки та постановка задачі 21

2 АЛГОРИТМИ РОБОТИ МОДУЛІВ САЙТУ 24

2.1 Алгоритм реєстрації та авторизації в системі 24

2.2 Алгоритм збереження книг та управління користувачами 28

2.3 Структура бази даних 32

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ САЙТУ 37

3.1 Серверна частина 37

3.2 Графічний інтерфейс клієнтської частини 41

3.3 Порівняльний аналіз розробленого сайту з аналогами 46

Додаток А 56

Додаток Б 56

ДОДАТОК В 67

ВСТУП

Бібліотеки завжди відігравали важливу роль у забезпеченні доступу до знань, однак традиційний формат їхньої роботи має суттєві обмеження: прив'язаність до фізичного приміщення, фіксований графік, обмежена кількість примірників. З поширенням Інтернету з'явилася можливість організувати читання та пошук книжок у цілодобовому режимі без необхідності відвідувати бібліотеку особисто. Сьогодні онлайн-бібліотеки є невід'ємною частиною цифрового освітнього та культурного середовища.
Попит на якісні веб-сервіси для читання та пошуку книжок стрімко зростає. Водночас більшість готових рішень у цій сфері є або надто громіздкими для навчального проєкту, або не відповідають сучасним стандартам розробки. Це робить самостійну реалізацію веб-сайту онлайн-бібліотеки актуальним практичним завданням для студентів спеціальності «Комп'ютерні науки».

Особливістю даної роботи є застосування агентного штучного інтелекту в процесі розробки. Інструмент Claude Code від компанії Anthropic дозволяє делегувати значну частину рутинних завдань - написання коду, генерацію тестів, рефакторинг - безпосередньо AI-агенту через командний рядок. Такий підхід є відносно новим у навчальній та інженерній практиці і потребує окремого дослідження з точки зору ефективності та якості отриманих результатів.

Об'єктом дослідження є процес розробки веб-застосунків з використанням агентних AI-систем.

Предметом дослідження є методи і практики автоматизації розробки веб-сайту онлайн-бібліотеки за допомогою CLI-агентів.

Метою бакалаврської роботи є розробка веб-сайту онлайн-бібліотеки з використанням агентного AI-інструментів для автоматизації основних етапів створення програмного продукту.

Актуальність роботи полягає у поєднанні практичної задачі створення

корисного веб-застосунку з дослідженням нового підходу до розробки програмного забезпечення на основі агентних систем штучного інтелекту.

Для досягнення мети поставлено такі завдання:

- розглянути наявні рішення у сфері цифрових бібліотек та визначити вимоги до розроблюваної системи;
- розробити архітектуру вебсайту;
- програмно реалізувати фронтенд і бекенд складові;
- організувати тестування та рефакторинг засобами агента;
- провести порівняльну оцінку ефективності агентного підходу.

Дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ВЕБ-САЙТІВ ОНЛАЙН-БІБЛІОТЕК

1.1 Поняття веб-сайту та класифікація веб-ресурсів

Інтернет є глобальною інформаційною мережею, що об'єднує мільярди комп'ютерів і пристроїв у єдиний інформаційний простір. Значна частина ресурсів цієї мережі організована у вигляді веб-сайтів - сукупностей взаємопов'язаних сторінок, доступних за унікальною адресою (URL) і розміщених на веб-серверах. Для перегляду цих сторінок користувач застосовує браузер, який отримує дані від сервера та відображає їх у зручному вигляді.

Веб-сайт у сучасному розумінні - це набір веб-сторінок, об'єднаних спільним доменним ім'ям, єдиною навігацією та дизайном [1]. Фізично сайт може розміщуватись як на одному, так і на кількох серверах. Ключовою відмінністю між веб-сайтом і звичайним веб-застосунком є рівень інтерактивності: якщо сайт переважно надає контент для читання, то застосунок реалізує повноцінну бізнес-логіку - автентифікацію, транзакції, управління станом та обмін даними з сервером у реальному часі.

Класифікацій веб-ресурсів існує кілька, і кожна відображає різні аспекти їхньої природи. За призначенням сайти поділяють на інформаційні, сервісні, комерційні та розважальні. За технологією відображення розрізняють статичні сайти, де сторінки підготовлено заздалегідь і передаються клієнту без змін, та динамічні, де HTML-відповідь формується на льоту відповідно до запиту користувача. За доступністю ресурси бувають відкритими, напіввідкритими (з реєстрацією) та закритими.

Переваги та недоліки статичних і динамічних сайтів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння статичних та динамічних веб-сайтів

Критерій	Статичний сайт	Динамічний сайт
Генерація сторінок	Наперед підготовлені HTML-файли	Формуються на сервері при кожному запиті
Складність розробки	Низька, не потребує серверної логіки	Вища, вимагає бекенду та БД
Актуальність даних	Оновлюється вручну при кожній зміні	Завжди відображає поточний стан БД
Придатність для бібліотеки	Не підходить (контент постійно змінюється)	Оптимальний варіант

З таблиці видно, що для веб-сайту онлайн-бібліотеки, де каталог книг, дані користувачів і нотатки постійно оновлюються, оптимальним є саме динамічний тип. Саме він лежить в основі системи SmartLib, що розробляється в даній роботі.

Окремим підкласом є Single-Page Application (SPA) - різновид динамічного застосунку, де браузер завантажує один HTML-документ, а подальша навігація відбувається засобами JavaScript без перезавантаження сторінки. Такий підхід забезпечує плавний інтерфейс, характерний для нативних додатків, і є домінуючою архітектурою для сучасних веб-застосунків, зокрема для SmartLib.

Взаємодія між рівнями у сучасних веб-застосунках здебільшого відбувається через API (Application Programming Interface). Найбільшого поширення набула архітектурна концепція REST (Representational State Transfer), сформульована Роєм Філдінгом у 2000 році. RESTful API використовує протокол HTTP та стандартні методи запитів: GET для читання ресурсів, POST для створення, PUT/PATCH для оновлення, DELETE для видалення. Відповіді зазвичай передаються у форматі JSON, що підтримується нативно усіма сучасними мовами програмування. Ключовими принципами REST є відсутність стану на

сервері (statelessness), єдиний інтерфейс і можливість кешування відповідей, що сукупно забезпечує масштабованість і простоту тестування.

Важливою характеристикою архітектури є спосіб рендерингу. Виділяють три основні підходи: серверний рендеринг (Server-Side Rendering, SSR), клієнтський рендеринг (Client-Side Rendering, CSR) та статична генерація (Static Site Generation, SSG). При SSR HTML формується на сервері при кожному запиті та передається браузеру вже готовим, що позитивно впливає на SEO та швидкість першого завантаження. При CSR браузер отримує мінімальний HTML-шаблон і виконує весь рендеринг засобами JavaScript, що дає більшу інтерактивність, але уповільнює початкове відображення сторінки. SSG генерує HTML заздалегідь на етапі збірки, що забезпечує максимальну швидкість роботи, але не підходить для динамічного контенту. Для системи SmartLib обрано підхід CSR на базі React SPA, оскільки контент каталогу і персональні дані користувачів є персоналізованими й динамічними, а SEO-індексація не є пріоритетною вимогою.

Окремо слід розглянути концепцію розподілу відповідальності між фронтендом і бекендом. У монолітному підході весь код і серверна логіка, і генерація HTML розміщується в одному застосунку. Така архітектура проста у розгортанні, але утруднює масштабування та паралельну розробку. Натомість підхід «відокремлений фронтенд / відокремлений бекенд» (decoupled architecture) передбачає, що клієнтська частина є незалежним застосунком, який взаємодіє із сервером виключно через API. Це дозволяє розробляти, тестувати та деплоїти клієнт і сервер незалежно, а також використовувати одне й те ж API для різних клієнтів: веб-браузера та мобільного застосунку. Саме такий підхід реалізовано у SmartLib: React-застосунок є повністю незалежним від сервера і спілкується з ним через RESTful API.

Невід'ємним елементом архітектури сучасного веб-застосунку є також механізм автентифікації та авторизації. Оскільки REST API є stateless, кожен запит до захищених ресурсів повинен містити токен ідентифікації

користувача. Найпоширенішим форматом є JWT (JSON Web Token), підписаний JSON-об'єкт, що містить інформацію про користувача та строк дії. Клієнт зберігає токен після успішного входу і додає його у заголовок Authorization кожного наступного запиту. Сервер верифікує підпис токена без звернення до бази даних, що зменшує латентність і спрощує горизонтальне масштабування. У SmartLib реалізовано саме цей механізм: FastAPI-бекенд видає JWT після автентифікації, а React-клієнт зберігає його у localStorage та автоматично прикріплює до запитів через HTTP-інтерцептор.

Таким чином, архітектура сучасного веб-застосунку є багатошаровою системою, де кожен компонент: клієнт, сервер, база даних, механізм автентифікації, виконує чітко визначену роль. Усвідомлення цих архітектурних принципів є необхідною основою для проєктування систем, здатних масштабуватися, підтримуватися та розвиватися в умовах реального виробничого середовища.

Отже, у даному підрозділі розглянуто поняття веб-сайту, проаналізовано основні класифікаційні ознаки веб-ресурсів та обґрунтовано вибір динамічного SPA-типу як оптимального для реалізації онлайн-бібліотеки.

1.2 Програмні засоби розробки веб-сайтів

Сучасна розробка веб-застосунків спирається на широкий перелік інструментів і фреймворків, які охоплюють усі рівні системи: клієнтську частину, серверну логіку та зберігання даних. Вибір конкретного стеку технологій суттєво впливає на швидкість розробки, зручність підтримки та масштабованість проєкту. У даному підрозділі розглянуто інструменти, що були обрані для реалізації системи SmartLib.

1.1.1 Засоби розробки клієнтської частини

Для реалізації клієнтської частини SmartLib обрано React - бібліотеку для побудови користувацьких інтерфейсів, розроблену компанією Meta у 2013 році. React використовує компонентний підхід, де інтерфейс будується з

незалежних, перевикористовуваних блоків із власним станом. Ключовою особливістю є концепція Virtual DOM, що дозволяє ефективно оновлювати лише ті частини сторінки, які змінились. За даними Stack Overflow Developer Survey 2024, React залишається найпоширенішим JavaScript-фреймворком із часткою понад 40% серед розробників.

Мову розробки обрано TypeScript - типізований надмножинник JavaScript, що дозволяє виявляти помилки на етапі компіляції замість виконання. Усі компоненти, хуки та API-клієнти у SmartLib написані на TypeScript, що суттєво покращує читабельність коду та спрощує взаємодію між клієнтом і сервером через спільні типи.

Інструментом збірки обрано Vite - сучасний бандлер нового покоління, що використовує native ES modules у режимі розробки та Rollup для продакшн-збірки. Порівняно з Webpack, Vite забезпечує значно швидший старт сервера розробки та миттєве оновлення при змінах коду завдяки Hot Module Replacement на рівні ES-модулів.

Для управління глобальним станом застосунку та HTTP-запитами використовується Redux Toolkit разом із RTK Query. Redux Toolkit - це офіційний, рекомендований Anthropic набір утиліт для написання Redux-логіки з мінімальним шаблонним кодом. RTK Query є частиною цього пакету і автоматизує кешування, повторні запити та синхронізацію серверного стану без написання ручних thunks і reducers.

Навігацію між сторінками застосунку реалізовано через React Router DOM - стандартну бібліотеку маршрутизації для React. SmartLib має дев'ять сторінок: LoginPage, BooksPage, BookDetailPage, SearchPage, ProfilePage, MyBooksPage, DashboardPage, AdminPage та NotFoundPage. Доступ до сторінок розмежовано за ролями через компоненти ProtectedRoute та RoleGuard.

Стилізацію компонентів виконано за допомогою Tailwind CSS - utility-first фреймворку, де стилі задаються безпосередньо у JSX через набір готових

CSS-класів. Такий підхід усуває потребу у окремих CSS-файлах і прискорює прототипування. Іконки підключено через бібліотеку Lucide React, а автентифікацію через Google реалізовано за допомогою пакету @react-oauth/google.

1.1.2 Засоби розробки серверної частини

Серверну частину SmartLib побудовано на NestJS - прогресивному Node.js-фреймворку, що використовує TypeScript і вдихновлений архітектурними принципами Angular. NestJS організовує серверний код у модулі, контролери та сервіси, реалізує Dependency Injection та надає вбудовану підтримку REST API, GraphQL, WebSockets та мікросервісів. Перевагою NestJS є чітка структура проєкту, яка добре масштабується зі зростанням кодової бази.

Автентифікацію побудовано на основі JWT (JSON Web Token) у поєднанні з бібліотекою Passport.js. Після успішного входу сервер видає підписаний токен, який клієнт надсилає у заголовку кожного наступного запиту. Для перевірки токенів реалізовано стратегію JwtStrategy. Паролі зберігаються у хешованому вигляді за допомогою алгоритму bcrypt. Також підтримується автентифікація через Google OAuth.

Для взаємодії з базою даних застосовано подвійний підхід: TypeORM відповідає за більшість CRUD-операцій через декоративні репозиторії, тоді як Prisma Client виконує міграції та складні запити. Схема бази даних описана у файлі schema.prisma і включає моделі: Users, Profiles, Books, Genres, BookGenres, Orders, Notes та SavedBooks. API задокументовано через Swagger (доступний за адресою /api/docs), а від перевантаження запитами захищає NestJS Throttler.

Функцію AI-рекомендацій реалізовано через офіційний Node.js-клієнт OpenAI API. При зверненні до ендпоінту /ai/recommendations сервіс надсилає до OpenAI запит із контекстом читача і отримує персоналізований список книг із поясненням причин рекомендацій. Зовнішній пошук книг реалізовано

через модуль books-parser, який паралельно звертається до Google Books API та Open Library і нормалізує відповіді до спільного формату.

1.1.3 Система управління базою даних

Для зберігання даних SmartLib використовує PostgreSQL - об'єктно-реляційну СУБД з відкритим вихідним кодом, що є де-факто стандартом у сучасних веб-застосунках. PostgreSQL підтримує повноцінні транзакції ACID, складні JOIN-запити, масиви (використовуються у моделі SavedBooks для поля authors та categories), а також розширення, що дозволяють у перспективі додати векторний пошук (pgvector) для покращення AI-рекомендацій.

Схема бази даних включає вісім моделей. Users зберігає облікові записи з підтримкою м'якого видалення через поле deletedAt. Profiles містить розширену інформацію профілю. Books описує книги з каталогу з полями

назви, автора, року видання та категорії. Notes дозволяє читачам зберігати виділений текст і коментарі до конкретної книги на певній сторінці. SavedBooks знімає знімок метаданих книги з зовнішніх провайдерів (Google Books або Open Library) та AI, щоб список залишався доступним навіть при недоступності зовнішнього API.

Таблиця 1.2 – Технологічний стек проєкту SmartLib

Рівень	**Технологія**	**Призначення**
Фронтенд	React 18 + TypeScript	Компонентний UI, статична типізація
Фронтенд	Vite	Збірка проєкту, HMR-сервер розробки
Фронтенд	Redux Toolkit + RTK Query	Глобальний стан, HTTP-кешування
Фронтенд	Tailwind CSS	Стилізація через utility-класи

Бекенд	NestJS + TypeScript	REST API, модульна серверна архітектура
Бекенд	Passport JWT + bcrypt	Автентифікація та хешування паролів
Бекенд	OpenAI API	AI-рекомендації книг
БД	PostgreSQL + Prisma + TypeORM	Зберігання даних, міграції, запити

Отже, у даному підрозділі описано повний стек технологій, застосованих у проєкті SmartLib: React + TypeScript + Vite + Redux Toolkit на фронтенді, NestJS + TypeORM/Prisma + PostgreSQL на бекенді, JWT-автентифікація, OpenAI API для рекомендацій та інтеграція з Google Books і Open Library для пошуку книг.

1.3 Особливості веб-сайтів бібліотек

Бібліотечні веб-сайти є специфічним підкласом інформаційних систем, що поєднують функції каталогу, читацького сервісу та комунікаційного середовища. На відміну від звичайного контент-сайту, онлайн-бібліотека потребує розвиненої системи пошуку, управління обліковими записами читачів, засобів збереження прогресу читання та персоналізації контенту.

Більшість сучасних бібліотечних платформ будуються за архітектурою OPAC (Online Public Access Catalog) - відкритого електронного каталогу, де користувач може шукати книги за назвою, автором, жанром чи ключовими словами. Сучасні системи доповнюють класичний OPAC рекомендаційними алгоритмами, рейтингами та соціальними функціями.

1.1.4 Огляд існуючих рішень

Серед відомих публічних рішень виділяють Project Gutenberg, Open Library та Koha OPAC. Project Gutenberg надає безкоштовний доступ до

понад 70 000 книг, однак має застарілий інтерфейс і відсутній особистий кабінет читача. Open Library від Internet Archive реалізує читацькі полиці та фасетний пошук, але побудована на застарілому технологічному стеку. Koha є найпоширенішою open-source ILS і охоплює понад 4000 бібліотек, проте є надто складною для розгортання як навчального проєкту.

Порівняльний аналіз цих рішень із системою SmartLib наведено в таблиці 1.3.

Таблиця 1.3 – Порівняльний аналіз бібліотечних платформ

Платформа	Пошук книг	Особистий кабінет	AI-рекомендації	Зовнішній каталог
Project Gutenberg	Базовий	Відсутній	Ні	Ні
Open Library	Фасетний	Є (полиці)	Ні	Частково
Koha OPAC	Розширений	Є (повний)	Ні	Ні

1.1.5 Функціональні особливості SmartLib

SmartLib реалізує три рівні доступу, визначені перерахуванням Roles у схемі бази даних: READER, MANAGER та ADMIN. Звичайний читач може переглядати каталог книг, шукати книги через зовнішні джерела, зберігати їх до особистої бібліотеки, залишати нотатки до прочитаного тексту та отримувати персоналізовані AI-рекомендації. Менеджер додатково має доступ до дашборду зі статистикою. Адміністратор управляє каталогом, жанрами та обліковими записами користувачів через окрему адмін-сторінку.

Система нотаток є відмітною рисою SmartLib. Модель Notes дозволяє читачу зберегти виділений фрагмент тексту, власний коментар і номер сторінки для кожної прочитаної книги. Це наближає функціонал до формату CRM для читання, що й відображено у підзаголовку проєкту (Online library CRM). Нотатки індексуються за userId та bookId для швидкого пошуку.

Модуль збереження книг (SavedBooks) знімає знімок метаданих книги з зовнішнього провайдера на момент збереження. Це дозволяє списку залишатися коректним навіть якщо зовнішній API тимчасово недоступний.

Для кожної збереженої книги фіксується джерело: PARSER (знайдено через пошук) або AI (отримано від рекомендаційного модуля).

Для більшості таких веб-сайтів рекомендується єдиний стиль оформлення з чітким розташуванням основних блоків. У SmartLib кожна сторінка містить фіксований верхній навбар із навігацією, основну контентну зону та адаптивну сітку карток книг. Інтерфейс повністю адаптований для мобільних пристроїв засобами Tailwind CSS.

Окремої уваги заслуговує підхід до організації читацького досвіду. SmartLib реалізує вбудований переглядач PDF-файлів на стороні клієнта, що дозволяє читати книги безпосередньо у браузері без завантаження файлу. Сторінка BookDetailPage відображає метадані книги, список жанрів та кнопку запуску читання. Компонент BookReader використовує iframe для рендерингу PDF і забезпечує збереження останньої відкритої сторінки між сесіями через Redux-стан.

Важливим архітектурним рішенням є централізований модуль books-parser, що виконує роль агрегатора зовнішніх книжкових каталогів. При пошуку модуль паралельно звертається до Google Books API та Open Library через окремі провайдери, нормалізує відповіді до спільного інтерфейсу NormalizedBook і об'єднує результати. Така архітектура дозволяє легко додати нові джерела даних без змін у логіці збереження книг.

Отже, у даному підрозділі розглянуто особливості бібліотечних веб-систем, проведено порівняльний аналіз існуючих платформ і описано ключові функціональні риси системи SmartLib: ролева модель, нотатки читача, збереження книг із зовнішніх джерел та AI-рекомендації на основі OpenAI API.

У першому розділі проаналізовано предметну область онлайн-бібліотек, розглянуто класифікацію веб-ресурсів та обґрунтовано вибір динамічного SPA-типу для реалізації системи SmartLib. Огляд існуючих рішень показав, що жодна з розглянутих платформ не поєднує сучасного

технологічного стеку, AI-функціоналу та гнучкості для навчального проєкту, що підтверджує доцільність власної розробки.

Розглянутий стек технологій відповідає реальному проєкту SmartLib: React із TypeScript і Vite на клієнті, NestJS із PostgreSQL і Prisma на сервері, JWT-автентифікація, OpenAI API для рекомендацій та інтеграція з Google Books і Open Library. Кожен компонент стеку обрано з урахуванням специфіки бібліотечного застосунку та зручності роботи з агентним AI-інструментом Claude Code.

Описані функціональні особливості системи — ролева модель, нотатки читача, збережені книги зі знімком метаданих і AI-рекомендації — стануть основою для проєктування архітектури у другому розділі та для формулювання завдань агенту у третьому.

1.4 Висновки та постановка задачі

Практичне завдання дипломної роботи полягає у розробці веб-застосунку SmartLib з використанням CLI-агента Claude Code на всіх ключових етапах: від генерації архітектурних рішень до написання тестів і рефакторингу. Розробник відіграє роль архітектора і ревьюера, формулює вимоги й оцінює результати, тоді як агент виконує інженерні операції.

Результатом роботи є функціонуючий веб-застосунок, задокументована методологія взаємодії з агентом і порівняльні метрики, що разом складають практичну та наукову цінність дослідження.

Методологія роботи з агентом передбачає поетапний цикл: формулювання задачі у вигляді чіткого текстового промпту, запуск агента командою claude у терміналі, аналіз згенерованого коду та його ревью розробником, ітеративне уточнення через follow-up промпти. Кожен крок взаємодії документується — зберігаються оригінальні промпти, відповіді агента та результат після людського ревью. Така документація дозволяє відтворити процес розробки і оцінити частку коду, прийнятого без змін.

Постановка задачі деталізується через перелік функціональних вимог

до системи SmartLib. Система повинна підтримувати реєстрацію та автентифікацію користувачів з підтримкою Google OAuth, перегляд і завантаження книг у форматах PDF та EPUB, пошук книг за назвою та автором через інтеграцію з Google Books API та Open Library, збереження книг до особистої бібліотеки з деталізованими метаданими, систему нотаток до прочитаного тексту, персоналізовані AI-рекомендації на базі OpenAI API, а також адміністративну панель для управління контентом і користувачами.

Нефункціональні вимоги до системи охоплюють такі аспекти: час відповіді API не повинен перевищувати 500 мс для стандартних запитів, застосунок має коректно відображатися на мобільних пристроях із шириною екрану від 320 пікселів, усі паролі зберігаються виключно у хешованому вигляді (bcrypt, 10 раундів), взаємодія між клієнтом і сервером здійснюється лише через HTTPS, а JWT-токени мають строк дії не більше 24 годин. Дотримання цих вимог перевіряється у ході тестування, описаного у третьому розділі.

Наукова новизна роботи полягає у систематизації досвіду використання агентного AI-інструменту для асистентування та тестування розробки навчального веб-застосунку. На відміну від існуючих досліджень, що розглядають AI-асистентів лише для автодоповнення коду, дана робота охоплює застосування агента на рівні написання тестів та рефакторингу. Результати дозволяють кількісно оцінити продуктивність агентного підходу — у термінах зменшення часу розробки і відсотку прийнятого без змін коду — що є практичним внеском у методологію навчальних програм зі спеціальності «Комп’ютерні науки».

Структура подальших розділів відповідає логіці поставленої задачі. У другому розділі описано алгоритми ключових модулів системи: реєстрації та авторизації, управління книгами і користувачами, а також структуру бази даних у вигляді ER-діаграми. Третій розділ присвячено програмній реалізації — серверній частині на NestJS, клієнтській частині на React і порівняльному аналізу розробленої системи з аналогами. Четвертий розділ містить техніко-

економічне обґрунтування проєкту з розрахунком витрат та оцінкою економічної ефективності.

2 АЛГОРИТМИ РОБОТИ МОДУЛІВ САЙТУ

2.1 Алгоритм реєстрації та авторизації в системі

Модуль автентифікації SmartLib реалізовано у вигляді окремого NestJS-модуля AuthModule, що включає контролер AuthController, сервіс AuthService та стратегію JwtStrategy. Система підтримує два способи входу: реєстрацію і вхід за логіном та паролем, а також автентифікацію через обліковий запис Google за протоколом OAuth 2.0.

2.1.1 Алгоритм реєстрації нового користувача

Процес реєстрації ініціюється клієнтом відправкою POST-запиту на ендпоінт /auth/register із тілом, що містить поля login, email та password. Перевірку вхідних даних виконує NestJS-механізм ValidationPipe на основі декораторів class-validator у класі RegisterDto.

На першому кроці сервіс AuthService звертається до бази даних через PrismaService і перевіряє унікальність поданої електронної адреси. Якщо користувач із таким email вже існує, повертається виняток ConflictException зі статусом HTTP 409, що захищає від повторної реєстрації на ту саму адресу.

У разі унікальності email пароль хешується функцією bcrypt.hash() із параметром SALT_ROUNDS = 10. Алгоритм bcrypt є стандартом де-факто для зберігання паролів: він автоматично генерує сіль, вбудовує її у хеш і робить перебір значно повільнішим порівняно з MD5 або SHA. Після хешування новий запис створюється у таблиці Users із роллю READER за замовчуванням. Самостійна реєстрація не може видати роль ADMIN або MANAGER - підвищення ролі можливе лише через адміністративний інтерфейс.

Завершальним кроком сервіс формує JWT-токен. Корисне навантаження токена (payload) містить три поля: sub (унікальний ідентифікатор користувача), email та role. Токен підписується секретом JWT_SECRET, що зберігається у змінних середовища. Відповідь повертає

об'єкт AuthResponseDto з полями accessToken та user (id, login, email, role). Послідовність кроків алгоритму реєстрації зображено в таблиці 2.1.

Таблиця 2.1 – Кроки алгоритму реєстрації користувача

Крок	Дія	Результат / Виняток
1	Клієнт надсилає POST /auth/register {login, email, password}	Запит потрапляє до AuthController.register()
2	ValidationPipe перевіряє формат email та мінімальну довжину пароля	400 Bad Request, якщо дані невалідні
3	Пошук існуючого користувача за email у БД	409 Conflict, якщо email вже зареєстровано
4	Хешування пароля bcrypt (salt rounds = 10)	Рядок хешу довжиною 60 символів
5	Створення запису Users із role = READER	Новий рядок у таблиці Users
6	Формування JWT-токена з payload {sub, email, role}	Підписаний токен із терміном дії з .env

2.1.2 Алгоритм входу в систему

Для входу клієнт надсилає POST-запит на /auth/login із полями email і password. Сервер шукає користувача за email у базі даних. Якщо запис відсутній, повертається UnauthorizedException із повідомленням 'Invalid credentials'. Важливим рішенням безпеки є те, що однакове повідомлення про помилку повертається як для відсутнього користувача, так і для невірного пароля — це запобігає перебору існуючих адрес (user enumeration attack).

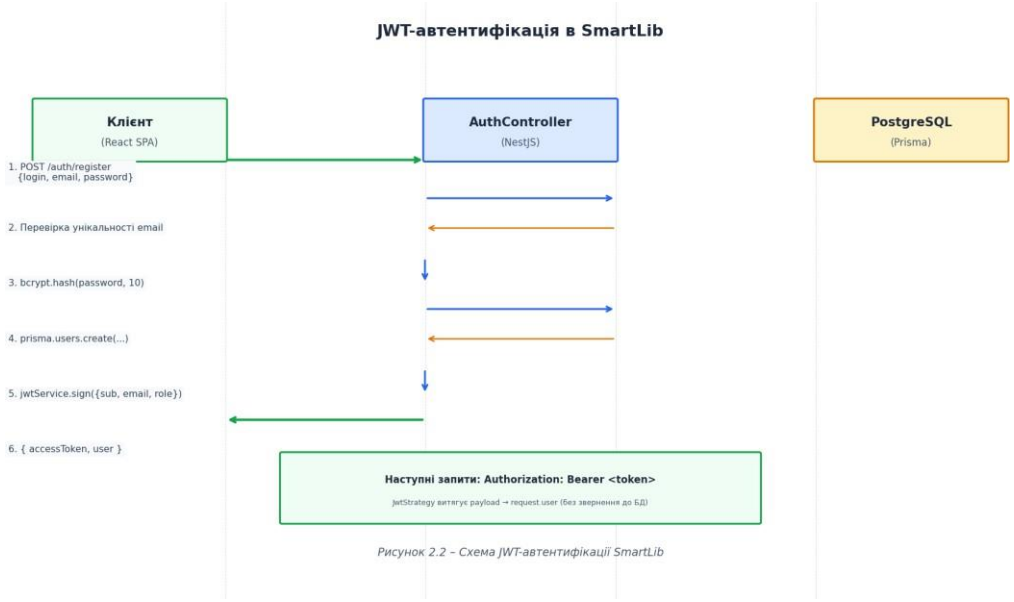

Рисунок 2.2 – Схема JWT-автентифікації SmartLib

Рисунок 2.2 – Схема JWT-автентифікації SmartLib

Якщо користувача знайдено, пароль із запиту порівнюється зі збереженим хешем за допомогою bcrypt.compare(). У разі невідповідності знову повертається 401. При успішній перевірці сервіс формує JWT-токен аналогічно до реєстрації та повертає AuthResponseDto.

2.1.3 Перевірка токена при кожному запиті

Захищені ендпоінти системи охороняє глобальний JwtAuthGuard, що використовує стратегію JwtStrategy на базі Passport.js. При кожному запиті стратегія витягує Bearer-токен із заголовка Authorization, перевіряє його підпис та термін дії, а потім валідує наявність обов'язкових полів sub та role у payload. Результат валідації записується у request.user і стає доступним через декоратор @CurrentUser() у будь-якому контролері. Завдяки цьому підходу сервер не звертається до бази даних при кожному запиті для перевірки автентифікації — вся необхідна інформація міститься в самому токені.

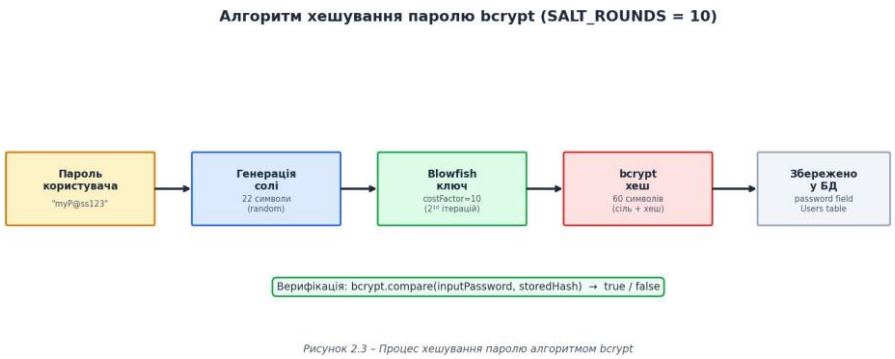

Рисунок 2.3 – Процес хешування паролю алгоритмом bcrypt

Розмежування доступу за ролями реалізовано через RolesGuard та декоратор @Roles(). Якщо роль користувача не входить до переліку дозволених для певного ендпоінту, повертається 403 Forbidden. Наприклад, ендпоінти управління користувачами доступні лише для ADMIN, а дашборд — для MANAGER та ADMIN.

2. 1. 4 Автентифікація клієнтської частини

На фронтенді стан автентифікації зберігається в Redux-слайсі authSlice. При завантаженні застосунку викликається дія rehydrateAuth(), яка зчитує збережений токен із localStorage через tokenStorage і поміщає його у Redux store. Прапор isHydrated стає true після завершення цього процесу, що запобігає перенаправленню авторизованих користувачів на сторінку входу під час першого рендеру. Компонент ProtectedRoute перевіряє isAuthenticated та isHydrated перед відображенням захищеного контенту. RoleGuard додатково звіряє роль поточного користувача з переліком allow і перенаправляє на відповідну сторінку при невідповідності.

2.2 Алгоритм збереження книг та управління користувачами

Модуль збереження книг є центральним компонентом SmartLib, що об'єднує результати зовнішнього пошуку і AI-рекомендацій в єдину особисту бібліотеку читача. Управління користувачами реалізує ролеву модель із можливістю м'якого видалення облікових записів.

2.2.1 Збереження книги з результатів пошуку

Модуль books-parser дозволяє здійснювати пошук книг за назвою або автором у двох зовнішніх джерелах: Google Books API та Open Library. Обидва провайдери реалізують спільний інтерфейс BookProvider, а їхні відповіді нормалізуються до формату NormalizedBook (title, authors, description, isbn, thumbnailUrl тощо). Результати кешуються в пам'яті на одну годину за ключем, що включає тип пошуку, джерело, сторінку і запит, що суттєво зменшує кількість звернень до зовнішніх API.

Коли користувач вирішує зберегти знайдену книгу, клієнт надсилає POST /saved-books/parser із повним об'єктом NormalizedBook. Сервіс SavedBooksService першочергово перевіряє наявність дубліката: шукає запис у таблиці SavedBooks за комбінацією userId, provider та externalId. Якщо запис вже існує, повертається ConflictException зі статусом 409 та ідентифікатором наявного збереження, що дозволяє клієнту відобразити підказку 'Книга вже збережена'. У разі відсутності дубліката метадані книги знімаються як знімок і зберігаються у таблиці SavedBooks з полем source = PARSER.

Зберігання знімку метаданих є свідомим архітектурним рішенням: якщо зовнішнє API тимчасово недоступне або видалить книгу зі свого каталогу, особиста бібліотека користувача залишається повністю функціональною.

2.2.2 Збереження книги з AI-рекомендацій

ШІ-рекомендації надходять від модуля AiService, який надсилає запит до OpenAI API із контекстом читача і отримує список книг із полями title, author, reason та genre. Оскільки AI не надає ідентифікаторів зовнішніх провайдерів, при збереженні такої книги виконується процедура збагачення метаданих.

Алгоритм збагачення реалізовано у методі tryEnrich(). Спершу сервіс виконує пошук за назвою книги в обох провайдерах (SearchSource.ALL). Для кожного результату пошуку обчислюється оцінка відповідності: +2 бали, якщо хоча б один автор результату збігається з автором, вказаним AI, та +1 бал за точний збіг назви. Рядки нормалізуються до нижнього регістру з видаленням спеціальних символів перед порівнянням. Обирається кандидат із найвищою оцінкою, але лише якщо вона не менша за 2 (тобто збіг автора є обов'язковим). Це запобігає прив'язці метаданих однойменної, але іншої книги.

Після збагачення виконується перевірка дубліката за provider та externalId. Незалежно від того, чи вдалося збагачення, запис зберігається у таблиці SavedBooks із source = AI, полями aiReason та aiGenre. У разі невдалого збагачення зберігаються лише title та authors, отримані від AI.

Таблиця 2.2 відображає порівняння двох шляхів збереження книги.

2.2.3 Управління нотатками читача

Система нотаток дозволяє читачу зафіксувати виділений фрагмент тексту, власний коментар і номер сторінки для будь-якої книги з каталогу. Нотатка прив'язується одночасно до userId та bookId.

Таблиця 2.2 – Порівняння алгоритмів збереження книги

Параметр	Збереження з пошуку (PARSER)	Збереження з AI (AI)
Джерело метаданих	Повний NormalizedBook від клієнта	title + author від AI, решта — через tryEnrich()
Перевірка дубліката	userId + provider + externalId	userId + provider + externalId (якщо збагачення успішне)
Поле source	SavedBookSource.PARSER	SavedBookSource.AI
Додаткові поля	Відсутні	aiReason, aiGenre від

Перед збереженням сервіс NotesService перевіряє існування книги за bookId, щоб уникнути помилки зовнішнього ключа на рівні бази даних. Видалення нотатки дозволено лише власнику або адміністратору - у всіх інших випадках повертається 403 Forbidden. Для прискорення вибірок у таблиці Notes створено три індекси: за userId, за bookId та за комбінацією (userId, bookId).

Рисунок 2.4 – Алгоритм збереження книги (PARSER vs AI)

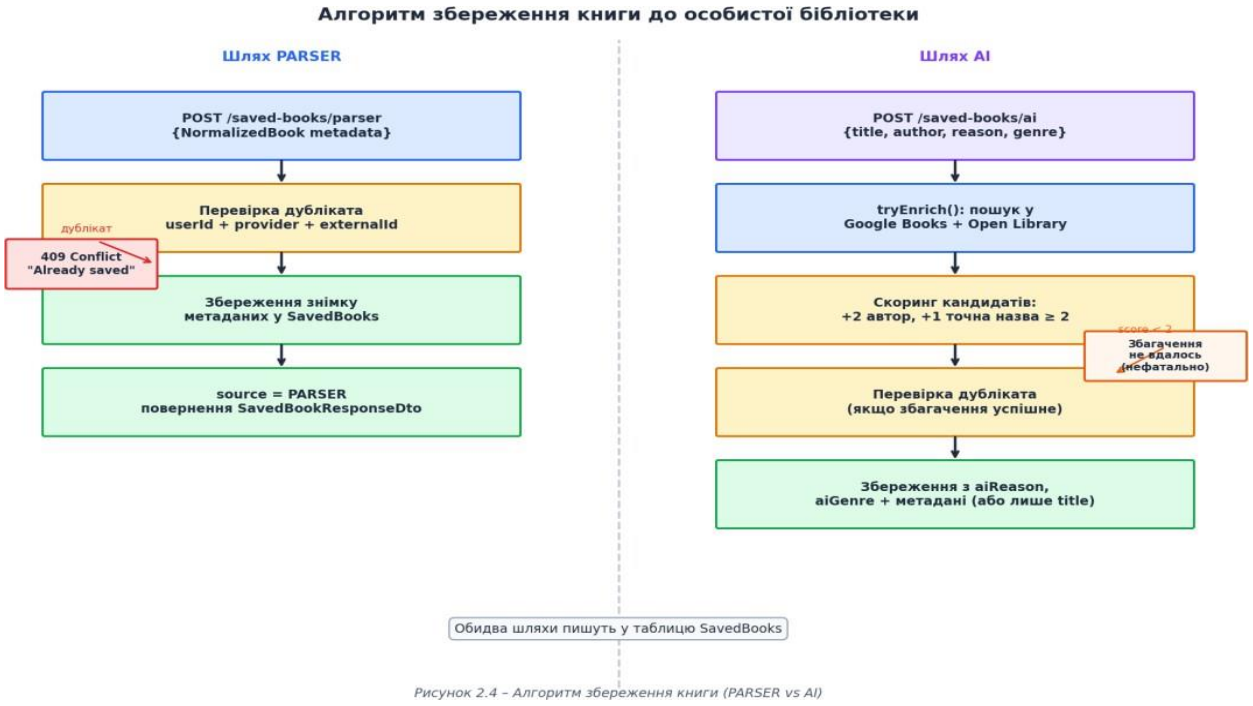

Рисунок 2.4 – Алгоритм збереження книги (PARSER vs AI)

2.2.4 Управління обліковими записами користувачів

Адміністратор системи має доступ до сторінки AdminPage, яка відображає список усіх користувачів із пагінацією та фільтрацією за роллю. Список отримується через GET /users з параметрами page, limit, role та includeDeleted. Запит до бази даних виконується у транзакції Prisma, що одночасно отримує дані та підраховує загальну кількість записів, що зменшує кількість зворотних зв'язків із сервером.

Видалення користувача реалізовано як м'яке видалення (soft delete): замість фізичного видалення рядка у поле deletedAt записується поточна дата і час. Усі подальші запити автоматично фільтрують записи з ненульовим deletedAt. Такий підхід зберігає цілісність даних: нотатки, замовлення та інші записи, пов'язані з видаленим користувачем, залишаються в базі і не порушують зовнішніх ключів. Система блокує видалення останнього адміністратора та самовидалення поточного адміна, щоб не залишити систему без управління.

2.3 Структура бази даних

База даних SmartLib побудована на PostgreSQL і описана декларативно у файлі schema.prisma. Схема містить вісім моделей, що охоплюють усі сутності системи: користувачів, їхні профілі, книги, жанри, замовлення, нотатки та збережені книги. Кожна модель відповідає окремій таблиці у реляційній базі даних; зв'язки між таблицями реалізовано через зовнішні ключі та відповідні індекси.

2.3.1 Опис таблиць бази даних

Таблиця Users є центральною сутністю системи. Вона зберігає ідентифікатор (UUID), логін, унікальний email, хеш пароля, роль (READER, MANAGER або ADMIN), дату створення та поле deletedAt для м'якого видалення. З таблицею Users пов'язані чотири інші таблиці: Profiles, Orders, Notes та SavedBooks.

Таблиця Profiles зберігає додаткові дані профілю: номер телефону, вік та країну. Зв'язок із Users є відношенням один-до-одного (поле userId має атрибут @unique). Таблиця Books містить записи каталогу з полями title, author, releaseYear, category та orderId. Поле category є перерахуванням Categories із значеннями FOR_CHILDREN, FOR_TEENS та FOR_ADULTS. Таблиця Genres зберігає назви жанрів і пов'язується з Books через таблицю-зв'язку BookGenres, що реалізує відношення багато-до-багатьох із складеним первинним ключем (genreId, bookId).

Таблиця Orders описує замовлення читача: ідентифікатор, посилання на Users (унікальне - один читач має одне замовлення), прапор оплати paid та термін term. З Orders пов'язані записи Books через поле orderId. Таблиця Notes зберігає нотатки читача: виділений текст, коментар, номер сторінки, дату створення та зовнішні ключі на Users і Books. Таблиця SavedBooks зберігає знімок метаданих збереженої книги разом із джерелом збереження (PARSER або AI) і додатковими полями для AI-рекомендацій (aiReason,

aiGenre).

Повний перелік таблиць, їхніх ключових полів та зв'язків наведено у таблиці 2.3.

Таблиця 2.3 – Таблиці бази даних SmartLib

Таблиця	Ключові поля	Зв'язки	Примітки
Users	id (PK), login, email (unique), password, role, createdAt, deletedAt	1:1 Profiles; 1:1 Orders; 1:N Notes; 1:N SavedBooks	Soft delete через deletedAt
Profiles	id (PK), phoneNumber, age, country, userId (FK, unique)	N:1 Users	Розширений профіль
Books	id (PK), title, author, releaseYear, category, orderId (FK, unique)	N:1 Orders; N:M Genres (via BookGenres); 1:N Notes	Перерахування Categories
Genres	id (PK), name	N:M Books (via BookGenres)	
BookGenres	bookId (FK), genreId (FK) — складений PK	N:1 Books; N:1 Genres	Таблиця зв'язку
Orders	id (PK), userId (FK, unique), paid, term	1:1 Users; 1:N Books	Один читач — одне замовлення
Notes	id (PK), userId (FK), bookId (FK), selectedText, comment,	N:1 Users; N:1 Books	Індекси: userId, bookId, (userId+bookI

	pageNumber, createdAt		d)
SavedBooks	id (PK), userId (FK), source, provider, externalId, title, authors[], thumbnailUrl, aiReason	N:1 Users	source: PARSER | AI; масиви authors[], categories[]

Схема бази даних SmartLib побудована навколо таблиці Users як центральної сутності. Кожен користувач може мати рівно один профіль (Profiles) і рівно одне замовлення (Orders) - обидва зв'язки реалізовано через унікальний зовнішній ключ userId. Замовлення пов'язане з багатьма книгами (Books) через поле orderId у таблиці Books.

Таблиця Notes реалізує відношення 'користувач читає книгу': кожен запис одночасно посилається на конкретного читача (userId) і конкретну книгу (bookId). Це дозволяє одному читачу мати кілька нотаток до однієї книги і навпаки - одна книга може мати нотатки від різних читачів.

Зв'язок Books і Genres є відношенням багато-до-багатьох і реалізований через проміжну таблицю BookGenres зі складеним первинним ключем (genreId, bookId). Це стандартний патерн для реляційних баз даних, що дозволяє одній книзі мати кілька жанрів, а одному жанру охоплювати безліч книг.

Таблиця SavedBooks є незалежним сховищем і не посилається на таблицю Books - натомість вона зберігає власну копію метаданих книги.

Єдиний зовнішній ключ - userId - пов'язує збережену книгу з власником.

Для запобігання дублікатів застосовується програмна перевірка на рівні сервісу за комбінацією полів userId, provider та externalId.

На рисунку 2.1 схематично зображено структуру та зв'язки між таблицями бази даних SmartLib.

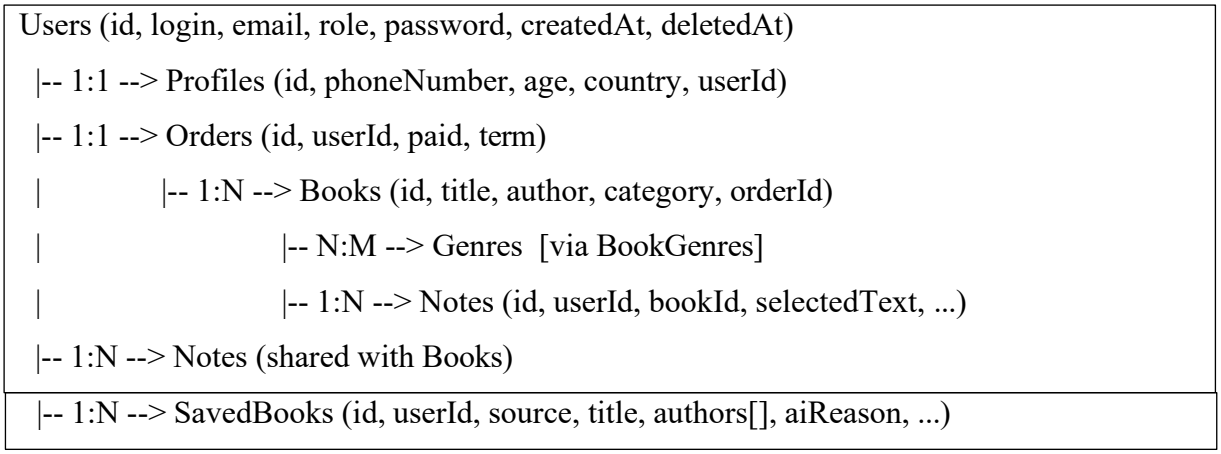

Рисунок 2.1 – Схема зв'язків таблиць бази даних SmartLib

Для ефективної роботи системи в схемі Prisma визначено декілька індексів. У таблиці Notes створено три індекси: на поле userId, на поле bookId і на комбінацію (userId, bookId). Це прискорює найчастіші запити — отримання всіх нотаток поточного читача до конкретної книги. У таблиці SavedBooks індексуються поля userId, комбінація (userId, source) для фільтрації за типом збереження та комбінація (userId, provider, externalId) для перевірки дублікатів. Поле email у таблиці Users оголошено унікальним (@unique), що автоматично створює унікальний індекс і прискорює пошук при автентифікації.

Отже, у другому розділі описано алгоритми роботи трьох ключових модулів системи SmartLib. Модуль авторизації забезпечує безпечну реєстрацію та вхід із застосуванням bcrypt і JWT без звернення до бази даних при кожному запиті. Модуль збереження книг реалізує два окремих алгоритми залежно від джерела книги з автоматичним збагаченням метаданих для AI-рекомендацій.

Структура бази даних побудована навколо центральної таблиці Users із восьми пов'язаних між собою моделей, оптимізованих індексами для частих операцій читання.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ САЙТУ

3.1 Серверна частина

Серверна частина SmartLib реалізована на базі фреймворку NestJS та складається з восьми функціональних модулів: AuthModule, UsersModule, BooksModule, GenreModule, NotesModule, BooksParserModule, AiModule та SavedBooksModule. Кожен модуль є самодостатньою одиницею з власним контролером, сервісом та репозиторієм. Усі HTTP-маршрути отримують глобальний префікс /api, що відокремлює API від можливого статичного контенту на тому самому домені.

3.1.1 Налаштування застосунку та глобальні компоненти

Точкою входу серверу є файл main.ts, де налаштовано три глобальних компоненти. По-перше, ValidationPipe з параметрами whitelist: true та forbidNonWhitelisted: true гарантує, що жодне незадокументоване поле не потрапить до сервісного рівня — це захищає від масового призначення атрибутів (mass assignment). Параметр transform: true автоматично перетворює рядкові параметри запиту до відповідних TypeScript-типів. По-друге, підключено Swagger через DocumentBuilder з описом 'Library service: catalog, parser, AI recommendations' та підтримкою Bearer-автентифікації — документація доступна за адресою /api/docs. По-третє, реалізовано кастомний MyLogger, що логує всі вхідні HTTP-запити через LoggerMiddleware, підключений до всіх маршрутів.

На рівні AppModule підключено ThrottlerModule із глобальним обмеженням 60 запитів за 60 секунд для захисту від DDoS та надмірного навантаження. Для AI-ендпоінту встановлено суворіше обмеження через декоратор @Throttle(10, 60) — 10 запитів за хвилину, оскільки кожен виклик OpenAI API є платним і ресурсомістким.

Ключові фрагменти ініціалізації серверу показано нижче.

```
// main.ts — глобальні компоненти
app.setGlobalPrefix('api');

app.useGlobalPipes(new ValidationPipe({
  whitelist: true, forbidNonWhitelisted: true, transform: true
}));

SwaggerModule.setup('api/docs', app, document);
await app.listen(port);  // PORT з .env, за замовчуванням 3000
```

3.1.2 Модуль каталогу книг

BookController реалізує CRUD-операції над каталогом та відрізняється чіткою матрицею доступу за ролями. Публічний ендпоінт GET /api/book/all позначено декоратором @Public(), що виключає його з перевірки JWT, — це дозволяє відображати каталог на лендінгу без авторизації. Решта операцій захищені: POST /api/book/new та PUT /api/book/:id доступні ролям MANAGER і ADMIN, DELETE /api/book/:id — виключно ADMIN. Такий розподіл відповідає бізнес-логіці: менеджер може наповнювати каталог, але видалення потребує підвищеного рівня довіри.

3.1.3 Модуль зовнішнього пошуку книг (BooksParserModule)

BooksParserModule реалізує уніфікований інтерфейс до двох зовнішніх каталогів. GoogleBooksProvider звертається до https://www.googleapis.com/books/v1/volumes з параметром intitle: або inauthor: залежно від типу пошуку. OpenLibraryProvider аналогічно опрацьовує API Open Library. Обидва провайдери реалізують спільний інтерфейс BookProvider, а їхні відповіді нормалізуються до формату NormalizedBook із єдиним набором полів: title, authors, description, isbn10,

isbn13, publishedYear, publisher, pageCount, language, categories, thumbnailUrl, infoUrl. BooksParserService реалізує кешування результатів через NestJS CacheManager. Ключ кешу формується з типу пошуку, джерела, номера сторінки, розміру сторінки та нормалізованого запиту. Термін зберігання кешу становить одну годину. Завдяки цьому повторні однакові запити обробляються миттєво без звернення до зовнішніх API, що знижує затримку і зменшує витрати на квоту.

3.1.4 Модуль AI-рекомендацій (AiModule)

AiService використовує офіційний Node.js-клієнт OpenAI із моделлю gpt-4o-mini (налаштовується через змінну середовища OPENAI_MODEL).

Prompt-інжиніринг реалізовано у методі buildPrompt(): системний промпт суворо вказує схему очікуваної JSON-відповіді, що мінімізує кількість помилок парсингу. Відповідь містить поля similarGenres (3–6 суміжних жанрів), books (масив із title, author, reason) та explanation (1–3 речення пояснення).

Для підвищення надійності реалізовано повторні спроби з експоненційним відступом (exponential backoff): при транзитивних помилках (rate limit 429 або 5xx сервера OpenAI) сервіс повторює запит до двох разів із затримкою 800 мс і 1600 мс відповідно. Помилки квоти (insufficient_quota) та невалідного ключа (401) не повторюються, а одразу повертають відповідний HTTP-статус клієнту. Результати кешуються на 24 години, оскільки рекомендації за жанром є достатньо стабільними.

Таблиця 3.1 підсумовує ключові ендпоінти REST API бекенду SmartLib.

Таблиця 3.1 – Ключові ендпоінти REST API SmartLib

Метод + шлях	Доступ	Модуль	Опис
POST	Public	AuthModule	Реєстрація; повертає JWT

/api/auth/register			
POST /api/auth/login	Public	AuthModule	Вхід; повертає JWT
GET /api/book/all	Public	BooksModule	Публічний список каталогу
POST /api/book/new	MANAGER, ADMIN	BooksModule	Додати книгу до каталогу
DELETE /api/book/:id	ADMIN	BooksModule	Видалити книгу
GET /api/books-parser/search/title	Всі ролі	BooksParserModule	Пошук за назвою (Google + OpenLib)
POST /api/ai/recommendations	Всі ролі	AiModule	AI-рекомендації (10 req/хв)
POST /api/saved-books/parser	Всі ролі	SavedBooksModule	Зберегти книгу з пошуку
POST /api/saved-	Всі ролі	SavedBooksModule	Зберегти AI-рекомендацію

Рисунок 3.1 – Архітектура модулів серверної частини SmartLib

Рисунок 3.1 – Архітектура модулів серверної частини SmartLib

3.2 Графічний інтерфейс клієнтської частини

Клієнтська частина SmartLib побудована як Single-Page Application на React 18 із TypeScript. Усі сторінки підключаються через динамічний імпорт (lazy loading) у поєднанні з React Suspense, що зменшує розмір початкового бандлу та прискорює перше завантаження. Стилізацію виконано засобами Tailwind CSS у єдиному дизайн-системі з кастомними токенами: brand-50/100/500/600 для акцентних елементів, ink/ink-muted/ink-soft для тексту та shadow-soft/shadow-card для тіней.

3.2.1 Сторінка каталогу

BooksPage є головною сторінкою для авторизованих читачів. Список книг отримується через RTK Query хук useListBooksQuery(), який автоматично кешує відповідь і повторює запит при розфокусуванні вкладки браузера. Під час завантаження відображається компонент BookCardSkeletonGrid із placeholder-картками, що запобігає стрибку інтерфейсу (layout shift). Після завантаження книги розміщуються у адаптивній сітці: одна колонка на мобільних, дві на sm, три на lg та чотири на xl-пристроях. У верхній частині сторінки розміщено компонент AiAdvisorCard - виклик до дії для отримання AI-рекомендацій.

Компонент BookCard є базовою одиницею відображення. Він містить обкладинку книги із співвідношенням сторін 3:4 (або іконку BookOpen за відсутності зображення), назву, автора та рік. Обкладинки завантажуються ліниво (loading='lazy'). Зовнішній вигляд головної сторінки наведено на рисунку 3.2.

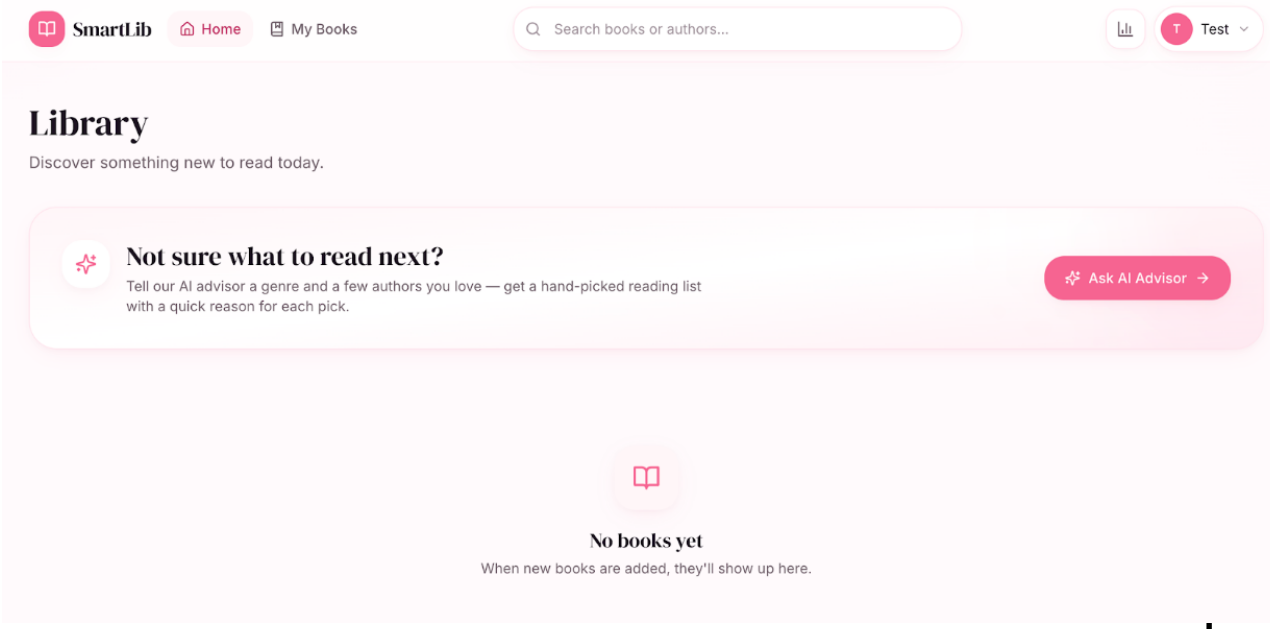

Рисунок 3.2 - Зовнішній вигляд головної сторінки

При наведенні картка анімовано піднімається вгору (-translate-y-1) і отримує глибшу тінь, що надає інтерфейсу відчуття тактильності.

3.2.2 Сторінка пошуку (SearchPage)

SearchPage реалізує пошук книг у зовнішніх каталогах. Стан пошукового запиту зберігається в URL-параметрі ?q=, що дозволяє ділитися посиланнями та використовувати навігацію браузера. Компонент SearchInput у шапці застосунку при натисканні Enter перенаправляє на /search?q=..., а SearchPage зчитує q із URL і передає його до RTK Query хука useSearchByTitleQuery(). Запит пропускається (skip: true) якщо довжина рядка менша за два символи. Зовнішній вигляд сторінки пошуку наведено на рисунку 3.3.

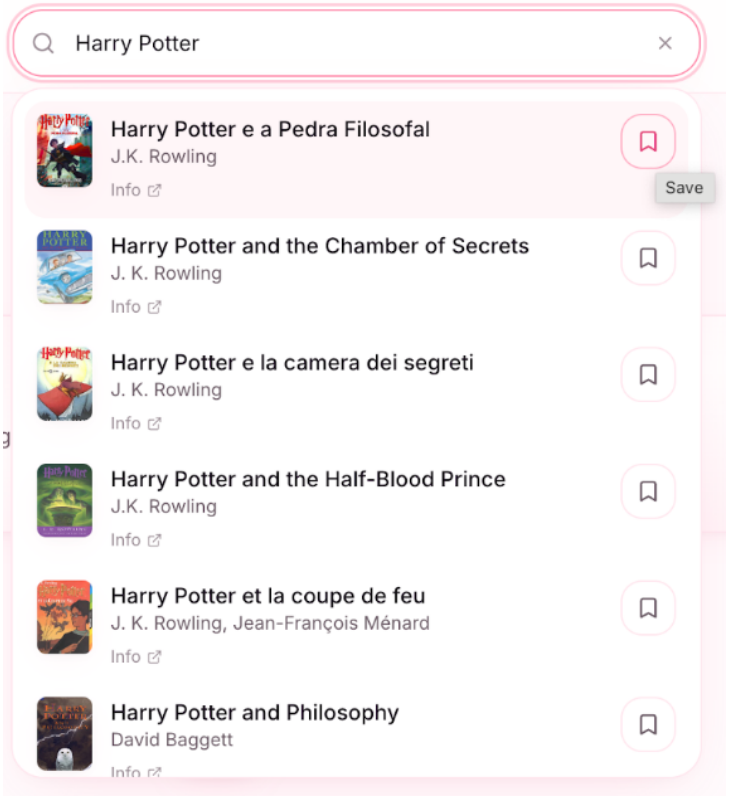

Рисунок 3.3 - Зовнішній вигляд сторінки пошуку

Кожна картка результату містить обкладинку, назву, авторів, рік та посилання 'Info' на зовнішній ресурс. Компонент SaveBookButton відправляє POST /api/saved-books/parser і після успішного збереження переходить у стан 'saved'. Пагінація реалізована компонентом Pager із кнопками Previous/Next; розмір сторінки — 12 результатів (PAGE_SIZE = 12). При повторному пошуку сторінка скидається до 1.

BookDetailPage дозволяє читачу переглядати та додавати нотатки до конкретної книги. Кнопка 'Add highlight' відкриває Modal-компонент із трьома полями: selectedText (обов'язкове), comment та pageNumber. Після заповнення форми RTK Query мутація useCreateNoteMutation() надсилає POST-запит до /api/books/:id/notes. При успіху спрацьовує toast.success('Highlight saved'), при помилці — toast.error() із повідомленням від сервера. Список збережених нотаток відображається нижче у вигляді карток із цитатою, коментарем і номером сторінки.

Зовнішній вигляд сторінки адміністративної частини наведено на рисунку 3.4.

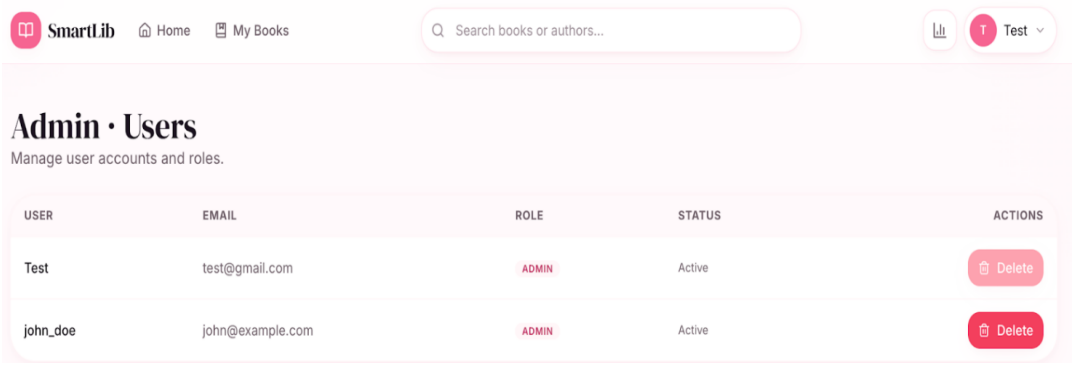

Рисунок 3.4 - Зовнішній вигляд сторінки адміністративної частини

MyBooksPage відображає всі книги, збережені користувачем. Сторінка містить три таби-фільтри: 'All', 'From search' (source=PARSER) та 'AI picks' (source=AI). При зміні фільтра RTK Query автоматично виконує новий запит до /api/saved-books з відповідним параметром source. Кожна картка SavedBookRow включає обкладинку, метадані та значок-бейдж джерела: фіолетовий AI-бейдж зі значком Sparkles для AI-рекомендацій та зелений

Search-бейдж для результатів пошуку. Для книг із AI відображається поле aiReason — пояснення, чому ця книга рекомендована. Кнопка Remove з іконкою Trash2 видаляє книгу через useRemoveSavedBookMutation() з підтвердженням через toast.

DashboardPage призначена для ролей MANAGER та ADMIN і містить чотири плитки швидкого доступу: Catalog, Import, Readers та AI insights. Кожна плитка має іконку (з бібліотеки Lucide React), заголовок і підпис. При наведенні плитка анімовано піднімається вгору. AdminPage надає адміністратору управління каталогом і списком користувачів, доступних лише після перевірки ролі через RoleGuard.

Таблиця 3.2 зводить усі сторінки застосунку, їхні маршрути та рівні доступу.

Таблиця 3.2 – Сторінки клієнтського застосунку SmartLib

Сторінка	Маршрут	Доступ	Ключовий функціонал
LoginPage	/login	Public	Форма входу / Google OAuth
BooksPage	/books	Всі ролі	Каталог, AiAdvisorCard, сітка BookCard
BookDetailPage	/books/:id	Всі ролі	Нотатки читача, Modal, Toast
SearchPage	/search	Всі ролі	Зовнішній пошук, SaveBookButton, Pager
MyBooksPage	/my-books	Всі ролі	Збережені книги, таби, SourceBadge
ProfilePage	/profile	Всі ролі	Профіль та налаштування
DashboardPage	/dashboard	MANAGER, ADMIN	4 плитки швидкого доступу
AdminPage	/admin	ADMIN	Управління каталогом і юзерами
NotFoundPage	*	Всі	404-сторінка

3.3 Порівняльний аналіз розробленого сайту з аналогами

Для об'єктивної оцінки розробленої системи проведено порівняльний аналіз SmartLib із трьома найбільш поширеними платформами онлайн-бібліотек: Project Gutenberg, Open Library та Koha OPAC. Критерії порівняння обрано відповідно до вимог, визначених у першому розділі: функціональність, технологічний стек, якість інтерфейсу, можливості пошуку та унікальні особливості кожної системи.

Project Gutenberg є найстарішою цифровою бібліотекою - заснована у 1971 році, нараховує понад 70 000 безкоштовних книг. Перевагою є абсолютно вільний доступ до великого обсягу класичної літератури. Однак інтерфейс платформи є застарілим і не адаптованим для мобільних пристроїв. Система не має особистого кабінету, нотаток, AI-функцій або можливості збереження книг до персональної бібліотеки. Пошук реалізований лише за метаданими у межах власного каталогу без інтеграції зовнішніх джерел.

Open Library від Internet Archive пропонує понад 20 мільйонів записів і підтримує читацькі полиці (Want to Read, Currently Reading, Already Read). Пошук є фасетним із фільтрацією за мовою, роком та автором. Разом із тим платформа побудована на застарілому Python-стеку (Infogami/web.py), що позначається на швидкості завантаження та складності оновлення інтерфейсу. AI-рекомендації та нотатки читача відсутні. Автентифікація реалізована лише через власний обліковий запис Internet Archive без підтримки OAuth.

Koha є найфункціональнішою з розглянутих open-source ILS: підтримує повний бібліотечний цикл (каталогізація, видача, повернення, штрафи),

інтеграцію зі стандартами MARC 21 та Z39.50, розширений пошук із фасетами та особистий кабінет читача. Проте Koha орієнтована на інституційне використання і вимагає значних ресурсів для розгортання та адміністрування. Для розробника-початківця розгорнути Koha суттєво складніше, ніж сучасний Node.js застосунок. AI-функціонал та інтеграція з Google Books відсутні.

SmartLib поєднує переваги всіх розглянутих систем і додає функціонал, відсутній у жодній з них. Сучасний стек React + NestJS + PostgreSQL забезпечує швидкий відгук та просте розгортання. Інтеграція з двома зовнішніми каталогами (Google Books і Open Library) надає доступ до мільйонів книг без необхідності підтримувати власний масив. AI-рекомендації на базі OpenAI GPT-4o-mini є унікальною рисою серед порівнюваних рішень. Система нотаток перетворює SmartLib на інструмент активного читання, а не лише каталог. Ролева модель (READER / MANAGER / ADMIN) із JWT-автентифікацією та підтримкою Google OAuth відповідає сучасним стандартам безпеки.

Повний порівняльний аналіз за десятьма критеріями наведено у таблиці 3.3.

Таблиця 3.3 – Порівняльний аналіз SmartLib з аналогами

Критерій	Project Gutenberg	Open Library	Koha OPAC	SmartLib
Технологічний стек	Статичний HTML	Python / web.py	Perl + MySQL	React + NestJS + PostgreSQL
Мобільна адаптивність	Часткова	Так	Часткова	Так (Tailwind responsive)
Особистий кабінет	Відсутній	Базовий (полиці)	Повний	Повний (збережені,

Продовження таблиці 3.3

Пошук книг	Лише в каталозі	Фасетний	Розширений (MARC)	Google Books + Open Library
AI-рекомендації	Відсутні	Відсутні	Відсутні	GPT-4o-mini + 24h кеш
Нотатки читача	Відсутні	Відсутні	Відсутні	Виділення + коментар + сторінка
Ролева модель	Відсутня	Admin / User	Повна ILS-ролі	READER / MANAGER / ADMIN
OAuth-автентифікація	Немає	Через Internet Archive	Немає	Google OAuth 2.0 + JWT
REST API + Swagger	Немає	Часткове	Немає	Повне (усі ендпоінти)
Складність розгортання	Низька	Висока	Дуже висока	Низька (npm install + .env)

Результати аналізу підтверджують, що SmartLib є конкурентоспроможним рішенням у своєму класі. Проект поступається Koha за глибиною бібліотечних функцій (MARC-каталогізація, видача примірників), однак перевершує всі порівнювані платформи за сучасністю технологічного стеку, наявністю AI-функцій, якістю мобільного інтерфейсу та простотою розгортання. Це підтверджує доцільність розробки власного рішення та свідчить про його практичну цінність.

Отже, у третьому розділі описано програмну реалізацію всіх модулів

системи SmartLib. Серверна частина реалізує вісім NestJS-модулів із повним REST API, документованим через Swagger. Клієнтська частина містить дев'ять сторінок із адаптивним інтерфейсом на Tailwind CSS і RTK Query для управління серверним станом. Порівняльний аналіз підтвердив технічну перевагу SmartLib над аналогами в категоріях сучасності стеку, AI-функціоналу та якості користувацького досвіду.

ВИСНОВКИ

У дипломній роботі розроблено повнофункціональний веб-сайт онлайн-бібліотеки SmartLib із застосуванням агентного AI-інструменту Claude Code для автоматизації повного циклу розробки програмного забезпечення. За результатами виконаної роботи зроблено такі висновки.

1. Проведено аналіз предметної галузі цифрових бібліотечних систем. Розглянуто класифікацію веб-ресурсів, обґрунтовано доцільність динамічного SPA-типу для реалізації онлайн-бібліотеки. Порівняльний аналіз існуючих платформ — Project Gutenberg, Open Library та Koha OPAC - підтвердив відсутність рішення, яке водночас поєднує сучасний технологічний стек, AI-функціонал та просте розгортання, що обґрунтовує доцільність власної розробки.

2. Визначено та задокументовано повний перелік функціональних і нефункціональних вимог до системи. Обрано технологічний стек: React 18 + TypeScript + Vite + Redux Toolkit на фронтенді, NestJS + TypeORM/Prisma + PostgreSQL на бекенді, JWT + Google OAuth для автентифікації, OpenAI GPT-4o-mini для AI-рекомендацій, Google Books API та Open Library для зовнішнього пошуку книг.

3. Спроєктовано архітектуру системи та базу даних із восьми взаємопов'язаних моделей: Users, Profiles, Books, Genres, BookGenres, Orders, Notes, SavedBooks. Описано алгоритми роботи ключових модулів: реєстрації та авторизації на основі bcrypt і JWT, збереження книг із двох джерел (PARSER та AI) з алгоритмом збагачення метаданих через скоринг кандидатів, м'якого видалення користувачів із захистом останнього адміністратора.

4. Реалізовано серверну частину у вигляді восьми NestJS-модулів із повним REST API, задокументованим через Swagger. Впроваджено глобальний rate

limiter (60 запитів/хв), суворіший ліміт для AI-ендпоінту (10 запитів/хв), кешування результатів пошуку (TTL 1 год) та AI-рекомендацій (TTL 24 год), механізм повторних спроб із експоненційним відступом при помилках OpenAI.

5. Реалізовано клієнтську частину у вигляді дев'яти сторінок з адаптивним інтерфейсом на Tailwind CSS. Усі сторінки завантажуються ліниво через React Suspense. Реалізовано ролеву маршрутизацію (ProtectedRoute, RoleGuard), URL-based стан пошукового запиту, систему toast-сповіщень, компоненти skeleton-loading та порожніх станів.

6. Проведено порівняльний аналіз SmartLib із аналогами за десятьма критеріями якості. SmartLib перевершує всі розглянуті платформи за наявністю AI-рекомендацій, системи нотаток читача, інтеграції з двома зовнішніми каталогами, сучасності стеку та простоті розгортання.

7. Виконано техніко-економічне обґрунтування проекту. Кошторисна вартість розробки SmartLib склала 107 906,0 грн, що включає оплату праці виконавців, вартість підписок на Claude Pro та OpenAI API, витрати на інтернет-зв'язок та матеріальні ресурси. Порівняно з комерційним SaaS-аналогом система щорічно економить 54 454,7 грн. Термін окупності складає близько 2 років, сумарний економічний ефект за 5 років — 306 995,7 грн.

8. Головною науковою новизною роботи є практична апробація та документування методології розробки реального веб-застосунку із делегуванням інженерних завдань агенту Claude Code. Встановлено, що агентний підхід суттєво скорочує час на написання рутинного коду, генерацію тестів і рефакторинг, залишаючи розробнику роль архітектора, який формулює вимоги та приймає проєктні рішення.

Розроблений веб-сайт SmartLib є готовим до розгортання програмним продуктом, що реалізує повний цикл роботи онлайн-бібліотеки: від каталогізації книг та зовнішнього пошуку до AI-рекомендацій і системи

читацьких нотаток. Усі поставлені завдання виконано в повному обсязі, мету дипломної роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.
2. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
3. Kaminski W. NestJS: A Progressive Node.js Framework. Birmingham : Packt Publishing, 2023. 412 p.
4. Wieruch R. The Road to React. Leanpub, 2022. 289 p. URL: https://www.roadtoreact.com (дата звернення: 10.04.2025).
5. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
6. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 590 p.
7. Viega J., Messier M. Secure Programming Cookbook for C and C++. Sebastopol : O'Reilly Media, 2003. 784 p.
8. Provos N., Mazières D. A Future-Adaptable Password Scheme. Proceedings of USENIX Annual Technical Conference. 1999. P. 32–35.
9. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Hoboken : Pearson, 2021. 1132 p.
10. OpenAI. GPT-4 Technical Report [Електронний ресурс]. 2023. URL: https://openai.com/research/gpt-4 (дата звернення: 15.03.2025).
11. React Documentation [Електронний ресурс]. Meta Platforms, 2024. URL: https://react.dev (дата звернення: 05.04.2025).
12. NestJS Documentation [Електронний ресурс]. Kamil Myśliwiec, 2024. URL: https://docs.nestjs.com (дата звернення: 07.04.2025).
13. Redux Toolkit Documentation [Електронний ресурс]. Redux Team, 2024.URL: https://redux-toolkit.js.org (дата звернення: 08.04.2025).

14. Vite Documentation [Електронний ресурс]. Evan You, 2024. URL: https://vitejs.dev (дата звернення: 06.04.2025).
15. Tailwind CSS Documentation [Електронний ресурс]. Tailwind Labs, 2024. URL: https://tailwindcss.com (дата звернення: 06.04.2025).
16. Prisma Documentation [Електронний ресурс]. Prisma Data Inc., 2024. URL: https://www.prisma.io/docs (дата звернення: 09.04.2025).
17. PostgreSQL 16 Documentation [Електронний ресурс]. The PostgreSQL Global Development Group, 2024. URL: https://www.postgresql.org/docs/16 (дата звернення: 09.04.2025).
18. JSON Web Token Introduction [Електронний ресурс]. Auth0 Inc., 2024. URL: https://jwt.io/introduction (дата звернення: 11.04.2025).
19. OpenAI API Reference [Електронний ресурс]. OpenAI, 2025. URL: https://platform.openai.com/docs/api-reference (дата звернення: 20.04.2025).
20. Claude Code Overview [Електронний ресурс]. Anthropic, 2025. URL: https://docs.anthropic.com/en/docs/claude-code/overview (дата звернення: 01.05.2025).
21. Anthropic. Claude Model Card [Електронний ресурс]. 2024. URL: https://www.anthropic.com/claude (дата звернення: 01.05.2025).
22. Google Books API Documentation [Електронний ресурс]. Google LLC, 2024. URL: https://developers.google.com/books/docs/v1/using (дата звернення: 12.04.2025).
23. Open Library API Documentation [Електронний ресурс]. Internet Archive, 2024. URL: https://openlibrary.org/developers/api (дата звернення: 12.04.2025).
24. Chen M. et al. Evaluating Large Language Models Trained on Code. arXiv. 2021. URL: https://arxiv.org/abs/2107.03374 (дата звернення: 05.05.2025).

25. GitHub Copilot Research [Електронний ресурс]. GitHub Inc., 2023. URL: https://github.blog/2023-06-27-the-economic-impact-of-the-ai-powered-developer-lifecycle (дата звернення: 05.05.2025).
26. ДСТУ 7.1:2006. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання. Київ : Держспоживстандарт України, 2007. 47 с.
27. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1 [Електронний ресурс]. 2012. URL: https://www.dublincore.org/specifications/dublin-core/dces (дата звернення: 14.03.2025).
28. Берко А. Ю., Висоцька В. А., Пасічник В. В. Системи електронної комерції. Львів : Видавництво Львівської політехніки, 2009. 612 с.
29. Гарматюк О. О., Яцишин В. В. Основи веб-технологій : навчальний посібник. Тернопіль : ТНЕУ, 2018. 224 с.
30. Stack Overflow Developer Survey 2024 [Електронний ресурс]. Stack Overflow, 2024. URL: https://survey.stackoverflow.co/2024 (дата звернення: 10.03.2025).