ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку електронної бібліотеки із соціальною взаємодією»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Михаїл ВАРИЧ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Михаїл ВАРИЧ

Керівник: Наталя АНДРУСЕВИЧ
ст.викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Варичу Михаїлу Юрійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку електронної бібліотеки із соціальної взаємодією»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ
Наталя АНДРУСЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки web-застосунків, опис роботи електронних бібліотек, технічна документація фреймворків Spring та Next.js.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі електронних бібліотек.
2. Вибір засобів програмної реалізації.
3. Проєктування архітектури web-застосунку.
4. Програмна реалізація та тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Титульний лист.
2. Мета, об’єкт та предмет дослідження.
3. Задачі кваліфікаційної роботи.
4. Аналіз аналогів.
5. Вимоги до програмного забезпечення.
6. Програмні засоби реалізації.
7. Діаграма варіантів використання.
8. Діаграма архітектури
9. Діаграма класів.
10. Екранні форми.
11. Апробація результатів дослідження.
12. Висновки.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд предметної галузі електронних бібліотек	14.03.-21.03.2025	
4	Огляд та аналіз засобів реалізації web-застосунку електронної бібліотеки із соціальною взаємодії	24.03.-04.04.2025	
5	Проєктування архітектури web-застосунку	07.04.-11.04.2025	
6	Програмна реалізація та тестування web-застосунку	14.04.-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Михаїл ВАРИЧ
(підпис)

Керівник
кваліфікаційної роботи _____ Наталя АНДРУСЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _53_ стор., _1_ табл., _29_ рис., _20_ джерел.

Мета роботи – покращення соціальної взаємодії між користувачами електронних бібліотек шляхом створення клієнт-серверного web-застосунка.

Об'єкт дослідження – процес соціальної взаємодії між користувачами електронних бібліотек.

Предмет дослідження – web-застосунок електронних бібліотек.

Короткий зміст роботи: В роботі проаналізовано предметну галузь та методів реалізації електронних бібліотек. Проаналізовано існуючі рішення електронних бібліотек: Україніка, Лібрук, Аргонавти Всесвіту та УкрЛіб. Розроблено web-застосунок та реалізовано основний функціонал, зокрема: систему авторизації, робота з бібліотекою, коментарі та оцінки до матеріалів, користувацькі профілі, глобальний чат, система ролей.

Проведено модульне тестування додатку. В роботі використано фреймворк Spring Boot, Spring Security, JWT-токени для бекенду, фреймворк Next.js на базі React для фронтенду, базу даних MongoDB.

Сферою використання застосунку є web-застосунки електронних бібліотек.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, SPRING, NEXT.JS, ЕЛЕКТРОННА БІБЛІОТЕКА, СОЦІАЛЬНА ВЗАЄМОДІЯ, СЕРВЕР, КЛІЄНТ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment

API – Applied Programming Interface

SQL – Structured Query Language

JSON – JavaScript Object Notation

CORS – Cross-Origin Resource Sharing

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

SSG – Static Site Generation

SSR – Server-Side Rendering

UI – User Interface

SEO – Search Engine Optimization

PDF – Portable Document Format

ВСТУП

На сьогоднішній день в умовах цифровізації технологій стрімко зростає попит на зручний доступ до знань і літературних ресурсів. Електронні бібліотеки стали важливими інструментами для студентів, викладачів, дослідників та усіх зацікавлених у саморозвитку особах. Водночас більшість наявних рішень обмежуються лише базовим функціоналом – пошуком та переглядом книг, без глибокої інтеграції інструментів для обговорення, взаємодії користувачів або створення читацьких спільнот.

Розробка спеціалізованого web-застосунку електронної бібліотеки, який об’єднує доступ до літературних ресурсів із соціальними функціями є надзвичайно актуальною. Такий підхід дозволяє не лише спростити пошук і читання книг, а й створює простір для комунікації, обміну думками, колективного навчання та підтримки користувачів. Завдяки цьому web-застосунок може стати не просто архівом текстів, а повноцінним соціально-освітнім середовищем.

Зважаючи на зростаючу конкуренцію серед освітніх онлайн-сервісів, розширення функціоналу електронної бібліотеки через додавання інструментів соціальної взаємодії дозволяє підвищити залученість користувачів, персоналізувати досвід та стимулювати регулярне користування платформою. Автоматизація основних процесів, таких як пошук, рекомендації, читання та обговорення літератури, сприяє ефективнішому використанню інформаційних ресурсів. Впровадження чату, де користувачі можуть у режимі реального часу вести обговорення та ділитися обраними матеріалами, допоможе соціальній складовій набути кращого рівня досвіду використання таких web-застосунків.

Об’єкт дослідження: процес соціальної взаємодії між користувачами електронних бібліотек.

Предмет дослідження: web-застосунки електронних бібліотек

Мета роботи: покращення соціальної взаємодії між користувачами електронних бібліотек шляхом створення клієнт-серверного web-застосунка.

Для досягнення мети необхідно виконати наступні задачі:

Огляд застосунку електронної бібліотеки – огляд існуючих рішень web-застосунків електронних бібліотек, визначення їхнього основного функціоналу, переваг і недоліків; Проєктування web-застосунку – вибір засобів реалізації, проєктування архітектури клієнтської та серверної частин, визначення бази даних.

Реалізація функціоналу – система авторизації, робота із матеріалами бібліотеки, взаємодія між користувачами через матеріали, глобальний чат, користувацькі профілі, система ролей.

Наукова новизна роботи полягає у використанні живого чату, що дозволяє користувачам спілкуватися та обмінюватися предметами бібліотеки у режимі реального часу.

1 ОГЛЯД ЗАСТОСУНКУ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ

1.1 Web-застосунок електронної бібліотеки

Електронні бібліотеки стали важливим інструментом збереження, поширення та популяризації знань у цифрову епоху. Їхнє основне завдання — надати користувачам доступ до інформаційних ресурсів та зберігати їх у більш обширному електронному форматі[1]. Однією з ключових технологій, що забезпечує таку доступність, є реалізація таких систем у вебпросторі.

Web-застосунок електронної бібліотеки - це програмне забезпечення, що функціонує у браузері та дозволяє користувачам взаємодіяти з цифровими колекціями: переглядати книги, шукати за каталогами, зберігати обране, завантажувати файли, писати відгуки тощо. Такий підхід усуває потребу в локальному встановленні програм і забезпечує мультиплатформенність - доступ можливий з комп’ютера, планшета або смартфона.

Сучасні електронні бібліотеки відіграють важливу роль у забезпеченні рівного доступу до знань та освітніх ресурсів. Вони особливо актуальні в умовах онлайн-навчання, коли користувачі потребують надійного джерела інформації у зручному цифровому форматі. Такі бібліотеки також сприяють розвитку читацької культури, популяризації літератури, науки та мистецтва серед широких верств населення.

Окрему увагу заслуговує роль електронних бібліотек у збереженні національної культурної спадщини. За їх допомогою можна не лише підтримувати та поширювати вже класичні роботи українських авторів, а й зберігати маловідомі роботи, здатні надати унікальну, невідому раніше інформацію. Деякі бібліотеки навіть мають тематичні добірки, присвячені українському фольклору, історії, діаспорі чи мовознавству. У час, коли все українське набуває більшого поширення

та цікавості зі сторони населення, важливо зберігати елементи української історії та культури та поширювати їх серед українців.

Окрім доступу до літератури, низка електронних бібліотек пропонує додатковий функціонал, що сприяє розвитку спільнот читачів і підтримці освітніх ініціатив. Сучасні електронні бібліотеки можуть реалізовувати функції, що дозволяють учасникам не лише споживати контент, але й взаємодіяти між собою через коментарі, голосування, обговорення, особисті повідомлення чи участь у спільних подіях. Такі елементи, як персональні профілі, рейтинги учасників та спільні обговорення, підвищують відчуття причетності та довіри до спільноти. Повторна соціальна взаємодія, видимість активностей інших користувачів та підтримка комунікації в реальному часі або асинхронно сприяють створенню так званого соціотехнічного капіталу, який накопичується у вигляді стосунків, довіри та готовності до співпраці [2]. Ефективне впровадження таких механізмів дозволяє перетворити бібліотеку на справжній осередок спілкування, обміну досвідом і формування спільних цінностей навколо літератури та знань.

Важливим аспектом також є можливість для авторів і дослідників розміщувати власні праці у відкритому доступі. Такі бібліотеки створюють платформу для популяризації нових імен в літературі, поширення результатів наукових досліджень або художніх текстів серед ширшої аудиторії. Це забезпечує двосторонню комунікацію: не лише читач отримує доступ до контенту, а й автор має змогу знайти свого читача, отримати зворотний зв’язок та сформувати власну аудиторію. Таким чином, електронна бібліотека стає не лише цифровим архівом, а повноцінною культурно-інформаційною екосистемою.

1.2 Цільова аудиторія

Електронні бібліотеки охоплюють декілька видів аудиторії, зацікавленої у їхньому використанні. Серед них можна виділити:

– учнів та студентів, що використовують електронні бібліотеки для навчання та пошуку інформації для власних робіт;

– науковців, які переглядають бібліотеки у пошуку матеріалів для власних досліджень та поширюють свої праці;

– викладачів, що шукають матеріали для викладання власних дисциплін;

– бібліотекарів, що працюють над організацією та адмініструванням освітніх та наукових установ;

– звичайних користувачів, зацікавлених у окремих темах.

1.3 Аналіз аналогів

На сьогоднішній день існує різноманітна кількість електронних бібліотек. Для проведення аналізу аналогів web-застосунку обрано такі бібліотеки, як Україніка, Лібрук, Аргонавти Всесвіту та УкрЛіб. Розглянемо їх детальніше.

1.3.1 Україніка

«Україніка» - це національна електронна бібліотека, створена Національною бібліотекою України імені В.І. Вернадського»**[**3]. Вона містить цифрові копії рідкісних, історичних, наукових і культурних документів, що мають значення для національної спадщини. Україніка орієнтована передусім на науковців, дослідників та студентів, які займаються українознавством.

Бібліотека охоплює велику кількість матеріалів: книги, періодичні видання, архівні документи, нотні видання, карти тощо. Користувачі можуть шукати документи за різними параметрами, переглядати їх у сканованому вигляді та завантажувати у форматах PDF і DjVu.

Разом з тим, «Україніка» практично не підтримує елементи соціальної взаємодії. Відсутні можливості для коментування, оцінювання документів, створення персоналізованих добірок або обговорення матеріалів, що робить її більш архівною платформою, аніж інтерактивним середовищем.

Основні можливості:

– пошук за каталогами, автором, темою, роком видання;

– перегляд сканованих копій документів;

– тематична класифікація фондів;

– архівування матеріалів у PDF чи DjVu форматі.

Переваги:

– висока наукова цінність джерел;

– доступ до рідкісних історичних документів;

– безкоштовне використання.

Недоліки:

– застарілий інтерфейс;

– обмежена інтерактивність.

Екранна форма застосунку представлена на рисунку 1.1.

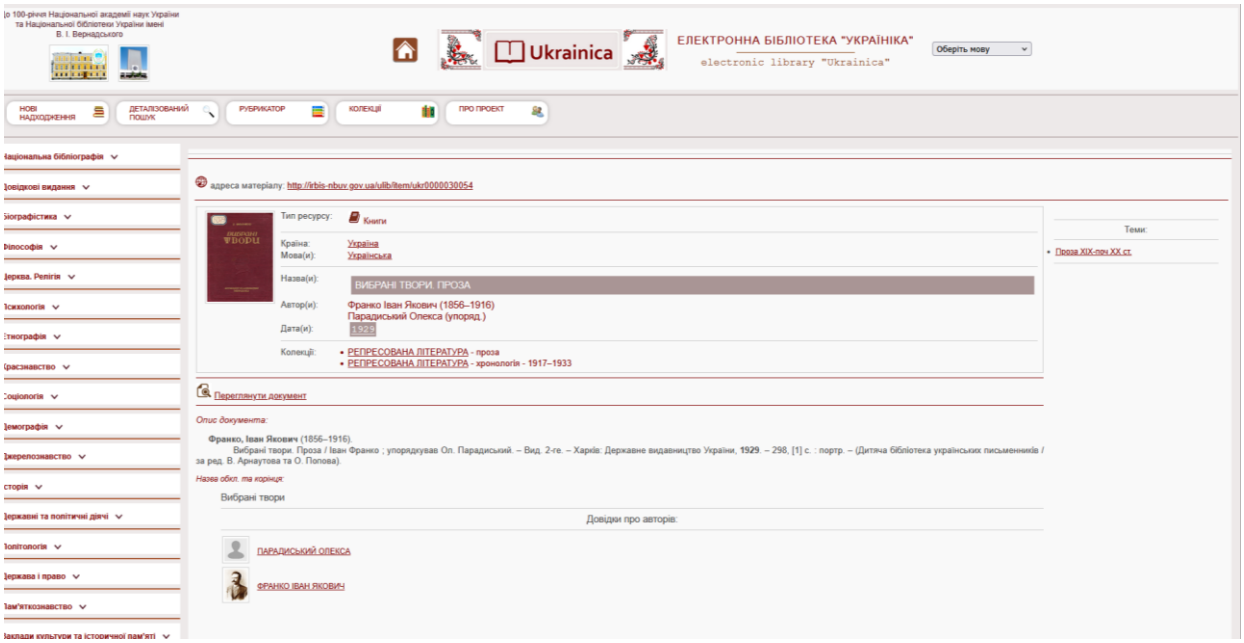


Рис. 1.1 Екранна форма електронної бібліотеки «Україніка»

1.3.2 Лібрук

«Лібрук» - це електронна бібліотека української художньої літератури, створена ентузіастами[4]. Вона орієнтована на популяризацію українського читання, літературної спадщини та сучасної прози й поезії. Платформа функціонує як агрегатор та читач онлайн.

На відміну від академічних платформ, «Літрук» реалізує елементи соціальної взаємодії: користувачі можуть залишати коментарі, брати участь у обговореннях, оцінювати книги, додавати власні твори, об’єднуватися у спільні проєкти та волонтерські ініціативи (наприклад, допомога в оцифруванні). Це формує навколо платформи спільноту активних читачів і авторів.

Основні можливості:

– онлайн-читання книг без реєстрації;

– каталогізація за жанрами та авторами;

– функція завантаження книг у форматі fb2, epub, txt.

Переваги:

– простий, швидкий доступ до українських книжок;

– висока якість верстки електронних книг.

Недоліки:

– невелика кількість наукових видань;

– обмежені можливості пошуку.

Екранна форма застосунку представлена на рисунку 1.2.

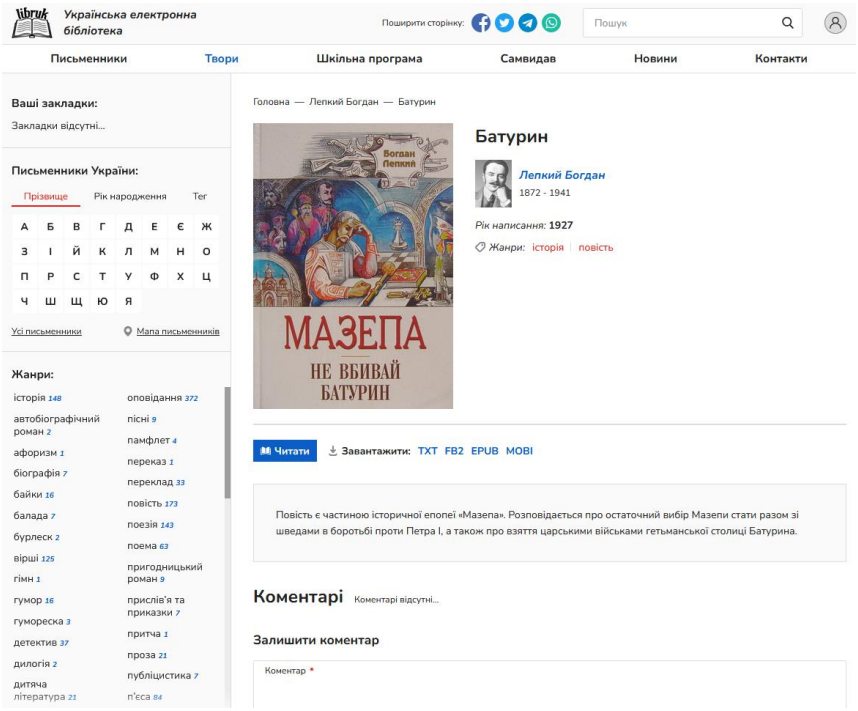


Рис. 1.2 Екранна форма української електронної бібліотеки «Літрук»

1.3.3 Аргонавти Всесвіту

«Аргонавти Всесвіту» - це спеціалізована електронна бібліотека україномовної фантастики[5]. Вона об’єднує твори як відомих, так і маловідомих авторів, пов’язані з жанром фантастики. Представлена у вигляді форуму. Реалізовано можливість публікації творів, коментування, голосування, участі у конкурсах, спілкування через форум і чат. Аудиторія — аматори та автори фантастики. Платформа підтримується волонтерами, структура побудована у вигляді форуму та каталогу. Структура форуму може бути складною для нових користувачів, що ускладнює навігацію по розділах.

Основні можливості:

– пошук творів за автором та видавництвом;
– рецензії та обговорення користувачів;
– публікація власної творчості.

Переваги:

– унікальний контент — україномовна фантастика;
– функціонал для формування активної спільноти;
– можливість зворотного зв’язку з авторами.

Недоліки:

– застарілий інтерфейс.

Екранна форма застосунку представлена на рисунку 1.3.

Рис. 1.3 Екранна форма застосунку «Аргонавти Всесвіту – сайт україномовної фантастики»

1.3.4 УкрЛіб

УкрЛіб - це популярна електронна бібліотека, яка спеціалізується на творах української класичної та сучасної літератури[6]. Особливо корисна для школярів та студентів, адже містить не лише художні твори, а й біографії авторів, короткий зміст, аналіз, характеристики героїв тощо.

Сайт має простий і швидкий у навігації інтерфейс, алфавітний покажчик, поділ за темами і жанрами. Водночас платформа не передбачає можливостей для коментування, оцінювання творів чи будь-якої соціальної взаємодії між користувачами.

Основні можливості:

– велика база української літератури;
– авторські довідки, аналізи творів, плани-конспекти;
– зручний алфавітний пошук і тематичні розділи;
– можливість завантажити будь-який матеріал.

Переваги:

– одна з найповніших електронних бібліотек української літератури;
– зручний інтерфейс для учнів та викладачів;
– освітня орієнтованість.

Недоліки:

– відсутність соціальної взаємодії.

Екранна форма застосунку представлена на рисунку 1.4.

Рис. 1.4 Екранна форма бібліотеки української літератури «УкрЛіб».

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу характеристик додатків електронних бібліотек

Показник	Україніка	Лібрук	Аргонавти Всесвіту	УкрЛіб	ЛібХабЮа
Перегляд матеріалів бібліотеки	+	+	+	+	+
Система авторизації	–	+	+	–	+
Робота із наповненням бібліотеки (додавання, видалення, редагування)	–	+	+	–	+
Взаємодія між користувачами через матеріали бібліотеки (коментування, оцінка, обрані матеріали);	–	+/- (відсутня оцінка)	+	–	+
Глобальний чат	–	–	–	–	+
Користувацькі профілі	–	+	+	–	+
Функціонал для експертів	–	–	–	–	+
Функціонал для адміністраторів	–	–	–	–	+

1.4. Постановка задачі

Метою даної роботи є створення web-застосунку, що виконує функції електронної бібліотеки, та забезпечує соціальну взаємодію між користувачами через наповнення бібліотеки. Проаналізувавши предметну галузь, визначивши цільову аудиторію та провівши аналіз вже існуючих рішень для існуючих бібліотек, для успішного виконання роботи необхідно виконати наступні задачі:

Аналіз засобів реалізації: Визначити середовище розробки, мову та технології для реалізації застосунку.

Проєктування застосунку: Обрати архітектуру застосунку, визначити структуру внутрішніх компонентів та бази даних.

Функціональна реалізація застосунку: Реалізувати наступні функціональні вимоги:

– перегляд матеріалів бібліотеки;
– система авторизації користувачів;
– робота із наповненням бібліотеки (додавання матеріалів, редагування та видалення);
– взаємодія між користувачами через матеріали бібліотеки (коментування, оцінка матеріалів, система обраних матеріалів);
– глобальний чат, що дає змогу у режимі реального часу спілкуватися з авторизованими користувачами та ділитися матеріалами бібліотеки;
– користувацькі профілі з функціоналом для редагування користувачів, відображення обраних матеріалів та матеріалів, створених користувачем;
– функціонал для запрошених експертів (написання особливих коментарів до предметів, відображення наукових досягнень у профілі);
– функціонал для адміністраторів (зміна ролей, видалення предметів, користувачів та коментарів).

Нефункціональна реалізація застосунку: Реалізувати наступні нефункціональні вимоги:

– шифрування паролів;
– захист від поширених видів атак;
– робота без критичних помилок із можливістю відновлення;
– можливість розширення системи;
– завантаження сторінок не більше, ніж 2-3 секунди.

2 ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) – це програмне забезпечення, що надає програмістам інструментарій для написання програмного коду. Зазвичай ці середовища містять у собі текстовий редактор, інструменти для відладки, збірки та запуску коду та інтеграцію із системами контролю версій. Ці засоби стали предметом першої необхідності для програмістів при розробці програм, що дозволяють ефективно писати та тестувати своє програмне забезпечення[7].

2.1.1 IntelliJ IDEA

IntelliJ IDEA — це одне з найпопулярніших середовищ розробки програмного забезпечення, створена компанією JetBrains. Вона спеціалізується на мовах програмування, зокрема Java, Kotlin та інших, і є надзвичайно потужним інструментом для розробки додатків будь-якої складності. IntelliJ IDEA надає розширену підтримку для Java-проєктів, підтримує інтеграцію систем контролю версій, велику кількість допоміжних бібліотек, а також підтримку тестування, збірки та запуску програм прямо з середовища[8].

Інтелектуальні підказки, автоматичне завершення коду, навігація по проєкту, рефакторинг, перевірка якості коду — усе це робить IntelliJ IDEA потужним інструментом у руках розробника. Крім того, завдяки модульній структурі та плагінам, користувачі можуть адаптувати середовище під конкретні задачі. Наприклад, через плагіни можна додати підтримку Spring Boot, Docker, Kubernetes, а також підключити інструменти для роботи з TypeScript, JavaScript, HTML, CSS та іншими технологіями фронтенду. Загалом, IntelliJ IDEA є ідеальним вибором для повного циклу розробки програмного забезпечення на Java та суміжних технологіях.

2.1.2 WebStorm

WebStorm — це середовище для розробки клієнтських застосунків, також створене компанією JetBrains. Воно спеціалізується на підтримці скриптових мов JavaScript, TypeScript, мов розмітки HTML, CSS, а також сучасних фреймворків для розробки фронт-енду, таких як React, Angular, Vue.js та Node.js. WebStorm орієнтоване на веброзробку і надає користувачам усі необхідні інструменти для створення, тестування й налагодження вебдодатків [9].

Серед головних переваг WebStorm — розумне автодоповнення, швидка навігація по проєкту, зручні інструменти для рефакторингу коду, підтримка популярних пакетних менеджерів (npm, yarn), можливість підключення до системи контролю версій, а також вбудований термінал і дебаггер. Крім того, WebStorm має тісну інтеграцію з такими інструментами як ESLint, Prettier та Jest, що дозволяє підтримувати чистоту та якість коду на високому рівні.

2.2 Мови програмування

2.2.1 Java

Java — це одна з найпоширеніших об'єктно-орієнтованих мов програмування, яка була створена у середині 1990-х років і згодом стала власністю Oracle Corporation[10]. Java є кросплатформною завдяки віртуальній машині Java Virtual Machine (JVM), що дозволяє виконувати код на будь-якому пристрої, де встановлено JVM. Цей принцип відомий як "напиши один раз — запускай будь-де" (write once, run anywhere).

Мова відзначається строгим синтаксисом, стабільністю та високим рівнем безпеки, що робить її популярною у корпоративному програмуванні, банківських системах, мобільних застосунках (через Android), наукових обчисленнях та розробці серверних рішень. Java має велику екосистему бібліотек, фреймворків (Spring, Hibernate, Jakarta EE) та інструментів, які спрощують створення масштабованих і надійних програм.

2.2.2 TypeScript

TypeScript — це мова програмування, яка є надмножиною JavaScript і додає до нього статичну типізацію. Вона була розроблена компанією Microsoft і вперше представлена у 2012 році. Основною метою TypeScript є покращення процесу розробки великих проєктів за допомогою введення типів, що дозволяє запобігати багатьом помилкам ще на етапі компіляції[11].

TypeScript має всі можливості JavaScript, оскільки компілюється у звичайний JS-код, який підтримується всіма сучасними браузерами. Проте його головною перевагою є статична типізація, яка дозволяє розробникам чітко визначати типи змінних, параметрів функцій і повернених значень. Це полегшує підтримку великих проєктів, покращує читабельність коду та роботу команд розробників.

2.3 Серверна частина

Серверна частина web-застосунку використовує мову Java та фреймворк, написаний цією мовою, Spring Boot, що є доволі популярним рішенням для побудови web-застосунків.

2.3.1 Фреймворк Spring Boot

Spring Boot — це фреймворк, який базується на фреймворку Spring, але значно спрощує його використання завдяки автоматизованій конфігурації та стартовим залежностям[12]. Spring Boot підтримує RESTful архітектуру, завдяки якій сервер спілкується з клієнтом за допомогою HTTP-запитів, повертаючи дані у форматі JSON. Це забезпечує гнучкість і зручну інтеграцію з фронтендом, створеним на основі JavaScript/TypeScript. Також фреймворк дозволяє легко впроваджувати сторонні бібліотеки, логування, кешування та підтримку безпеки, що робить його ідеальним вибором для проєктів із розгалуженою логікою.

У межах web-застосунку електронної бібліотеки Spring Boot відіграє центральну роль, оскільки забезпечує обробку запитів користувача, з'єднання з базою даних, бізнес-логіку (наприклад, додавання коментарів, голосування за

елементи, перегляд бібліографічних матеріалів), а також керування життєвим циклом сервісів.

2.3.2 Spring Security

Spring Security - це гнучка бібліотека для реалізації функцій безпеки в Spring-додатках[13]. У проєкті електронної бібліотеки вона застосовується для обмеження доступу до ресурсів у залежності від користувацької ролі. Наприклад, лише адміністратор може видаляти будь-який контент, тоді як звичайний користувач може видаляти лише його власний. Spring Security дозволяє гнучко налаштовувати права доступу, створювати фільтри автентифікації, а також інтегрувати різні механізми ідентифікації.

Крім контролю доступу, Spring Security також відповідає за захист від поширених атак, таких як CSRF (міжсайтові запити), XSS (вставка скриптів), brute-force (підбір пароля) та ін. У нашому випадку система автентифікації реалізована з використанням JWT-токенів, і Spring Security обробляє кожен запит, перевіряючи наявність і дійсність токена, перш ніж передати керування відповідному контролеру.

2.3.3 JWT-токени

JWT (JSON Web Token) — це легкий спосіб передачі інформації між сторонами у форматі JSON, зазвичай у вигляді закодованого рядка[14]. У web-застосунку електронної бібліотеки JWT використовується для безсесійної автентифікації. Після авторизації користувач отримує токен, що містить зашифровану інформацію про його ідентифікатор та роль, який зберігається на стороні клієнта.

Під час кожного подальшого запиту клієнт надсилає токен у заголовку Authorization, і сервер перевіряє його дійсність. Це дозволяє уникнути зберігання сесій на сервері, зменшуючи навантаження та спрощуючи масштабування, а також дозволяє розділити функціонал на доступний для неавторизованих користувачів та доступний для тих, що пройшли авторизацію. JWT також дозволяє реалізувати

автоматичне оновлення сесій, перевірку терміну дії та відкликання доступу в разі потреби. У контексті бібліотеки це означає, що авторизовані користувачі можуть мати персоналізований доступ до своїх профілів, історії читання, збережених матеріалів тощо.

2.3.4 WebSocket

WebSocket — це протокол, що забезпечує постійне двостороннє з’єднання між клієнтом і сервером, на відміну від стандартного протоколу HTTP, де клієнт має ініціювати кожен запит[15]. На відміну від класичних HTTP-запитів, які працюють за моделлю "запит-відповідь", WebSocket забезпечує миттєву доставку повідомлень у реальному часі, що є надзвичайно корисним для динамічних web-застосунків. Це дозволяє реалізовувати функції, що працюватимуть у режимі реального часу без потреби оновлювати сторінку, такі як чат чи сповіщення.

Spring Boot надає вбудовану підтримку WebSocket через модулі spring-websocket та spring-messaging, що дозволяє легко налаштувати канали зв’язку, брокери повідомлень і топіки. Застосунок підтримує обмін повідомленнями у форматі JSON, а на клієнті обробка повідомлень здійснюється через бібліотеки на базі STOMP-протоколу.

2.4 Клієнтська частина

Клієнтська частина web-застосунку використовує фреймворк Next.js на основі React, з використанням мови TypeScript та бібліотекою для стилізації Tailwind CSS.

2.4.1 Фреймворк Next.js

Next.js — це фреймворк для створення сучасних web-застосунків на основі React, що надає широкий спектр функцій із коробки: маршрутизацію, серверний рендеринг (SSR), генерацію статичних HTML-сторінок (SSG), API-ендпоінти, оптимізацію продуктивності тощо[16]. Завдяки Next.js, web-застосунки стають

швидкими, зручними для SEO та адаптивними до різних пристроїв. Підтримка SSR особливо корисна для застосунків, які мають велике навантаження або потребують індексації пошуковими системами (наприклад, онлайн-бібліотеки).

У клієнтській частині web-застосунку електронної бібліотеки Next.js використовується як основна технологія побудови інтерфейсу. Він дозволяє створювати логічно розділені сторінки (наприклад, головна, каталог, сторінка книги, профіль користувача) та керувати маршрутами. Компонентний підхід React у поєднанні з можливостями Next.js сприяє створенню повторно використовуваних UI-елементів і зменшує складність підтримки проєкту. Окрім цього, Next.js добре інтегрується з API (через fetch та axios), що дозволяє зручно отримувати дані з бекенду, кешувати відповіді та оновлювати інтерфейс без перезавантаження сторінки.

2.4.2 Tailwind CSS

Tailwind CSS — це утилітарний CSS-фреймворк, який дозволяє швидко створювати адаптивний і сучасний дизайн без написання кастомних стилів[17]. Замість традиційного підходу з оголошенням CSS-класів, Tailwind використовує "класи-утиліти", які описують зовнішній вигляд елементів безпосередньо в HTML або JSX. Це дозволяє розробнику бачити структуру та стиль компонента в одному місці, що покращує продуктивність і читаємість коду.

У web-застосунку електронної бібліотеки Tailwind CSS використовується для побудови інтерфейсу користувача — меню, карток книг, кнопок, форм пошуку, модальних вікон тощо. Фреймворк дозволяє швидко створювати адаптивні сітки, налаштовувати кольорові схеми, тіні, відступи, а також легко підтримує темну тему. У поєднанні з компонентами React або Next.js, Tailwind дає змогу зручно створювати сучасний UI, що відповідає вимогам до зручності, швидкості завантаження та візуальної привабливості. Крім того, завдяки підтримці кастомізації через конфігураційний файл, Tailwind дозволяє адаптувати дизайн під конкретні потреби проєкту.

2.5 База даних

Як базу даних обрано документно-орієнтовану MongoDB. Вона зберігає дані у форматі BSON (розширений JSON), та на відміну від реляційних баз даних, MongoDB не використовує таблиці з чітко визначеними схемами, а натомість оперує колекціями та документами, що робить її гнучкішою у зберіганні та обробці неоднорідних даних[18]. Такий підхід особливо зручний для застосунків, структура даних яких часто змінюється або містить вкладені об’єкти.

У web-застосунку електронної бібліотеки MongoDB використовується як основне сховище для всіх сутностей, таких як користувачі, книги, коментарі, категорії, повідомлення чату тощо. Гнучкість MongoDB дозволяє легко реалізовувати вкладені структури — наприклад, коментарі всередині об'єкта книги або профіль користувача з вбудованими налаштуваннями. Завдяки потужним можливостям фільтрації та агрегації MongoDB також дозволяє ефективно реалізовувати пошук, сортування, фільтри за жанром, автором чи іншими атрибутами.

MongoDB також підтримує індексацію, що дозволяє значно підвищити продуктивність при пошуку даних. У межах електронної бібліотеки індекси застосовуються, наприклад, до полів назв матеріалів, категорій та ідентифікаторів користувачів. Це відіграє важливу роль при роботі з великими обсягами інформації, де швидкість пошуку безпосередньо впливає на зручність користувача. Завдяки підтримці складених індексів, також можна ефективно обробляти запити з кількома критеріями пошуку.

Окремої уваги заслуговує можливість реалізації агрегаційних запитів у MongoDB. Агрегації дають змогу виконувати складні аналітичні операції безпосередньо на рівні бази даних. У системі електронної бібліотеки це, зокрема, використовується для підрахунку кількості коментарів до матеріалів, побудови статистики за категоріями, підрахунку середньої оцінки предмету та визначення найактивніших користувачів. Таким чином, MongoDB виступає не лише як

сховище даних, а й як інструмент для обробки та аналізу інформації у реальному часі.

2.6 Інструменти розробника

Для того, аби перевірити дієздатність застосунку, на допомогу приходять інструменти розробника, які надають функціонал для того, аби перевірити роботу застосунку у реальних умовах.

2.6.1 Docker

Docker — це платформа для контейнеризації, яка дозволяє ізолювати середовище розробки та забезпечити однакову поведінку застосунку на різних етапах життєвого циклу: від локального розроблення до продакшн-середовища. У web-застосунку електронної бібліотеки Docker використовується для створення контейнерів, що містять серверну частину на базі Spring Boot, клієнтську частину на Next.js, а також базу даних MongoDB. Це забезпечує просте розгортання, масштабування та підтримку застосунків.

Інтеграція Docker з IDE, такими як IntelliJ IDEA, дозволяє розробникам керувати контейнерами безпосередньо з середовища розробки, спрощуючи процеси запуску, відладки та моніторингу застосунку[19].

2.6.2 Postman

Postman — це інструмент для тестування API, який дозволяє розробникам створювати, надсилати та аналізувати HTTP-запити до серверної частини застосунку. У проєкті електронної бібліотеки Postman використовується для перевірки коректності RESTful API, реалізованих за допомогою Spring Boot. Це включає тестування авторизації, обробки запитів користувачів, управління контентом тощо.

Postman дозволяє не лише надсилати запити, а й створювати колекції з прикладами типових сценаріїв використання API, які можуть бути повторно використані під час регресійного тестування або при додаванні нового функціоналу. У межах проєкту електронної бібліотеки створено окремі колекції для аутентифікації, управління предметами, додавання коментарів, голосування, чату та адміністративних дій. Кожен запит має приклад тіла запиту (JSON), прикріплені токени авторизації та можливі очікувані відповіді.

Крім того, Postman надає функції моніторингу та тестування у вигляді скриптів, які дозволяють перевіряти валідність відповіді, структуру об’єкта або статус-код. Це дозволяє реалізовувати базову автоматизацію тестування без потреби у складних фреймворках. Збереження результатів тестування у Postman також дозволяє відслідковувати зміни у поведінці API після оновлень, що є важливим для забезпечення стабільності web-застосунку.

3 ПРОЄКТУВАННЯ

3.1. Проєктування архітектури застосунку

Web-застосунок реалізовано за класичною клієнт-серверною архітектурною моделлю, яка передбачає розділення функціональних обов’язків між двома незалежними частинами: клієнтською та серверною. Такий підхід дозволяє забезпечити вищий рівень модульності, забезпечити масштабованість системи, спростити обслуговування програмного забезпечення та покращити розподіл обчислювального навантаження[20]. У межах цієї моделі клієнт-частина відповідає за взаємодію з користувачем, а серверна частина відповідає за обробку бізнес-логіки, зберігання та обробку даних, автентифікацію користувачів та реалізацію інших системних функцій.

Серверна (бек-енд) частина застосунку реалізована з використанням сучасного фреймворку Spring Boot, що є розширенням екосистеми Spring - одного з найпопулярніших фреймворків для побудови Java-застосунків корпоративного рівня. Завдяки Spring Boot вдалося значно спростити налаштування та запуск серверної частини, автоматизувати конфігурацію та зменшити обсяг шаблонного коду. Сервер виконує ключові завдання системи, зокрема: опрацювання HTTP-запитів від клієнта, управління користувацькими сесіями, обробку бізнес-логіки, взаємодію з базою даних (яка реалізована на основі MongoDB), автентифікацію з використанням JWT-токенів, контроль доступу до захищених ресурсів за допомогою Spring Security, а також підтримку двостороннього зв’язку у реальному часі через WebSocket.

Клієнтська (фронт-енд) частина побудована за допомогою сучасного фреймворку Next.js, який базується на React та підтримує гібридні методи рендерингу сторінок — як на клієнті, так і на сервері. Це дозволяє забезпечити швидке завантаження контенту, оптимізовану індексацію сторінок у пошукових

системах та покращену продуктивність. Фронт-енд орієнтований на зручність користувача, відображення динамічного контенту, взаємодію із сервером шляхом обробки запитів REST API, а також реалізацію реактивного та інтуїтивно зрозумілого інтерфейсу. Для стилізації інтерфейсу застосовано утилітарний CSS-фреймворк Tailwind CSS, що дозволяє швидко будувати адаптивні, гнучкі й візуально привабливі компоненти інтерфейсу без необхідності у створенні великої кількості власних стилів.

Фронт-енд і бек-енд обмінюються інформацією у форматі JSON через протокол HTTP, що дозволяє забезпечити уніфіковану та зрозумілу структуру даних. Клієнтська частина ініціює запити до серверу для отримання або зміни інформації (наприклад, створення коментаря, голосування за предмет, оновлення профілю), а сервер, у свою чергу, відповідає обробленими даними або повідомленнями про помилки у випадку порушення правил доступу або валідності.

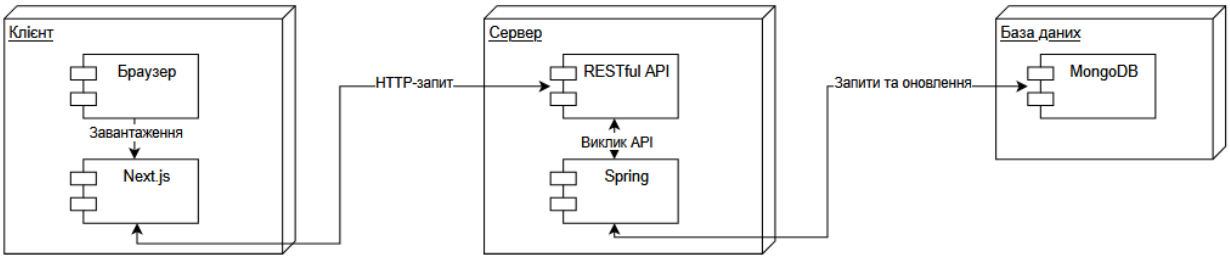


Рис. 3.1 Діаграма розгортання клієнт-серверної моделі застосунку

3.2. Архітектура серверної частини

Серверна частина застосунку побудована на основі фреймворку Spring Boot з дотриманням принципів сервісної архітектури, яка передбачає чітке розділення відповідальностей між окремими шарами системи. Такий підхід забезпечує високу

підтримку коду, гнучкість при масштабуванні функціоналу та спрощує модульне тестування.

Основу серверної логіки складають п’ять ключових пакетів: моделі, запити, сервіси, репозиторії та контролери. У пакеті моделей зберігаються Java-класи, які описують основні сутності застосунку. Ці класи відображають документи в базі даних MongoDB і містять відповідні анотації, такі як @Document (генерація записів у базі даних), @Id (поле з унікальним ідентифікатором), а також інші специфічні позначки для опису полів і зв’язків. Оскільки використовується документо-орієнтована база даних, структура моделей є гнучкою й дозволяє швидко адаптуватися до змін у форматі даних без потреби в складній міграції схеми.

Обмін даними між клієнтом і сервером відбувається через запити та відповіді, реалізовані у вигляді класів записів (Record). Це сучасна форма декларативного визначення immutable (незмінних) структур, яка зменшує кількість шаблонного коду та чітко виражає намір передачі даних без логіки. Кожен Record описує набір полів, необхідних для певної операції, наприклад, створення повідомлення або реєстрації користувача. Завдяки своїй незмінності Record добре підходить для безпечної роботи з HTTP-запитами та зменшує ризик випадкової мутації даних у процесі обробки.

Серцевиною всієї бізнес-логіки є сервісний шар. У пакетах сервісів реалізовано основні дії, які система виконує з даними, включаючи перевірку прав доступу, взаємодію з базою даних через репозиторії, обробку помилок і логіку, що визначає поведінку програми. Сервіси взаємодіють із MongoDB за допомогою технології Spring Data JPA, яка дозволяє працювати з базою даних через репозиторії, реалізовані у вигляді інтерфейсів. Завдяки цьому більшість стандартних CRUD-операцій реалізуються автоматично, що дозволяє зосередитися на написанні лише специфічної логіки, а також надає додаткову безпеку застосунку, автоматично виконуючи SQL-запити без необхідності прописувати їх у сервісах.

Контролери є точкою входу до серверної частини та обробляють HTTP-запити від клієнтів. Вони приймають запити, передають дані у відповідні сервіси

та формують відповіді у форматі JSON. Контролери пов’язані з механізмами валідації, обробки виключень і авторизації. Таким чином, кожен запит, що надходить на сервер, спочатку перевіряється на відповідність правилам доступу, потім обробляється відповідним сервісом і завершується формуванням відповіді на рівні контролера.

Логіка сервісних класів, контролерів та репозиторіїв взаємопов’язана між собою та реалізує зв’язки один до одного.

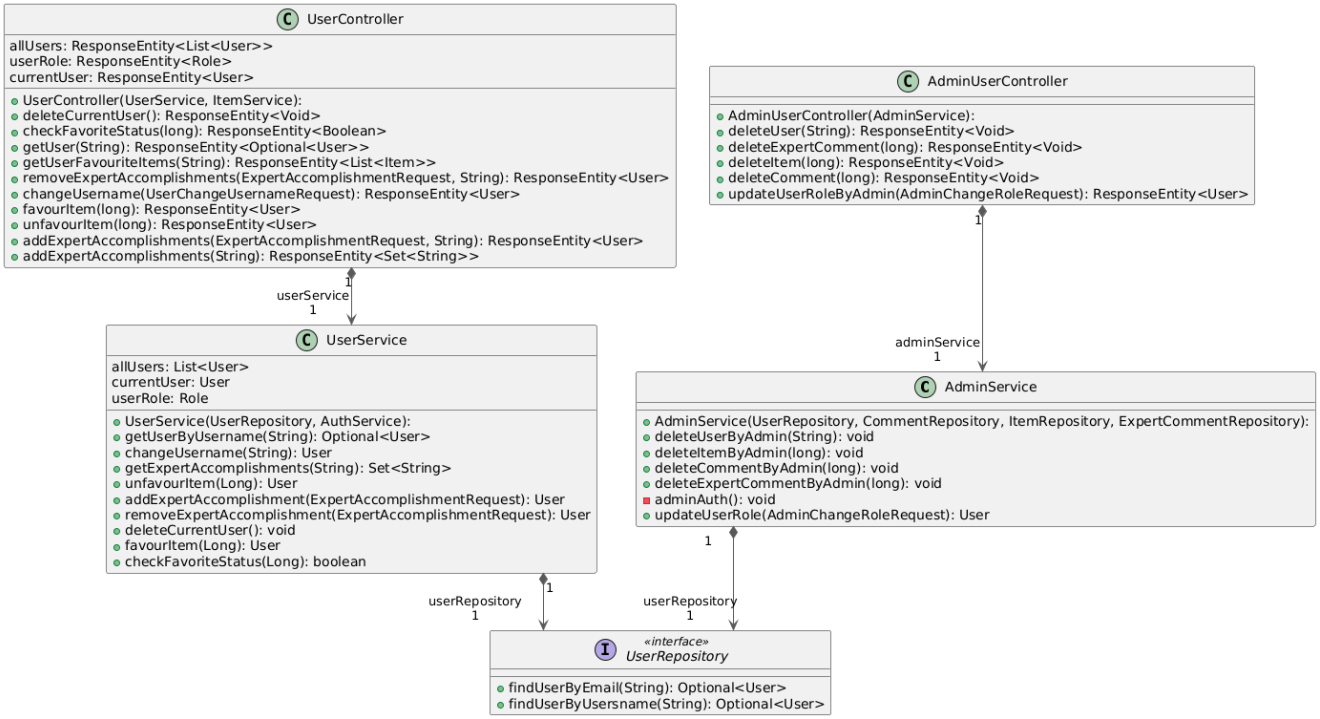


Рис. 3.2 Діаграма класів функціоналу користувача та адміністратора

Додатковим є використання протоколу WebSocket, що дозволяє створювати двостороннє підключення між клієнтами та сервером. Після підключення до брокера повідомлень клієнти можуть надсилати події, які обробляються відповідними методами на сервері, та отримувати відповіді без необхідності повторних HTTP-запитів. Це значно покращує користувацький досвід, особливо у багатокористувацьких середовищах із активною комунікацією. У проєкті цей протокол реалізує глобальний чат.

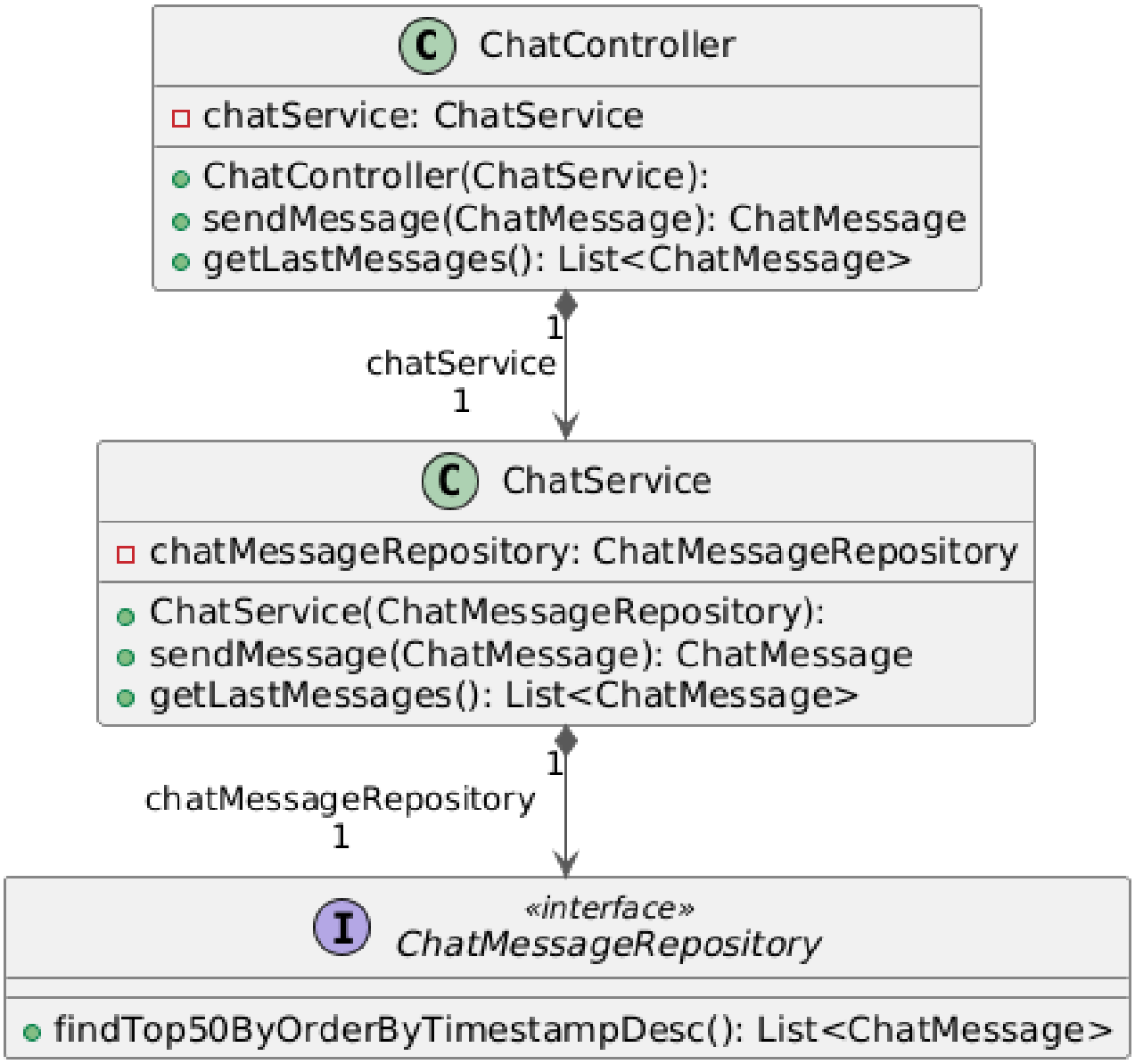


Рис. 3.3 Діаграма класів глобального чату

Безпека застосунку забезпечується за допомогою Spring Security у поєднанні з JWT (JSON Web Token). Після успішної автентифікації користувач отримує токен, який зберігається на клієнті та надсилається з кожним запитом у заголовку. Сервер перевіряє токен та дозволяє доступ до захищених ресурсів відповідно до ролі користувача. У конфігурації безпеки налаштовані фільтри для обробки JWT, а також політики дозволу для окремих маршрутів.

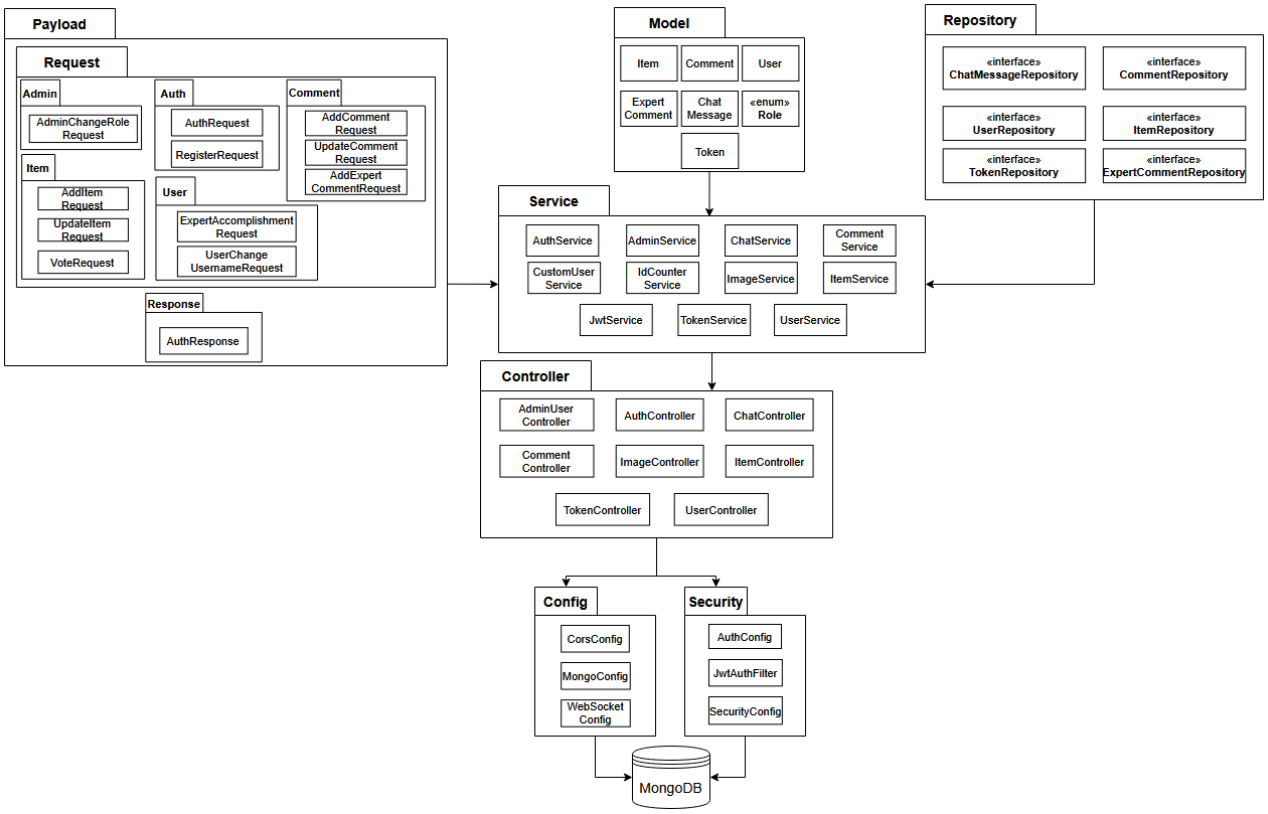


Рис. 3.4 Діаграма пакетів серверної частини застосунку

3.3. Архітектура клієнтської частини

Клієнтська частина застосунку реалізована з використанням фреймворку Next.js, побудованого на основі популярної бібліотеки для розробки web-застосунків React. Такий вибір зумовлений широкими можливостями Next.js щодо підтримки гібридного рендерингу, включаючи серверне (SSR) та статичне (SSG) відображення сторінок, що позитивно впливає на продуктивність, швидкість завантаження, SEO-оптимізацію та загальну стабільність роботи інтерфейсу.

Особливістю реалізації є чіткий поділ функціональних частин клієнтського коду на три основні пакети: логіку маршрутизації та відображення інтерфейсу зосереджено у пакеті pages, компоненти, що рендеряться на клієнті, реалізовано у components, а роботу з даними, авторизацією та запитами до бекенду інкапсульовано у пакеті services. Така модульна організація дозволяє спростити навігацію по проекту, полегшує масштабування системи, забезпечує високу зрозумілість та повторне використання коду.

У пакеті pages зберігаються усі маршрутизовані сторінки застосунку, як статичні сторінки (наприклад головна сторінка), так і динамічно згенеровані сторінки, які формуються на основі параметрів URL (наприклад ідентифікатор предмету каталогу), що дає змогу створювати унікальні сторінки для окремих ресурсів (предметів, профілів користувачів тощо). Вбудована маршрутизація Next.js дозволяє зручно керувати переходами між сторінками без необхідності конфігурування зовнішніх роутерів.

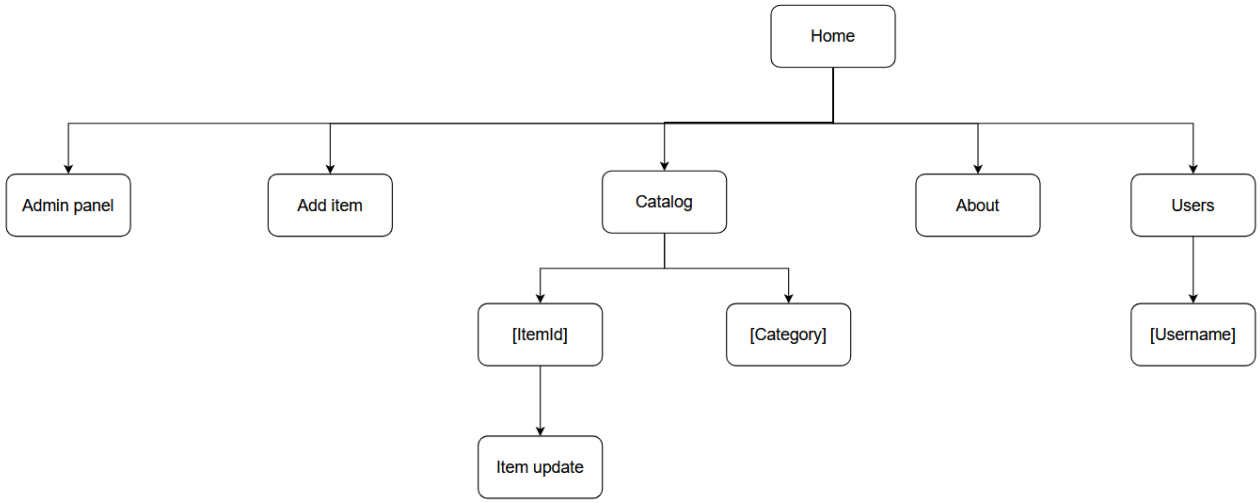


Рис. 3.5 Діаграма мапи сайту

Пакет services об'єднує логіку роботи з серверною частиною системи. У ньому реалізовано функції надсилання HTTP-запитів до REST API, побудованого на основі Spring Boot. Ці сервіси відповідальні за ключові бізнес-операції: автентифікацію та реєстрацію користувачів, отримання, оновлення, створення й видалення даних (предметів, коментарів, голосів тощо). Для авторизації клієнт використовує JWT-токени, які зберігаються локально у сховищі браузера (localStorage) і додаються до кожного захищеного запиту через HTTP-заголовки, що забезпечує безпечну та безперебійну аутентифікацію протягом сесії.

Пакет components містить універсальні, повторно використовувані елементи інтерфейсу, які застосовуються на різних сторінках застосунку. Це дозволяє

уникнути дублювання коду, підвищити консистентність візуального стилю та полегшити супровід клієнтської частини системи. Компоненти зазвичай охоплюють як елементи низького рівня (кнопки, поля вводу, модальні вікна), так і більш складні конструкції, такі як картки предметів каталогу, форми авторизації або списки коментарів.

Візуальна частина інтерфейсу реалізована із застосуванням Tailwind CSS - утилітарного CSS-фреймворку, який дозволяє будувати адаптивні й стильні компоненти без потреби у написанні великої кількості кастомного CSS-коду. Tailwind пропонує велику кількість готових класів, що вбудовуються безпосередньо у розмітку компонентів, полегшуючи процес стилізації і пришвидшуючи розробку. Це сприяє створенню єдиного стилю користувацького інтерфейсу та забезпечує високу гнучкість у його налаштуванні.

Крім того, вся клієнтська архітектура орієнтована на повторне використання компонентів. Значна частина функціональності винесена у вигляді універсальних React-компонентів, які використовуються на різних сторінках (наприклад, форма коментування, кнопка авторизації, картка предмета). Такий підхід дозволяє мінімізувати дублювання коду, покращити його підтримку та полегшити тестування.

Клієнтський інтерфейс дозволяє користувачам більш зрозуміло користуватися функціоналом web-застосунку.

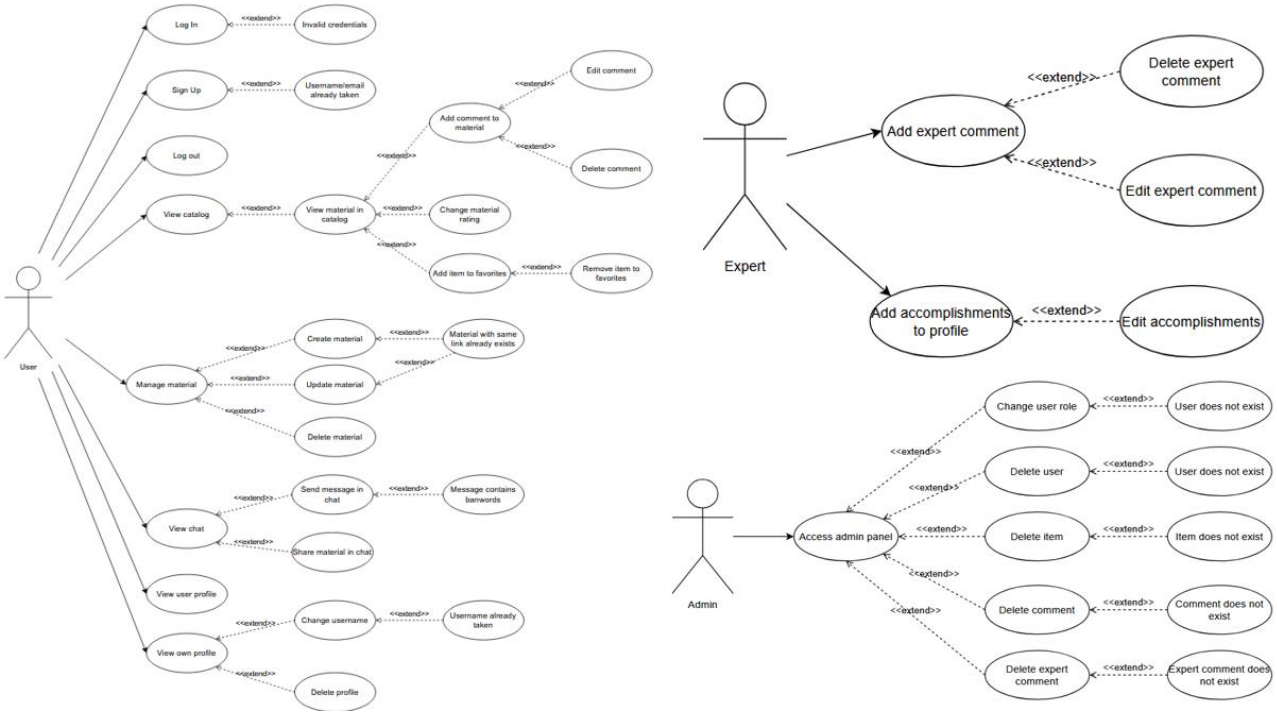


Рис. 3.6 Діаграма варіантів використання

3.4. Структура бази даних

База даних застосунку реалізована за допомогою документо-орієнтованої системи керування базами даних MongoDB, яка забезпечує гнучке зберігання, масштабованість та високу продуктивність при роботі з великими обсягами напівструктурованих даних. На відміну від реляційних СУБД, де дані організовано у вигляді таблиць зі строгою схемою, MongoDB зберігає інформацію у вигляді документів BSON (Binary JSON), які підтримують вкладені структури, масиви та інші складні типи даних. Це дозволяє досягти високої гнучкості у розробці та модифікації моделі даних без необхідності виконувати трудомісткі операції з міграції схем.

MongoDB ідеально підходить для сучасних web-застосунків, зокрема тих, що потребують частих змін у структурі даних. Це дозволяє уникнути складних міграцій схем та забезпечити адаптивність під час еволюції бізнес-логіки застосунку.

У застосунку зберігання даних реалізовано відповідно до моделей (entity-класів), що описані у пакеті моделей на стороні серверної частини. Кожна така модель відповідає окремій колекції у базі даних MongoDB. Для автоматичного зв’язування моделей із документами в колекціях використовуються спеціальні анотації Spring Data MongoDB, зокрема @Document(collection = "назва") вказує, що клас є моделлю документа, який зберігатиметься у відповідній колекції MongoDB, а @Id визначає унікальний ідентифікатор документа.

У межах реалізованого застосунку зберігання даних організовано згідно з моделями (entity-класами), які описані у пакеті model серверної частини на базі Spring Boot. Кожна така модель безпосередньо відповідає окремій колекції у базі даних MongoDB. Для автоматичного відображення моделей на документи та реалізації CRUD-операцій використовується Spring Data MongoDB, яка забезпечує абстракцію доступу до бази даних.

– User (users) — зберігає інформацію про зареєстрованих користувачів: ідентифікатор, логін, пароль (захешований), електронна пошта, роль, дата реєстрації, список улюблених предметів, к;

– Item (catalog) — описує окремі предмети у системі: назва, опис, автор, дата створення, кількість голосів, список тегів, зображення;

– Comment (comments) — містить коментарі до предметів, включаючи текст коментаря, автора, дату публікації та прив’язку до предмета.

– Message (messages) — реалізує структуру повідомлень у чаті: вміст повідомлення, автор, час надсилання.

– Token (tokens) — містить токен.

MongoDB не підтримує традиційних зовнішніх ключів, характерних для реляційних СУБД. Натомість взаємозв’язки між сутностями реалізуються через вбудовування об’єктів (embedded documents) або зберігання ідентифікаторів пов’язаних документів (reference). У застосунку обрано підхід з посиланнями (reference), коли, наприклад, у документі Comment зберігається itemId, що вказує на документ у колекції items. Це дозволяє забезпечити нормалізовану структуру даних, уникати дублювання та спростити оновлення пов’язаних сутностей.

Завдяки природі MongoDB та особливостям Spring Data, структура документів легко адаптується до змін — додавання нових полів або зміна формату даних не потребує додаткової міграції схеми. Це дозволяє швидко реагувати на зміну вимог без шкоди для цілісності даних або функціональності застосунку.

Безпека бази даних забезпечується шляхом обмеження доступу до неї лише з боку серверного застосунку, а також завдяки автентифікації MongoDB з унікальними обліковими даними.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Реалізація серверної частини

Середовище розробки IntelliJ IDEA має вбудований функціонал для створення Spring-застосунків, дозволяючи користувачам не лише ініціалізувати застосунок, а й обрати необхідні модулі з переліку доступних.

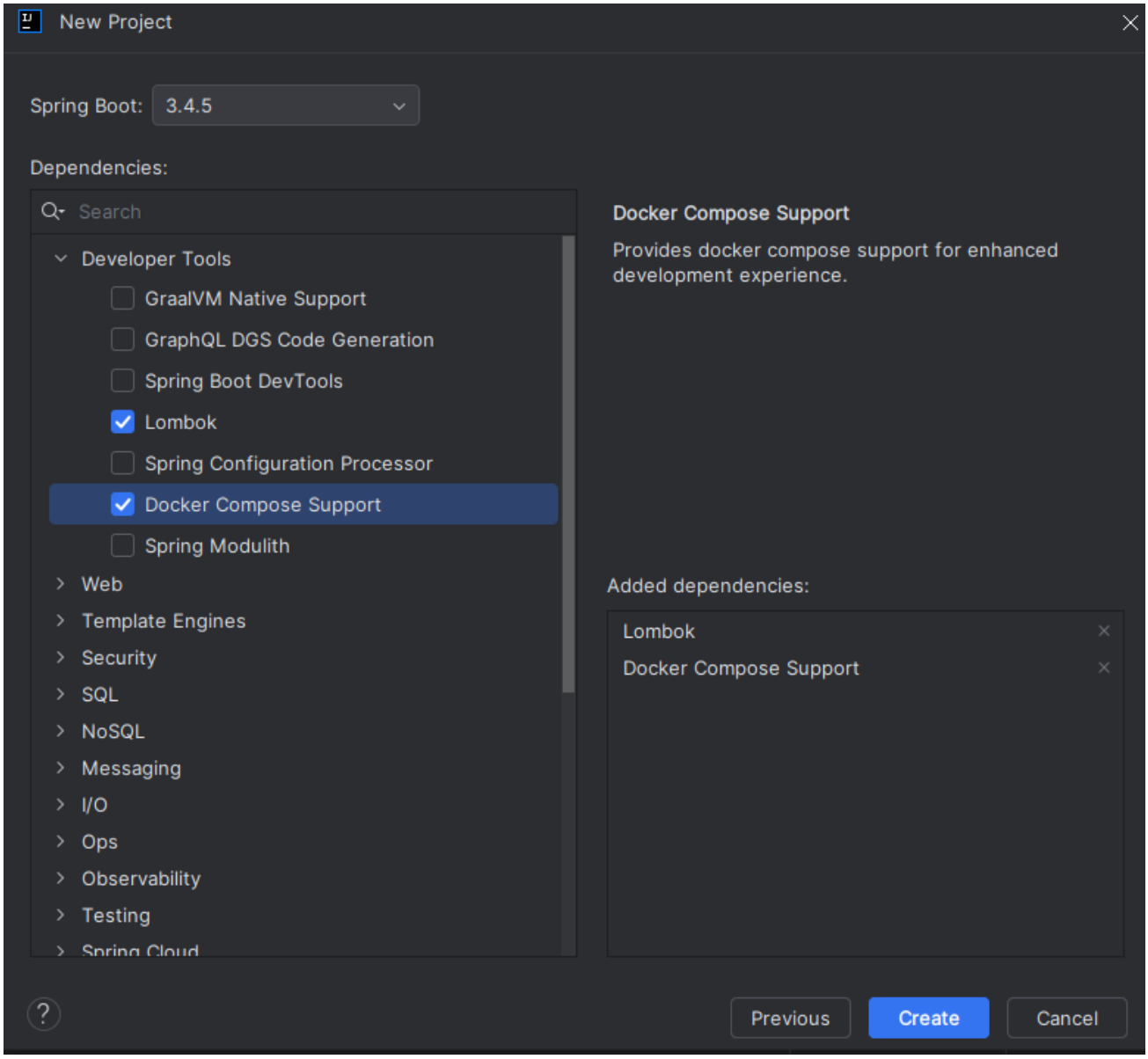


Рис. 4.1 Вибір програмних залежностей

З самого початку розробнику доступний основний файл програми, що запускає застосунок, та конфігураційний файл для Spring-проєкту під назвою application.properties, де найчастіше за все прописують зв’язок із базою даних, та у випадку використання JWT-токенів секретний ключ для них.

Компоненти Spring-застосунку позначаються анотаціями, аби розділити обов’язки між програмними класами. Так, контролери позначаються анотацією @Controller, сервісні класи позначаються як @Service тощо.

Необхідним компонентом для роботи Spring-застосунка є контролери. Ці класи реалізують RESTful архітектуру, відповідаючи за запити до застосунку через власні ендпоінти – посилання, які. У застосунку реалізовано наступні методи RESTful:

– GET: запит для отримання даних;
– POST: надсилання даних;
– PUT: вставка даних у вже існуючий об’єкт;
– PATCH: заміна даних у вже існуючому об’єкті;
– DELETE: видалення даних.

Основна взаємодія всередині застосунку побудована навколо моделей. Вони виступають як зразок для сутностей, які можна створювати та змінювати.

Основна бізнес-логіка реалізована у пакеті сервісів. У сервісах прописані методи, які згодом викликаються у контролерах при надсиланні запитів на сервер.

Взаємодія зі сторони бази даних реалізована через репозиторії. Це інтерфейси, що виконують SQL-запити без необхідності їх прописувати явно у сервісних методах, тим самим підвищуючи безпеку застосунку шляхом приховування запитів до бази даних.

Деякі методи потребують передачі даних. Для цього реалізовано пакет запитів, у яких зберігаються незмінні об’єкти класу Record. Вони відповідають за передачу необхідних даних при надсиланні запитів та отриманні відповідей від сервера.

```
@Builder  4 usages  MihaVar
public record RegisterRequest(
        @NotBlank(message = "Email cannot be empty")  2 usages
        String email,
        @NotBlank(message = "Username cannot be empty")  2 usages
        String username,
        @NotBlank(message = "Password cannot be empty")  1 usage
        String password
) {
}
```

Рис. 4.2 Приклад Record для реєстрації

У проєкті також підключена бібліотека Lombok, що за допомогою анотацій спрощує написання таких конструкцій, як гетери, сетери або конструктори, і також надає можливість зручніше створювати об’єкти через анотацію @Builder.

Для взаємодії з базою даних MongoDB у застосунку використано окремий клас конфігурації MongoConfig. Він позначений анотацією @Configuration, що вказує на те, що цей клас містить налаштування для Spring-контейнера. Анотація @EnableMongoRepositories активує підтримку репозиторіїв MongoDB у зазначеному пакеті, а через @Profile("!test") задається умова, що ця конфігурація не буде активована під час тестування, що дозволяє мати окреме налаштування для юніт- та інтеграційних тестів.

Усередині класу реалізовано підключення до MongoDB за допомогою клієнта MongoClient, який створюється на основі URI, переданого через змінну оточення spring.data.mongodb.uri, що зберігається у файлі application.properties. Метод getDatabaseName() вказує назву бази даних — social_elib, яка використовується для зберігання всіх колекцій.

Додатково оголошено Bean-компонент MongoTemplate, який надає розширену функціональність для взаємодії з MongoDB — наприклад, для виконання складніших запитів або кастомної логіки збереження даних. Така

модульна структура конфігурації дозволяє легко змінювати параметри підключення до бази, забезпечує гнучкість та полегшує супровід коду.

При старті проєкту створюється локальне підключення, зазвичай за портом 8080, з яким і взаємодіє клієнт.

4.2 Реалізація клієнтської частини

WebStorm має можливість створити автоматично створити проєкт Next.js, та зазвичай такі проєкти створюються шляхом встановлення фреймворку у потрібній локації через команду npx create-next-app у терміналі. При встановленні можна обрати встановлення додаткового функціоналу, як наприклад мову TypeScript чи бібліотеку Tailwind CSS.

```
PS F:\untitled> npx create-next-app
Need to install the following packages:
create-next-app@15.3.2
Ok to proceed? (y) y
√ What is your project named? ... my-app
√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like your code inside a `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to use Turbopack for `next dev`? ... No / Yes
√ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
Creating a new Next.js app in F:\untitled\my-app.
```

Рис. 4.3 Ініціалізація Next.js.

Next.js підтримує як серверний, так і клієнтський рендер компонентів. Це означає, що компоненти можуть створюватися як на сервері, так і прямо у клієнта.

Структура проєкту реалізована через три окремі теки – тека зі сторінками pages, тека із функціями API service та тека клієнтських компонентів components.

Сторінки обробляються на сервері, коли ж компоненти із основним функціоналом цих сторінок рендеряться з боку клієнта.

Фреймворк дозволяє динамічно розробляти застосунок завдяки динамічному оновленню web-сторінки після будь-яких змін у кодовій частині, дозволяючи зручно спостерігати за змінами без необхідності перезапускати проєкт. Аби запустити його, потрібно ввести команду npm run dev у терміналі.

4.3 Реалізація функціоналу

Функціонал web-застосунку реалізований через поєднання серверних функцій із інтерфейсом, що відображений у клієнті.

4.3.1 Авторизація користувача

Аби користувач отримав повний доступ до можливостей звичайного користувача, йому необхідно зареєструватися. При запиті за серверним ендпоінтом /auth/register він передає свою пошту, бажане користувацьке ім’я та пароль. Якщо пошта чи ім’я не зайняті, сервер формує об’єкт користувача, також генеруючи токен, що свідчить про авторизацію користувача у системі.

Якщо користувач вирішив вийти, то за запитом /auth/logout його токен робиться невалідним, тим самим унеможливлюючи користування застосунком через цей токен.

Для повторної авторизації користувачу потрібно надіслати свою пошту та пароль за ендпоінтом /auth/authenticate, в результаті чого йому надається новий токен.

У клієнтській частині користувач проходить авторизацію через форми, доступні за кнопками, розташованими у шапці сайту. Після авторизації сервер надсилає токен клієнту, що зберігає його у локальному сховищі браузеру для подальшого використання функціоналу, доступного лише авторизованим користувачам.

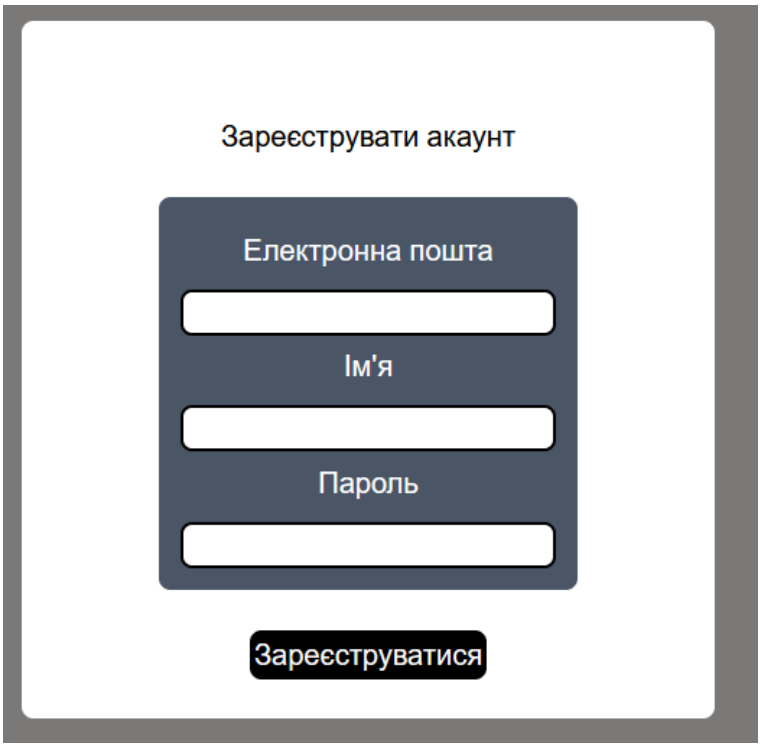


Рис. 4.4 Форма реєстрації на клієнті

4.3.2 Робота із матеріалами бібліотеки

Основний функціонал застосунку будується навколо предметів бібліотеки. Усі користувачі, незалежно від авторизації, можуть переглядати матеріали, присутні у бібліотеці, завдяки головній сторінці, де розміщені найновіші матеріали та матеріали з найвищим рейтингом, та завдяки каталогу, де розміщені усі матеріали каталогу.

Рис. 4.5 Головна сторінка

Кожен авторизований користувач може взаємодіяти із наповненням електронної бібліотеки та додавати свої власні матеріали. Для цього користувачу потрібно надіслати відповідний запит до серверу.

Аби створити предмет бібліотеки, користувач за ендпоінтом /catalog/add_item надсилає запит, в якому міститься:

– назва матеріалу;

– автор матеріалу (автор, наприклад, книжки, а не користувач, що її додає);

– опис матеріалу;

– дата публікації;

– посилання на джерело, де розміщено матеріал (окрім посилань з доменом .ru та .by), або власний файл у форматі PDF;

– Зображення матеріалу (опціонально).

Після запиту формується унікальний JSON-документ у базі даних, що містить дані про доданий предмет. Зображення можливо додати за допомогою окремого сервісу, що зберігає дані про фото у окремій таблиці та формує посилання на нього для документу матеріалу. За схожим принципом працює і сервіс для завантаження PDF-документів.

Перед створенням матеріалу перевіряється унікальність, а саме посилання на нього, аби не допустити створення двох ідентичних матеріалів. Щоб зручно ідентифікувати предмет працює сервіс, що створює унікальний лічильник для чисел і оновлюється з кожним створеним об’єктом.

З клієнтського боку користувач може створити предмет, натиснувши на кнопку «Створити об’єкт каталогу», розташовану у панелі ліворуч, де від нього потрібно заповнити форму та натиснути на кнопку додавання матеріалу. На сервер передається токен користувача та дані, що були введені ним у формі.

Після додавання матеріалу, він розміщується за унікальним для кожного матеріалу шляхом /catalog/[id]. Його можуть переглянути як авторизовані користувачі, так і будь-хто, хто вирішить зайти на сайт.

Рис. 4.6 Вигляд матеріалу на клієнті

Якщо користувач захоче змінити деякі параметри матеріалу, то за ендпоінтом /catalog/{itemId}/update-item, що містить у собі ідентифікатор матеріалу, він може надіслати нові значення для параметрів, що потребують змін. Запит надсилається за методом PATCH, призначений для операцій, пов’язаних зі змінами у об’єктах.

Якщо користувач захоче видалити предмет з бібліотеки, він може це зробити через запит з методом DELETE за ендпоінтом /catalog/{itemId}/delete-item.

Перед надсиланням запитів, пов’язаних з редагуванням та видаленням предметів, перевіряється авторизація користувача, аби користувачі не могли змінювати чужі предмети.

Якщо користувача цікавить певна категорія, він може переглянути усі матеріали, пов’язані з даною категорією, використавши бокову панель або натиснувши на категорію при перегляді матеріалу. Категорії можуть бути розширені в майбутньому, що забезпечує масштабованість web-застосунку електронної бібліотеки.

Рис. 4.7 Перегляд матеріалів за категорією «Література»

4.3.3 Взаємодія з користувачами через бібліотеку

Основна взаємодія між користувачами web-застосунку проходить через окремі предмети бібліотеки. Авторизовані користувачі можуть залишати власні коментарі до предметів, ставити предметам оцінки та зберігати їх до обраного.

Аби залишити коментар, користувачеві потрібно написати його у формі під предметом та натиснути на кнопку публікування коментаря. До сервера передається токен та текст коментаря, на основі чого формується відповідний об’єкт через ендпоінт /catalog/{itemId}/comments/add-comment.

Рис. 4.8 Форма для відправки коментаря та вигляд коментарів.

Коментарі, як і предмети, мають свій унікальний ідентифікатор, згенерований сервісом послідовності, а також вони містять у собі ідентифікатор матеріалу, до якого вони були додані, тож вони відображаються лише під своїми

матеріалами. Користувач за потреби може редагувати та видаляти власні коментарі.

Оцінювання предметів бібліотеки можливо через натискання відповідних позначок біля загального рейтингу предмета. Поставити свою оцінку можна лише один раз, та її також можливо відмінити. Від рейтингу матеріалу також змінюється рейтинг користувача, що додав цей матеріал.

Кожен користувач може додати предмет до обраного, що потім відобразиться у нього в профілі, як рекомендація даного користувача до прочитання.

4.3.4 Глобальний чат

Аби забезпечити пряму взаємодію між користувачами без необхідності перезавантажувати сторінку, у web-застосунку реалізовано глобальний чат.

Відправляти повідомлення до нього можуть лише авторизовані користувачі. Чат зберігає останні 50 повідомлень, які були надіслані до нього. Кожне повідомлення містить ім’я користувача, що надіслав його, вміст повідомлення та дату відправки.

Особливістю також є можливість поширювати матеріали, доступні на сайті. Коли користувач надсилає посилання матеріалу, воно автоматично обробляється та подається у вигляді предмету, який також можна відкрити. Власні повідомлення виводяться праворуч, а повідомлення інших користувачів – ліворуч.

Чат реалізовано завдяки технології WebSocket, що встановлює двосторонній зв’язок між сервером та клієнтом, дозволяючи відправляти запити та стежити за змінами у режимі реального часу, без потреби оновлювати сторінку. Сервіс ChatService дозволяє надсилати повідомлення та отримувати останні 50 повідомлень.

Для реалізації функціоналу реального часу, що лежить в основі роботи чату, було створено спеціальний клас конфігурації WebSocketConfig. Цей клас відповідає за налаштування WebSocket-з’єднання між клієнтом та сервером, а також за маршрутизацію повідомлень у межах застосунку.

Анотації @Configuration та @EnableWebSocketMessageBroker активують підтримку WebSocket та дозволяють використовувати брокер повідомлень на основі протоколу STOMP. У методі registerStompEndpoints() визначено точку підключення клієнтів (/ws), яка підтримує з'єднання через SockJS для забезпечення сумісності з різними браузерами. Дозволено підключення лише з клієнта, що працює на http://localhost:3000, що відповідає середовищу розробки.

У методі configureMessageBroker() налаштовано внутрішній брокер повідомлень, який обслуговує підписки на теми (/topic) та приймає повідомлення, адресовані серверу, за префіксом /app. Завдяки цим налаштуванням клієнти можуть надсилати повідомлення за визначеними маршрутами та отримувати оновлення в режимі реального часу. Така архітектура забезпечує ефективну та масштабовану взаємодію між користувачами в чаті.

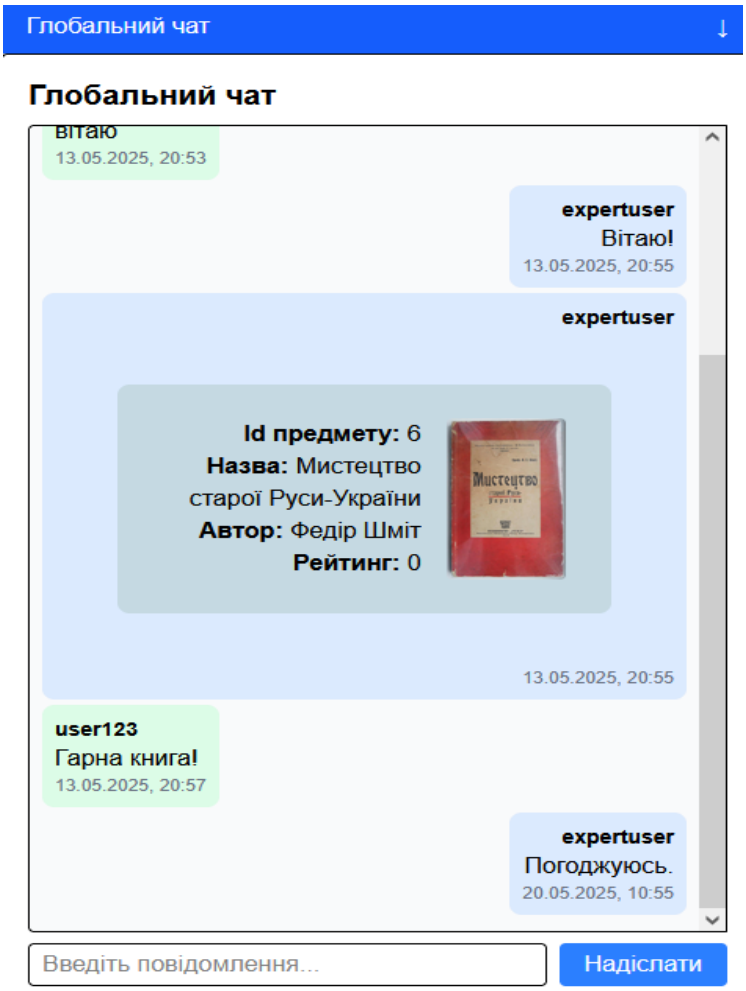


Рис. 4.9 Вигляд глобального чату

4.3.5 Користувацькі профілі

Кожен користувач має свій власний профіль, який можливо відкрити, натиснувши на ім’я користувача в елементах, де воно присутнє (чат, матеріали, шапка сайту). У профілі розміщена така інформація користувача, як:

– ім’я користувача;

– роль користувача;

– рейтинг користувача.

Рейтинг користувача залежить від оцінок його власних матеріалів, як позитивних, так і негативних.

У профілі також розміщуються матеріали, створені користувачем, та список обраних користувачем матеріалів.

Якщо користувач переглядає власний профіль, то йому доступна зміна власного імені та видалення профілю.

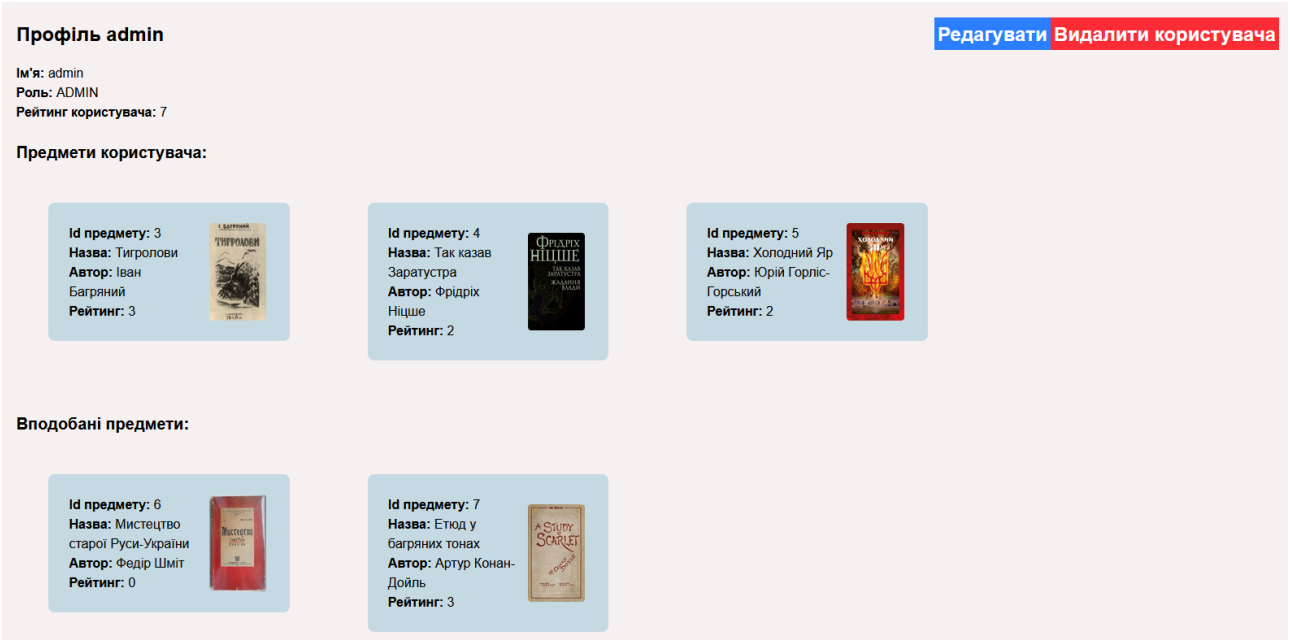


Рис. 4.10 Перегляд власного профілю

4.3.6 Особливості ролей користувачів

У web-застосунку реалізовані користувацькі ролі, що мають власний функціонал. Всього їх чотири:

– USER: звичайний авторизований користувач з доступом до основного функціоналу;

– RESPECTED_USER: користувач, що має понад 50 позитивних оцінок на власних матеріалах;

– EXPERT: запрошений експерт, що може додавати унікальні експертні коментарі до предметів;

– ADMIN: адміністратор, здатний змінювати ролі користувачів та видаляти користувачів, предмети і коментарі.

Варто виділити відмінності унікальних ролей RESPECTED_USER, EXPERT та ADMIN.

У ролей RESPECTED_USER та EXPERT зроблені унікальні кольори для коментарів, аби у користувачів була можливість відрізняти, хто саме коментує предмет. Звичайний користувач має білий колір, поважний користувач – синій, а експерт – жовтий.

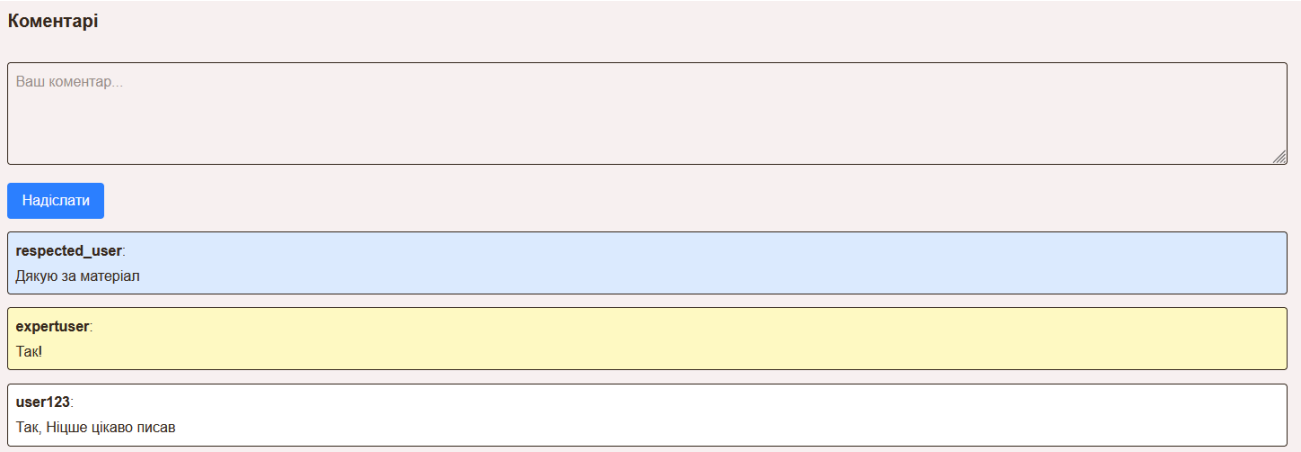


Рис. 4.11 Вигляд коментарів в залежності від ролей

У запрошених експертів є можливість додавати власні експертні коментарі до предметів. Вони відображаються у самому предметі та створюються як об’єкт, що зберігається всередині об’єкту предмету. Також їх можна редагувати та видаляти.

Рис. 4.12 Вигляд предмету з доданим експертним коментарем

Також запрошені експерти можуть додавати свої наукові досягнення, наприклад наукові ступені, у своєму профілі, аби користувачі могли їх бачити та визначати, чи варто довіряти його експертному коментарю.

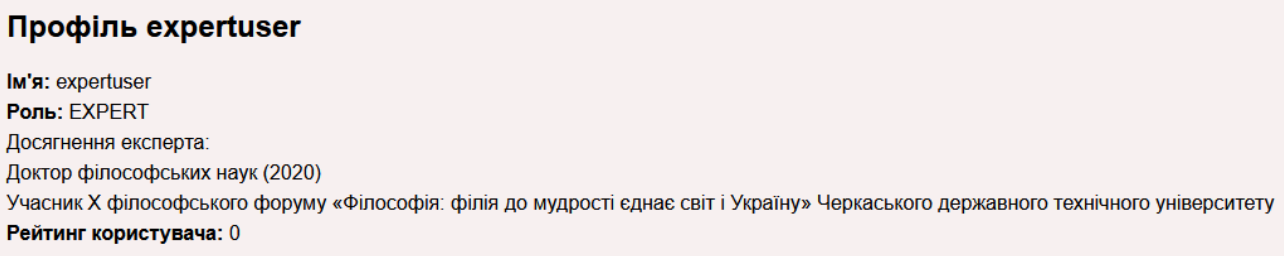


Рис. 4.13 Виведення досягнень експерта у профілі

Користувачам з роллю адміністратора доступна адміністраторська панель, яка дозволяє змінити роль користувача з USER на EXPERT, а також видалити вказаного користувача, предмет, коментар чи експертний коментар. Для видалення користувача потрібно ввести його ім’я, а для видалення решти потрібно вказати id. Перед виконанням кожної дії адміністратору треба підтвердити дію, перш ніж виконати її.

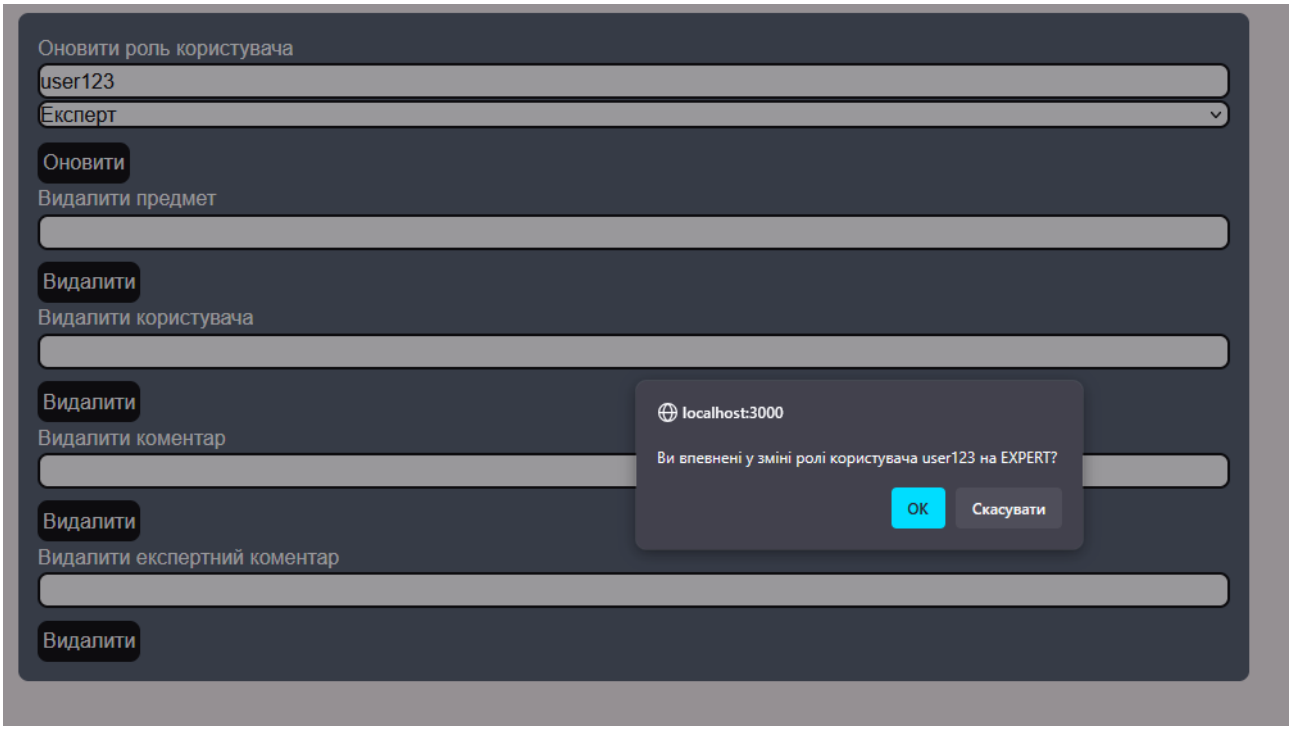


Рис. 4.14 Адмін-панель з підтвердженням зміни ролі користувача

4.4 Реалізація безпеки застосунку

Безпека web-застосунку реалізована на серверній частині завдяки безпековому фреймворку Spring Security та стандарту токенів JWT.

Основна безпека реалізована у пакеті security, де є клас налаштування авторизації, клас налаштування фільтру для JWT та клас налаштування безпеки ендпоінтів. У пакеті config зберігається клас для налаштування CORS.

Клас AuthConfig відповідає за налаштування автентифікації користувача та імплементує сервіси даних користувача і шифрування паролю. У стандартному Spring Security дані користувача представлені у вигляді імені та паролю, тож аби безпека працювала за поштою, а не за ім’ям, було створений окремий клас CustomUserService, що заміняє визначення імені користувача на електронну пошту.

Клас JwtAuthFilter є фільтром для генерації токенів. Він приймає надісланий токен та перевіряє його на валідність. У випадку, якщо токен виявився пошкодженим або несправжнім, фільтр відхиляє запити з цим токеном.

Авторизація проходить через класи JwtService, TokenService та AuthService. У першому класі реалізований функціонал для генерації токену, у який вкладається пошта та пароль користувача, та допоміжний функціонал для обробки достовірності токену. Цей токен використовує TokenService для побудови відповідного об’єкту, що зберігається у базі даних.

AuthService відповідає за авторизацію користувача. При реєстрації та авторизації користувач надсилає дані на сервер, де перевіряється їхня достовірність, та повертається токен, створений через TokenService.

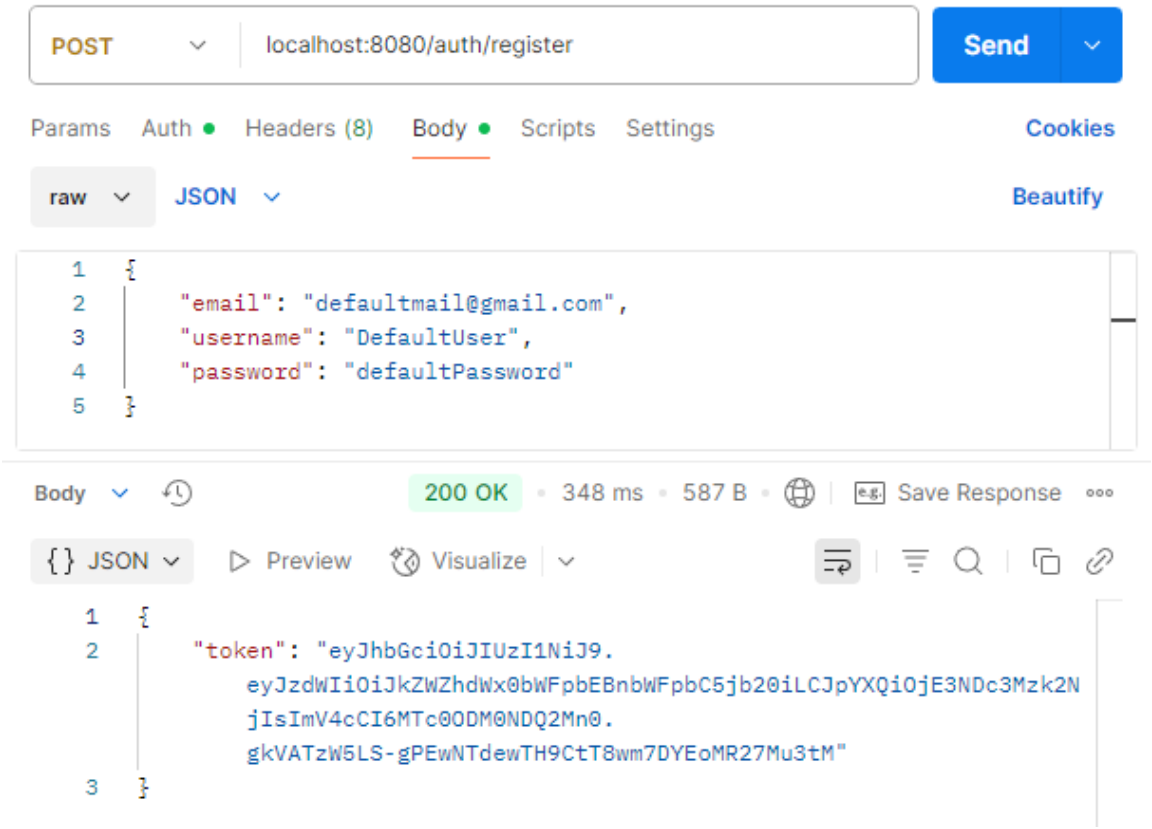


Рис. 4.15 Приклад запиту для реєстрації у Postman

Також реалізований вихід з акаунта, при якому наданий токен стає недійсним.

SecurityConfig відповідає за внутрішню безпеку серверу. У цьому класі прописані основні дозволи, що залежать від ролей користувачів, у вигляді списків, які прописуються у ланцюжку фільтрації.

```
private static final String[] WHITELIST_URLS = {  1 usage
        "/auth/**",
        "/error",
        "/catalog/items",
        "/catalog/{itemId}",
        "/catalog/category/{categoryName}",
        "/catalog/{itemId}/comments",
        "/images/{id}",
        "/catalog/items/{username}",
        "/users/{username}",
        "/users/{username}/favourites",
        "/users/{username}/expert-accomplishments",
        "/files/pdf/**"

};

private static final String[] EXPERTLIST_URLS = {  1 usage
        "/catalog/{itemId}/add_expert_comment",
        "/catalog/{itemId}/expert_comments/**",
        "/users/{username}/add-expert-accomplishments",
        "/users/{username}/remove-expert-accomplishment"
};

private static final String[] AUTHORIZEDLIST_URLS = {  5 usages
        "/catalog/**",
        "/users/me"
};

private static final String[] ADMINLIST_URLS = {  3 usages
        "/token/purge",
        "/admin/**"
};
```

Рис. 4.16 Списки доступних ендпоінтів для ролей

Клас CorsConfig виконує функцію регулятора для CORS - механізму захисту вебсторінок, який визначає дозволені джерела запитів. У даному застосунку як джерело визначено клієнтський сервер.

4.5 Тестування

Тестування основного функціоналу web-застосунку реалізовано на двох основних рівнях: модульному та інтеграційному.

Модульне тестування охоплює окремі компоненти (класи або методи) програми без залежності від зовнішніх ресурсів, таких як база даних або веб-сервер. Для реалізації модульного тестування використовується бібліотека JUnit Jupiter (частина JUnit 5) у поєднанні з Mockito — популярною бібліотекою для створення мок-об’єктів.

Завдяки Mockito вдається ізолювати тестовані компоненти, створюючи підроблені (mocked) версії залежностей. Наприклад, при тестуванні сервісного шару ініціалізуються мок-репозиторії, що дає змогу перевірити бізнес-логіку незалежно від роботи бази даних.

Інтеграційні тести охоплюють взаємодію між кількома компонентами системи, зокрема взаємодію з базою даних, реальними сервісами, контролерами тощо. Таке тестування дозволяє перевірити працездатність частини або всієї системи в умовах, наближених до реального середовища, із запущеним додатком.

Для реалізації інтеграційних тестів використовується Spring Boot Test, який забезпечує повноцінний контекст застосунку. Компоненти (контролери, сервіси, репозиторії) взаємодіють у реальному середовищі з підключенням до тестової бази даних (наприклад, H2), що дає змогу виявляти проблеми у взаємодії між шарами застосунку.

Для тестування REST-контролерів застосовується MockMvc у поєднанні з анотаціями @SpringBootTest, @AutoConfigureMockMvc та @Transactional. Це дозволяє емулювати HTTP-запити до контролерів і перевіряти статус-коди, тіла відповідей у форматі JSON, обробку параметрів запиту тощо.

Кожен інтеграційний тест орієнтується на окрему частину логіки - наприклад, ItemService, CommentService або AuthService. Для підготовки середовища перед тестами ініціалізуються необхідні дані - за допомогою SQL-скриптів, анотації @Sql, методу @BeforeEach або через TestEntityManager.

Поєднання модульного та інтеграційного тестування забезпечує як перевірку окремих одиниць логіки, так і цілісну перевірку функціонування системи в цілому. Такий підхід гарантує стабільність, виявляє помилки на ранніх етапах і спрощує подальший розвиток застосунку.

У процесі тестування також приділяється увага перевірці негативних сценаріїв — наприклад, некоректних запитів, неправильних параметрів або спроб доступу до захищених ресурсів без авторизації. Це дозволяє переконатися, що система поводиться передбачувано у разі помилок користувача або зовнішніх збоїв, і забезпечує необхідний рівень стійкості та безпеки.

У ході тестування розробленого web-застосунку було перевірено функціональні вимоги. Для прикладу успішного тестування функціоналу користувача наведена екранна форма інтеграційного тестування сервісу користувачів з видаленням користувача та зміною користувацького ім’я.

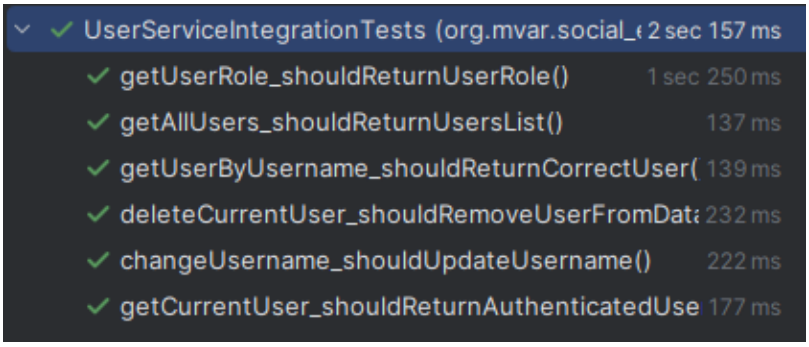


Рис. 4.17 Вигляд пройдених інтеграційних тестів

Серед нефункціональних вимог було перевірено усі наведені вимоги. Наведена форма зображує звичайне завантаження сторінки, що вкладається у терміни, визначені у вимогах.

```
GET /users/newuser 200 in 105ms
GET /add_item 200 in 85ms
GET /catalog 200 in 85ms
GET / 200 in 203ms
```

Рис. 4.18 Швидкість завантаження сторінок

4.6 Розгортання

Для того, аби перевірити дієздатність системи, було використано сучасні технології розгортання серверної частини застосунку з використанням контейнеризації. Основна серверна частина, реалізована за допомогою фреймворку Spring Boot, була упакована у Docker-контейнер, що дозволило ізолювати середовище виконання застосунку, забезпечити його портативність та легкість у масштабуванні.

Для організації повноцінного локального середовища розробки та тестування всі ключові компоненти системи — сервер застосунку та база даних MongoDB —

були інтегровані в єдину систему управління за допомогою Docker Compose. Використання Docker Compose дало змогу описати конфігурації обох сервісів у вигляді простого YAML-файлу, що забезпечило автоматичний запуск і злагоджену взаємодію між ними.

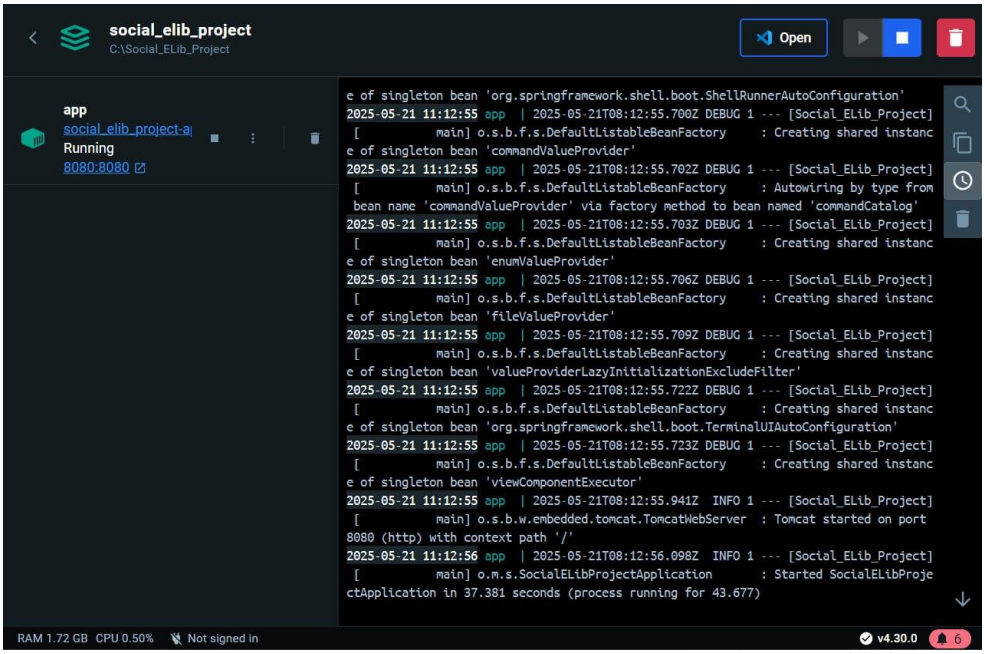


Рис. 4.19 Вигляд запущеного контейнера у Docker

Завдяки такому підходу розгортання стало максимально швидким та зручним: усю систему можна було запустити однією командою (docker-compose up), що значно спрощувало процес тестування та демонстрації функціоналу, а також зменшувало ризик помилок, пов’язаних із налаштуванням оточення.

Крім того, контейнеризація сприяла підвищенню стабільності роботи системи, оскільки кожен сервіс працював у своєму ізольованому середовищі з фіксованими версіями програмного забезпечення і залежностей. Це дозволяло уникнути проблем сумісності та спрощувало підтримку застосунку в майбутньому.

В подальшому такий підхід до розгортання може бути масштабований і на інші середовища, включаючи тестові, стендові та продуктивні, з мінімальними змінами в конфігураціях. Це створює основу для автоматизації процесів CI/CD (безперервної інтеграції та доставки), що підвищує ефективність розробки і впровадження оновлень.

ВИСНОВКИ

У кваліфікаційній роботі проведено аналіз предметної галузі електронних бібліотек та реалізовано web-застосунок з покращенням соціальної взаємодії між користувачами електронних бібліотек.

У результаті аналізу предметної області було розглянуто web-застосунок електронної бібліотеки та контексти його використання. Визначено цільову аудиторію електронних бібліотек. Проведено аналіз існуючих рішень електронних бібліотек, таких як Україніка, Лібрук, Аргонавти Всесвіту та Укрліб, також проведено порівняння усіх чотирьох аналогів. Визначено ключові задачі, необхідні для виконання роботи.

Розглянуто засоби реалізації. Як середовища розробки було обрано Intellij IDEA та WebStorm. Мовами програмування обрано Java та TypeScript. Для реалізації серверної частини було обрано фреймворк Spring Boot, а також обрано такі технології, як Spring Security, JWT-токени та WebSocket. Для реалізації клієнтської частини було обрано фреймворк Next.js та бібліотеку для стилізації Tailwind CSS. Як базу даних обрано документно-орієнтовану базу даних MongoDB.

Спроєктовано архітектуру застосунку. Як архітектуру застосунку було обрано клієнт-серверну модель, що дозволило розділити обов’язки між клієнтом та сервером. Серверну частину було вирішено проєктувати за принципом сервісної архітектури. Клієнтську частину вирішено представити у вигляді трьох пакетів: сторінок, компонентів та сервісу. Описано структуру бази даних.

Реалізовано функціонал та проведено тестування застосунку. У web-застосунку реалізована система авторизації, робота із матеріалами електронної бібліотеки, взаємодія між користувачами через предмети бібліотеки, власні профілі користувачів, глобальний чат з можливістю ділитися матеріалами, функціонал для користувачів із ролями експерта та адміністратора. Також було реалізовано нефункціональні вимоги, такі як шифрування паролів, захист від поширених атак,

робота без критичних помилок з можливістю відновлення, можливість розширення та завантаження сторінок не більше, ніж 2-3 секунди.

В результаті виконання кваліфікаційної роботи отримано web-застосунок електронної бібліотеки, що покращує соціальною взаємодією між її користувачами. Застосунок забезпечує безпеку даних користувачів, високу продуктивність та простоту у використанні, а також додає новий у предметній галузі функціонал, що робить його конкурентоспроможним на ринку.

Робота пройшла апробацію на науковій конференції. За результатами конференції було опубліковано наступні тези доповідей:

1. Варич М.Ю., Андрусевич Н.В. Визначення вимог до вебдодатку електронної бібліотеки із соціальною взаємодією. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с. 52-53.
2. Варич М.Ю., Андрусевич Н.В. Архітектура вебдодатку електронної бібліотеки із соціальною взаємодією. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с. 54-55.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bearman D. "Digital libraries." *Annual review of information science and technology 41.1*. 2007. P. 223-272.
2. Girgensohn A., Lee A. "Making web sites be places for social interaction." *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. 2002. URL: https://dl.acm.org/doi/abs/10.1145/587078.587098
3. Україніка - електронна бібліотека | НБУВ. URL: https://irbis-nbuv.gov.ua/cgi-bin/ua/elib.exe?C21COM=F&I21DBN=NAV&P21DBN=UKRLIB (дата звернення 15.03.2025).
4. Libruk | Українська електронна бібліотека. URL: https://libruk.com.ua/ (дата звернення 17.03.2025).
5. Аргонавти Всесвіту. Сайт україномовної фантастики. URL: https://argo-unf.at.ua/ (дата звернення 18.03.2025).
6. Бібліотека української літератури. URL: https://www.ukrlib.com.ua/ (дата звернення 20.03.2025).
7. Zayour I., Hajjdiab H. "How much integrated development environments (ides) improve productivity?." *J. Softw. 8.10*. 2013. P. 2425-2431.
8. IntelliJ IDEA overview | IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (дата звернення: 25.03.2025).
9. Getting started with WebStorm | WebStorm Documentation. URL: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html (дата звернення: 25.03.2025).
10. What is Java and why should I use it?. URL: https://www.java.com/en/download/help/whatis_java.html (дата звернення: 27.03.2025).

11. TypeScript: Documentation – TypeScript for the New Programmer. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (дата звернення: 27.03.2025).

12. Spring Boot. URL: https://spring.io/projects/spring-boot#overview (дата звернення: 31.03.2025).

13. Spring Security. URL: https://spring.io/projects/spring-security (дата звернення: 31.03.2025).

14. JSON Web Token Introduction - jwt.io. URL: https://jwt.io/introduction (дата звернення: 01.04.2025).

15. WebSocket API | WebSocket.org. URL: https://websocket.org/reference/websocket-api/ (дата звернення: 01.04.2025).

16. Introduction | Next.js. URL: https://nextjs.org/docs (дата звернення: 02.04.2025).

17. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/ (дата звернення: 02.04.2025).

18. What is MongoDB? – Database Manual v8.0. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 03.04.2025).

19. Docker Compose | Docker Docs. URL: https://docs.docker.com/compose/ (дата звернення: 04.04.2025).

20. Postman documentation overview | Postman Docs. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 04.04.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку електронної бібліотеки із соціальною взаємодією

Виконав студент 4 курсу
групи ПД-41
Варич Михаїл Юрійович
Керівник роботи
старший викладач кафедри ІПЗ Андрусевич Наталя Володимирівна

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - покращення соціальної взаємодії між користувачами електронних бібліотек шляхом створення клієнт-серверного вебзастосунка.
- **Об'єкт дослідження** - процес соціальної взаємодії між користувачами електронних бібліотек.
- **Предмет дослідження** - вебзастосунки електронних бібліотек.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. *Провести дослідження існуючих методів та рішень впровадження та забезпечення соціальної взаємодії користувачів вебзастосунків для подальшого використання у web-застосунку електронної бібліотеки.*

2. *Здійснити огляд та аналіз існуючих електронних бібліотек для визначення основного функціоналу та недоліків наявних рішень у предметній галузі.*

3. *Провести огляд засобів для реалізації програмного забезпечення клієнт-серверного web-застосунку електронної бібліотеки із соціальною взаємодією.*

4. *Сформувати вимоги до розробки клієнт-серверного web-застосунку електронної бібліотеки.*

5. *Спроектувати архітектуру клієнт-серверного web-застосунку електронної бібліотеки.*

6. *Розробити клієнт-серверний застосунок web-застосунку електронної бібліотеки із соціальною взаємодією.*

7. *Провести тестування клієнт-серверного web-застосунку електронної бібліотеки.*

3

АНАЛІЗ АНАЛОГІВ

Показник	Україніка	Лібрук	Аргонавти Всесвіту	УкрЛіб	ЛібХаб ЮА
Наявність живого чату	–	–	–	–	+
Широке направлення	+	-	-	-	+
Система авторизації	-	+	+	-	+
Посилання на джерела	+	+	+	-	+
Пошук за категорією	+	+	+	+	+
Тематична класифікація матеріалів	+	+	–	+	+
Коментування	–	+	+	–	+
Оцінювання матералів	–	–	+	–	+
Вільний доступ до публікації матеріалів	–	+	+	–	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

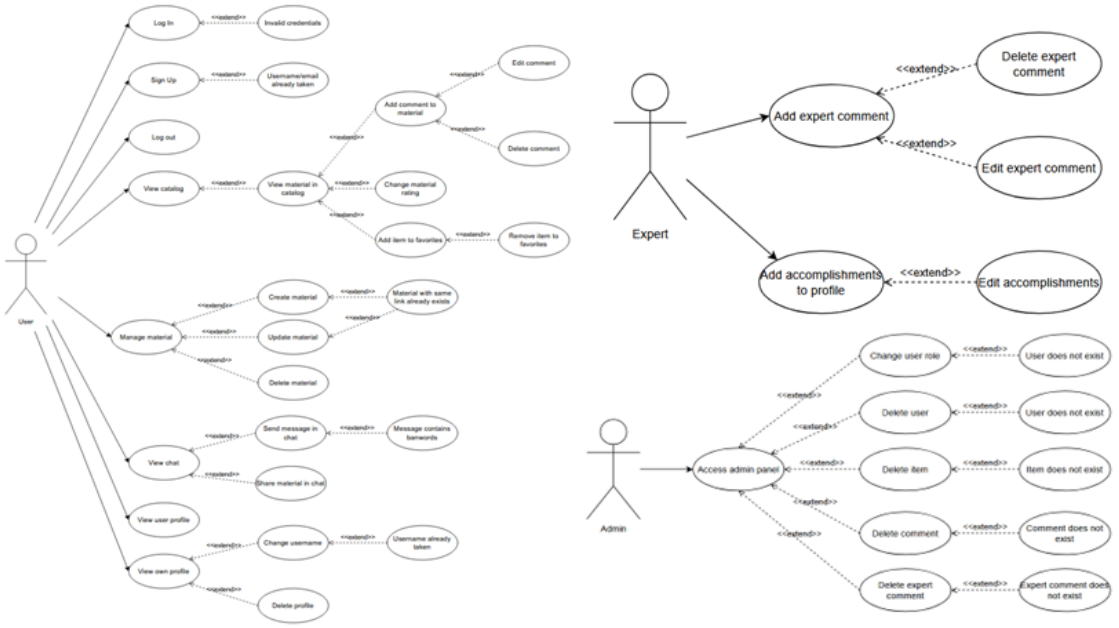
- Функціональні:
  1. Система авторизації;
  2. Взаємодія з профілем (редагування та видалення);
  3. Робота із бібліотекою (створення, редагування та видалення предметів бібліотеки);
  4. Взаємодія між користувачами (додавання та видалення коментарів і оцінок до матеріалів, надсилання повідомлень до чату);
  5. Панель адміністраторів (видалення користувачів, предметів та коментарів);
  6. Можливості для експертів (додавання та видалення експертних коментарів до предмету, редагування власних наукових досягнень).
- Нефункціональні:
  1. Шифрування паролів;
  2. Забезпечення захисту від поширених видів атак (Cross-Site Request Forgery, Cross-Site Scripting);
  3. Робота без критичних помилок системи з можливістю відновлення;
  4. Можливість розширення системи;
  5. Завантаження сторінок не більше, ніж 2-3 секунди.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

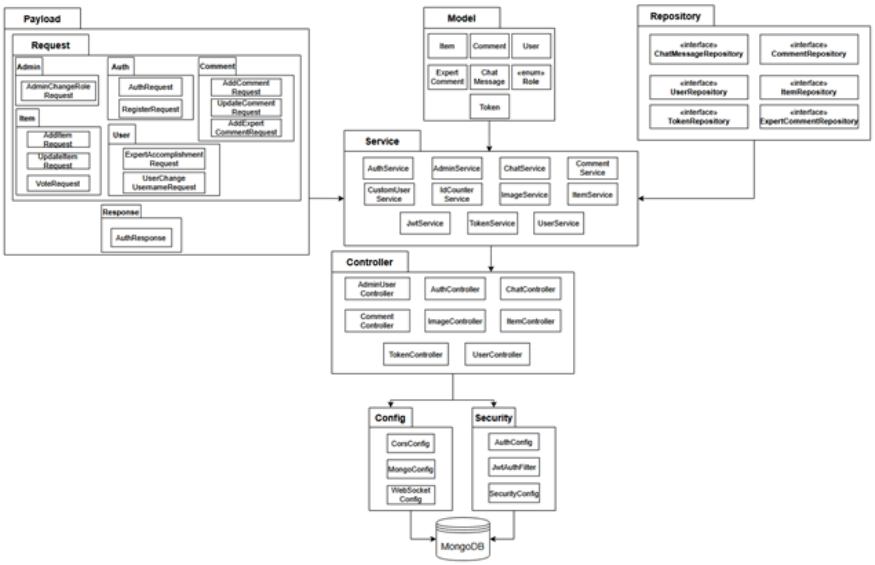
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



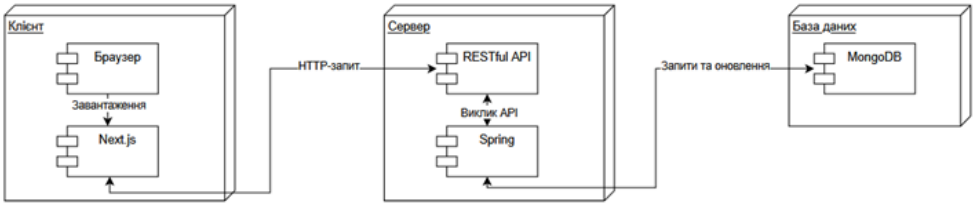
7

ДІАГРАМА ПАКЕТІВ СЕРВЕРНОЇ ЧАСТИНИ



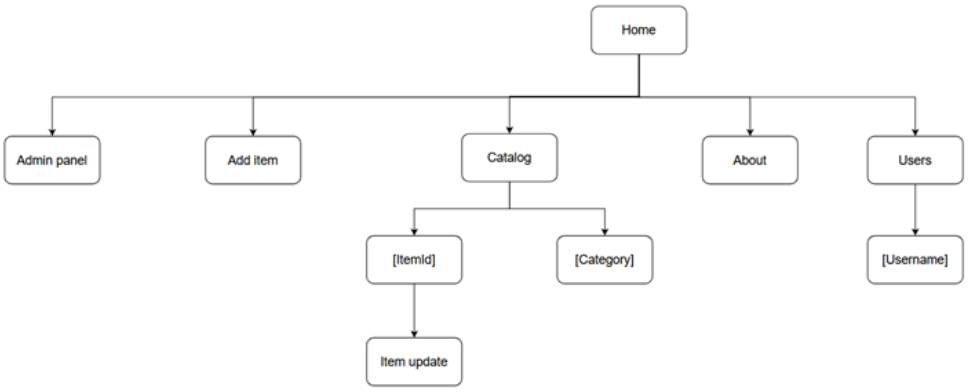
8

ДІАГРАМА РОЗГОРТАННЯ ЗАСТОСУНКУ



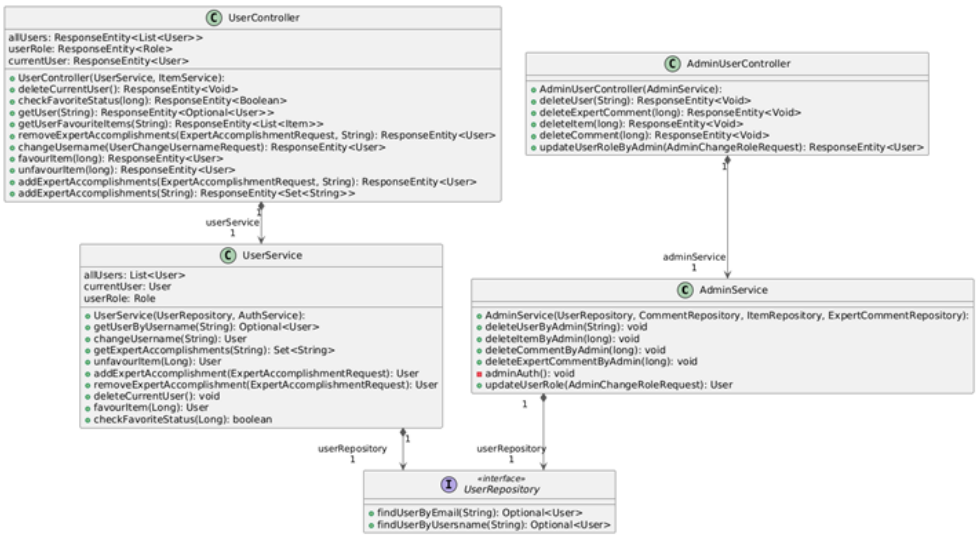
9

МАПА САЙТУ



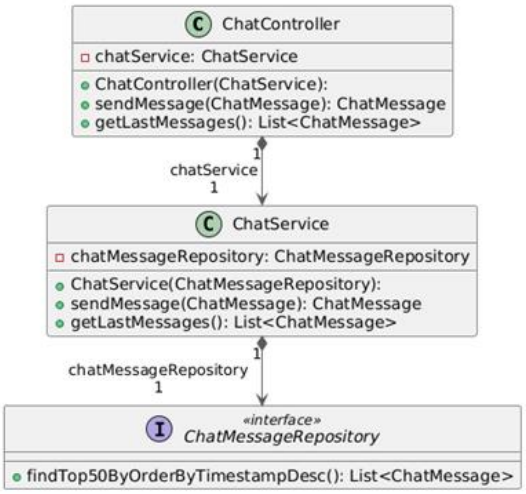
10

ДІАГРАМА КЛАСІВ ФУНКЦІОНАЛУ КОРИСТУВАЧІВ



11

ДІАГРАМА КЛАСІВ ГЛОБАЛЬНОГО ЧАТУ



12

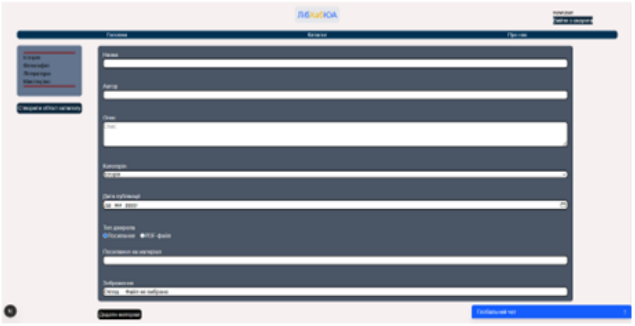
ЕКРАННІ ФОРМИ

Головна сторінка вебзастосунку з чатом

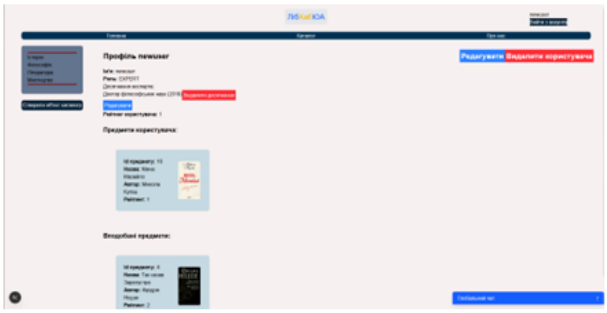
Сторінка матеріалу

13

ЕКРАННІ ФОРМИ



Форма додавання власного матеріалу



Профіль користувача

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Варич М.Ю., Андрусевич Н.В. ВИЗНАЧЕННЯ ВИМОГ ДО ВЕБДОДАТКУ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ ІЗ СОЦІАЛЬНОЮ ВЗАЄМОДІЄЮ. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с. 51-52.
2. Варич М.Ю., Андрусевич Н.В. АРХІТЕКТУРА ВЕБДОДАТКУ ЕЛЕКТРОННОЇ БІБЛІОТЕКИ ІЗ СОЦІАЛЬНОЮ ВЗАЄМОДІЄЮ. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с. 53-54.

15

ВИСНОВКИ

1. Проведено дослідження існуючих методів та рішень для впровадження та забезпечення соціальної взаємодії користувачів web-застосунків задля покращення соціальної взаємодії користувачів електронних бібліотек. Визначено відповідні рішення та методи.
2. Здійснено огляд та аналіз існуючих електронних бібліотек. Виявлено основний функціонал та недоліки.
3. Проведено огляд засобів для реалізації програмного забезпечення клієнт-серверного вебзастосунку електронної бібліотеки із соціальною взаємодією, серед них обрано відповідні для реалізації задачі кваліфікаційної роботи.
4. Сформовано вимоги до розробки клієнт-серверного web-застосунку електронної бібліотеки.
5. Спроектовано архітектуру клієнт-серверного web-застосунку електронної бібліотеки.
6. Розроблено клієнт-серверний web-застосунок електронної бібліотеки із соціальною взаємодією.
7. Проведено тестування клієнт-серверного web-застосунку електронної бібліотеки.

16

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Сервісні класи серверної частини:

```
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.*;
import
org.mvar.social_elib_project.payload.request.user.ExpertAcco
mplishmentRequest;
import org.mvar.social_elib_project.repository.UserRepository;
import org.springframework.stereotype.Service;

import java.util.*;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
    private final AuthService authService;

    public void deleteCurrentUser() {
        User user = authService.getAuthenticatedUser();
        userRepository.delete(user);
    }

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User getCurrentUser() {
        User user = authService.getAuthenticatedUser();
        return userRepository.findUserByEmail(user.getEmail())
                .orElseThrow(() -> new
IllegalArgumentException("User not found: " +
user.getEmail()));
    }


    @Transactional
    public User changeUsername(String newUsername) {
        User currentUser = authService.getAuthenticatedUser();
        if
(userRepository.findUserByUsersname(newUsername).isPrese
nt() &&

!userRepository.findUserByUsersname(newUsername).get().ge
tId().equals(currentUser.getId())) {
            throw new IllegalArgumentException("Username
already exists: " + newUsername);
        }
        currentUser.setUsersname(newUsername);
        return userRepository.save(currentUser);
    }

    public Role getUserRole() {
        User user = authService.getAuthenticatedUser();
        return user.getRole();
    }

    public Optional<User> getUserByUsername(String
username) {
        return userRepository.findUserByUsersname(username);
    }

    public User favourItem(Long itemId) {
        User user = authService.getAuthenticatedUser();
        if (user.getFavouredItems().contains(itemId)) {
            throw new IllegalStateException("User has already
favoured this item");
        }
        if (user.getFavouredItems().isEmpty()) {
            user.setFavouredItems(new HashSet<>());
        }
        user.getFavouredItems().add(itemId);
        return userRepository.save(user);
    }

    public User unfavourItem(Long itemId) {
        User user = authService.getAuthenticatedUser();
        if (!user.getFavouredItems().contains(itemId)) {
            throw new IllegalStateException("User did not favour
this item!");
        }
        user.getFavouredItems().remove(itemId);
        return userRepository.save(user);
    }

    public boolean checkFavoriteStatus(Long itemId) {
        User user = authService.getAuthenticatedUser();
        return user.getFavouredItems().contains(itemId);
    }

    public User
addExpertAccomplishment(ExpertAccomplishmentRequest
request) {
        User user = authService.getAuthenticatedUser();
        authService.checkExpertRole();
        if(user.getExpertAccomplishments() == null) {
            user.setExpertAccomplishments(new HashSet<>());
        }

user.getExpertAccomplishments().add(request.accomplishment
());
        return userRepository.save(user);
    }

    public User
removeExpertAccomplishment(ExpertAccomplishmentRequest
request) {
        User user = authService.getAuthenticatedUser();
        authService.checkExpertRole();

user.getExpertAccomplishments().remove(request.accomplish
ment());
        return userRepository.save(user);
    }

    public Set<String> getExpertAccomplishments(String
username) {
        User user =
userRepository.findUserByUsersname(username)
                .orElseThrow(() -> new
IllegalArgumentException("User with username not found: " +
username));
        Set<String> expertAccomplishments =
user.getExpertAccomplishments();
```

```
        System.out.println(expertAccomplishments);
        if (expertAccomplishments == null ||
expertAccomplishments.isEmpty()) {
            return Collections.emptySet();
        }
        System.out.println(expertAccomplishments);
        return user.getExpertAccomplishments();
    }
}
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.Token;
import
org.mvar.social_elib_project.repository.TokenRepository;
import org.springframework.stereotype.Service;

import java.util.Date;

@Service
@RequiredArgsConstructor
public class TokenService {
    private final TokenRepository tokenRepository;

    public Token createToken(String token, Date
expirationDate) {
        if (!tokenRepository.existsByJwt(token)) {
            return tokenRepository.save(
                    Token.builder()
                            .jwt(token)
                            .isValid(true)
                            .expireDate(expirationDate)
                            .build()
            );
        }
        throw new NullPointerException();
    }

    public void removeInvalidTokens() {
        tokenRepository.deleteTokenByExpireDate(new Date());
    }

    public void invalidateToken(String jwtToken) {
        Token storedToken = getTokenInformation(jwtToken);
        storedToken.setValid(false);
        tokenRepository.save(storedToken);
    }

    public Token getTokenInformation(String jwtToken) {
        return tokenRepository.findByJwt(jwtToken).orElse(null);
    }
}
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;
import io.jsonwebtoken.Jwts;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;


@Service
@RequiredArgsConstructor
public class JwtService {
    private final String SECRET_KEY =
System.getenv("JWT_SECRET");
    private final TokenService tokenService;

    public String extractEmail(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims,
T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSigningkey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSigningkey() {
        byte[] keyBytes =
Decoders.BASE64.decode(SECRET_KEY);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    public String generateToken(Map<String, Object> claims,
UserDetails userDetails) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis()
+ 1000 * 60 * 60 * 24 * 7))
                .signWith(getSigningkey(),
SignatureAlgorithm.HS256)
                .compact();
    }

    public boolean validateToken(String token, UserDetails
userDetails) {
        final String email = extractEmail(token);
        return email.equals(userDetails.getUsername()) &&
!isTokenExpired(token) &&
tokenService.getTokenInformation(token).isValid();
    }

    private boolean isTokenExpired(String token) {
        return extractExpire(token).before(new Date());
    }

    private Date extractExpire(String token) {
        return extractClaim(token, Claims::getExpiration);
    }
}
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.Item;
import org.mvar.social_elib_project.model.Role;
import org.mvar.social_elib_project.model.User;
import
```

```
org.mvar.social_elib_project.payload.request.item.AddItemReq
uest;
import
org.mvar.social_elib_project.payload.request.item.UpdateItem
Request;
import
org.mvar.social_elib_project.repository.CommentRepository;
import org.mvar.social_elib_project.repository.ItemRepository;
import org.mvar.social_elib_project.repository.UserRepository;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.time.LocalDateTime;
import java.util.*;

@Service
@RequiredArgsConstructor
public class ItemService {
    private final ItemRepository itemRepository;
    private final UserRepository userRepository;
    private final CommentRepository commentRepository;
    private final IdCounterService idCounterService;
    private final ImageService imageService;
    private final FileUploadService fileUploadService;
    private final AuthService authService;

    public Item createNewItem(AddItemRequest request,
MultipartFile image, MultipartFile pdf) throws IOException {
        User user = authService.getAuthenticatedUser();
        String materialLink;
        if (pdf != null && !pdf.isEmpty()) {
            materialLink = fileUploadService.savePdf(pdf);
        } else if (request.materialLink() != null &&
!request.materialLink().isBlank()) {
            materialLink = request.materialLink();
        } else {
            throw new IllegalArgumentException("Either PDF file
or materialLink must be provided");
        }

        if
(itemRepository.findItemByMaterialLink(materialLink).isPrese
nt()) {
            throw new IllegalArgumentException("Material with
the same PDF link already exists");
        }

        String imageUrl = (image != null && !image.isEmpty()) ?
imageService.saveImage(image) : null;

        Item item = Item.builder()
                .name(request.name())
                .author(request.author())
                .description(request.description())
                .category(request.category())
                .publishDate(request.publishDate())
                .materialLink(materialLink)
                .image(imageUrl)
                .user(user.getUsersname())
                .usersWhoVoted(new HashSet<>())
                .expertComment(new HashSet<>())
                .build();

        item.setCreationDate(LocalDateTime.now());

item.setItemId(idCounterService.generateSequence("items_seq
uence"));
        return itemRepository.save(item);
    }


    public void deleteItem(Long id) {
        Item item = itemRepository.findItemByItemId(id)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + id));
        User user = authService.getAuthenticatedUser();
        if (!item.getUser().equals(user.getUsersname())) {
            throw new IllegalArgumentException("User not
authorized to perform action");
        }
        itemRepository.deleteByItemId(id);
        commentRepository.deleteAllByItemId(id);
    }

    public List<Item> getAllItems() {
        return itemRepository.findAll();
    }

    public Optional<Item> getItemByItemId(Long id) {
        return itemRepository.findItemByItemId(id);
    }

    public List<Item> getItemsByUser(String username) {
return itemRepository.findItemsByUser(username); }

    public List<Item> getFavouredItemsByUser(String
username) {
        User user =
userRepository.findUserByUsersname(username)
                .orElseThrow(() -> new
IllegalArgumentException("User with username not found: " +
username));
        Set<Long> favouredItemIds = user.getFavouredItems();
        System.out.println(favouredItemIds);
        if (favouredItemIds == null || favouredItemIds.isEmpty())
{
            return Collections.emptyList();
        }

System.out.println(itemRepository.findItemsByItemId(favoure
dItemIds));
        return
itemRepository.findAllByItemIdIn(favouredItemIds);
    }

    public List<Item> getItemsByCategory(String category) {
        return itemRepository.findItemsByCategory(category);
    }

    @Transactional
    public Item updateItem(UpdateItemRequest
updateItemRequest, long itemId, MultipartFile image) throws
IOException {
        Item item = itemRepository.findItemByItemId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found"));
        if(!checkUserItemPermission(itemId)) {
            throw new IllegalArgumentException("You do not have
permission to perform action");
        }
        if (updateItemRequest != null) {
            if (updateItemRequest.name() != null) {
                item.setName(updateItemRequest.name());
            }
            if (updateItemRequest.author() != null) {
                item.setAuthor(updateItemRequest.author());
            }
```

```
        if (updateItemRequest.description() != null) {
item.setDescription(updateItemRequest.description());
        }
        if (updateItemRequest.category() != null) {
            item.setCategory(updateItemRequest.category());
        }
        if (updateItemRequest.publishDate() != null) {
item.setPublishDate(updateItemRequest.publishDate());
        }
        if (updateItemRequest.materialLink() != null) {
item.setMaterialLink(updateItemRequest.materialLink());
        }
    }
    if (image != null && !image.isEmpty()) {
        String imageUrl = imageService.saveImage(image);
        item.setImage(imageUrl);
    }

    return itemRepository.save(item);
  }



  public boolean checkUserItemPermission(Long itemId) {
    Item item = itemRepository.findItemByItemId(itemId)
        .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
    User user = authService.getAuthenticatedUser();
    return item.getUser().equals(user.getUsersname());
  }

  public Item voteItem(Long itemId, String user, int vote) {
    Item item = itemRepository.findItemByItemId(itemId)
        .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
    User author =
userRepository.findUserByUsersname(item.getUser())
        .orElseThrow(() -> new
IllegalArgumentException("Item creator not found: " +
item.getUser()));
    if (item.getUsersWhoVoted().contains(user)) {
        throw new IllegalStateException("User has already
voted for this item");
    }
    if (vote != 1 && vote != -1) {
        throw new IllegalArgumentException("Invalid vote
value. Must be 1 or -1");
    }
    item.getUsersWhoVoted().add(user);
    author.setUserRating(author.getUserRating() + vote);
    if(author.getUserRating() > 50) {
        author.setRole(Role.RESPECTED_USER);
    }
    if(author.getUserRating() == 50 && vote == -1) {
        author.setRole(Role.USER);
    }
    userRepository.save(author);
    item.setRating(item.getRating() + vote);
    return itemRepository.save(item);
  }

  public Item unvoteItem(Long itemId, String username) {
    Item item = itemRepository.findItemByItemId(itemId)
        .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
    if (!item.getUsersWhoVoted().contains(username)) {
        throw new IllegalStateException("User has not voted
for this item");
    }
    item.setRating(item.getRating() -
(Integer.compare(item.getRating(), 0)));
    item.getUsersWhoVoted().remove(username);
    return itemRepository.save(item);
  }

  public boolean checkIfUserVoted(Long itemId) {
    Item item = itemRepository.findItemByItemId(itemId)
        .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
    User user = authService.getAuthenticatedUser();
    return
item.getUsersWhoVoted().contains(user.getEmail());
  }
}
import com.mongodb.client.gridfs.model.GridFSFile;
import org.bson.types.ObjectId;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.data.mongodb.gridfs.GridFsOperations;
import
org.springframework.data.mongodb.gridfs.GridFsTemplate;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.io.InputStream;

import static
org.springframework.data.mongodb.core.query.Criteria.where;
import static
org.springframework.data.mongodb.core.query.Query.query;

@Service
public class ImageService {

  @Autowired
  private GridFsTemplate gridFsTemplate;

  @Autowired
  private GridFsOperations gridFsOperations;

  public String saveImage(MultipartFile file) throws
IOException {
    if (file == null || file.isEmpty()) {
        throw new IllegalArgumentException("File is empty or
null");
    }

    System.out.println("Saving file: " +
file.getOriginalFilename());

    try (InputStream inputStream = file.getInputStream()) {
        ObjectId id = gridFsTemplate.store(inputStream,
file.getOriginalFilename(), file.getContentType());
        System.out.println("Stored with ID: " + id);
        return id.toHexString();
    }
  }


  public GridFSFile getImageById(String id) {
    return
gridFsTemplate.findOne(query(where("_id").is(new
ObjectId(id))));
```

```
    }

    public InputStream getImageStream(GridFSFile file) throws
IOException {
        return
gridFsOperations.getResource(file).getInputStream();
    }

    public String getContentType(GridFSFile file) {
        return
gridFsOperations.getResource(file).getContentType();
    }
}
import jakarta.persistence.Id;
import lombok.*;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.data.mongodb.core.MongoOperations;
import
org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Component;

import static
org.springframework.data.mongodb.core.FindAndModifyOptio
ns.options;
import static
org.springframework.data.mongodb.core.query.Criteria.where;
import static
org.springframework.data.mongodb.core.query.Query.query;

@Component
public class IdCounterService {

    @Autowired
    private MongoOperations mongoOperations;

    public long generateSequence(String seqName) {
        DatabaseSequence counter =
mongoOperations.findAndModify(
                query(where("_id").is(seqName)),
                new Update().inc("seq", 1),
                options().returnNew(true).upsert(true),
                DatabaseSequence.class
        );
        return counter != null ? counter.getSeq() : 1;
    }

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    @Document(collection = "counter")
    public static class DatabaseSequence {
        @Id
        private String id;
        private long seq;
    }
}
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.UUID;

@Service
public class FileUploadService {
    private final String uploadDir = "uploads/pdf";

    public String savePdf(MultipartFile file) throws IOException
{
        if (!file.getOriginalFilename().endsWith(".pdf")) {
            throw new IllegalArgumentException("Only PDF files
are allowed.");
        }

        File dir = new File(uploadDir);
        if (!dir.exists()) {
            dir.mkdirs();
        }

        String filename = UUID.randomUUID() + "-" +
file.getOriginalFilename();
        Path path = Paths.get(uploadDir, filename);
        Files.copy(file.getInputStream(), path,
StandardCopyOption.REPLACE_EXISTING);

        return "/files/pdf/" + filename;
    }
}
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.repository.UserRepository;
import
org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsServi
ce;
import
org.springframework.security.core.userdetails.UsernameNotFo
undException;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class CustomUserService implements
UserDetailsService {
    private final UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
        return userRepository.findUserByEmail(username).get();
    }
}
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.*;
import
org.mvar.social_elib_project.payload.request.comment.AddCo
mmentRequest;
import
org.mvar.social_elib_project.payload.request.comment.AddEx
pertCommentRequest;
import
org.mvar.social_elib_project.payload.request.comment.Update
CommentRequest;
import
org.mvar.social_elib_project.repository.CommentRepository;
import org.mvar.social_elib_project.repository.ItemRepository;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
```

```
import java.util.Set;

@Service
@RequiredArgsConstructor
public class CommentService {
    private final CommentRepository commentRepository;
    private final ItemRepository itemRepository;
    private final IdCounterService idCounterService;
    private final AuthService authService;

    public Comment addCommentToItem(AddCommentRequest
addCommentRequest, long itemId) {
        Item item = itemRepository.findItemByItemId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
        User user = authService.getAuthenticatedUser();
        Comment comment = Comment.builder()
                .text(addCommentRequest.text())
                .itemId(item.getItemId())
                .user(user.getUsersname())
                .build();
        comment.setDate(LocalDateTime.now());

comment.setCommentId(idCounterService.generateSequence("
comments_sequence"));
        return commentRepository.save(comment);
    }

    public void deleteComment(long id) {
        Comment comment =
commentRepository.findCommentByCommentId(id)
                .orElseThrow(() -> new
IllegalArgumentException("Comment not found: " + id));
        User user = authService.getAuthenticatedUser();
        if (!comment.getUser().equals(user.getUsersname())) {
            throw new IllegalArgumentException("User not
authorized to delete comment");
        }
        commentRepository.deleteByCommentId(id);
    }

    public boolean checkUserCommentPermission(Long itemId)
{
        Comment comment =
commentRepository.findCommentByCommentId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Comment not found: " + itemId));
        User user = authService.getAuthenticatedUser();
        return comment.getUser().equals(user.getUsersname());
    }

    @Transactional
    public Comment
updateCommentText(UpdateCommentRequest request, Long
id) {
        Comment comment =
commentRepository.findCommentByCommentId(id)
                .orElseThrow(() -> new
IllegalArgumentException("Comment not found: " + id));
        User user = authService.getAuthenticatedUser();
        if (!comment.getUser().equals(user.getUsersname())) {
            throw new IllegalArgumentException("User not
authorized to perform action");
        }
        comment.setText(request.text());
        return commentRepository.save(comment);
    }

    public List<Comment> getCommentsByItem(long itemId) {
        return
commentRepository.findCommentsByItemId(itemId);
    }

    public Item
addExpertCommentToItem(AddExpertCommentRequest
addExpertCommentRequest, long itemId) {
        Item item = itemRepository.findItemByItemId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
        User user = authService.getAuthenticatedUser();
        authService.checkExpertRole();
        if(!authService.checkExpertRole()) {
            throw new IllegalArgumentException("User not
authorized to add expert comment");
        }
        ExpertComment expertComment =
ExpertComment.builder()
                .text(addExpertCommentRequest.text())
                .itemId(item.getItemId())
                .user(user.getUsersname())
                .creationDate(LocalDateTime.now())
                .build();
        item.getExpertComment().add(expertComment);

expertComment.setExpertCommentId(idCounterService.genera
teSequence("expert_comments_sequence"));
        return itemRepository.save(item);
    }

    @Transactional
    public void deleteExpertComment(long expertCommentId) {
        User user = authService.getAuthenticatedUser();
        authService.checkExpertRole();
        if(!authService.checkExpertRole()) {
            throw new IllegalArgumentException("User not
authorized to add expert comment");
        }
        Optional<Item> itemOptional =
itemRepository.findAll().stream()
                .filter(item -> item.getExpertComment().stream()
                        .anyMatch(comment ->
comment.getExpertCommentId() == expertCommentId))
                .findFirst();
        if (itemOptional.isEmpty()) {
            throw new IllegalArgumentException("ExpertComment
not found in any item: " + expertCommentId);
        }

        Item item = itemOptional.get();

        ExpertComment commentToDelete =
item.getExpertComment().stream()
                .filter(comment -> comment.getExpertCommentId()
== expertCommentId)
                .findFirst()
                .orElseThrow(() -> new
IllegalArgumentException("ExpertComment not found: " +
expertCommentId));

        if
(!commentToDelete.getUser().equals(user.getUsersname())) {
            throw new IllegalArgumentException("User not
authorized to delete this comment");
        }
        item.getExpertComment().remove(commentToDelete);
        itemRepository.save(item);
    }
```

```
    public boolean checkExpertCommentPermission(Long
expertCommentId) {
        User user = authService.getAuthenticatedUser();
        Optional<Item> itemOptional =
itemRepository.findAll().stream()
                .filter(item -> item.getExpertComment().stream()
                        .anyMatch(comment ->
comment.getExpertCommentId() == expertCommentId))
                .findFirst();
        if (itemOptional.isEmpty()) {
            throw new IllegalArgumentException("Expert comment
not found: " + expertCommentId);
        }
        Item item = itemOptional.get();
        return item.getExpertComment().stream()
                .filter(comment -> comment.getExpertCommentId()
== expertCommentId)
                .anyMatch(comment ->
comment.getUser().equals(user.getUsersname()));
    }


    @Transactional
    public void
updateExpertCommentText(UpdateCommentRequest request,
Long expertCommentId) {
        User user = authService.getAuthenticatedUser();
        Optional<Item> itemOptional =
itemRepository.findAll().stream()
                .filter(item -> item.getExpertComment().stream()
                        .anyMatch(comment ->
comment.getExpertCommentId() == expertCommentId))
                .findFirst();

        if (itemOptional.isEmpty()) {
            throw new IllegalArgumentException("ExpertComment
not found in any item: " + expertCommentId);
        }
        Item item = itemOptional.get();
        ExpertComment commentToUpdate =
item.getExpertComment().stream()
                .filter(comment -> comment.getExpertCommentId()
== expertCommentId)
                .findFirst()
                .orElseThrow(() -> new
IllegalArgumentException("ExpertComment not found: " +
expertCommentId));
        if
(!commentToUpdate.getUser().equals(user.getUsersname())) {
            throw new IllegalArgumentException("User not
authorized to perform action");
        }
        commentToUpdate.setText(request.text());
        itemRepository.save(item);
    }


    public Set<ExpertComment>
getExpertCommentsByItemId(long itemId) {
        Item item = itemRepository.findItemByItemId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
        return item.getExpertComment();
    }
}
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.ChatMessage;
import
org.mvar.social_elib_project.repository.ChatMessageRepositor
y;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;

@Service
@RequiredArgsConstructor
public class ChatService {
    private final ChatMessageRepository
chatMessageRepository;

    public ChatMessage sendMessage(ChatMessage message) {
        message.setTimestamp(LocalDateTime.now());
        return chatMessageRepository.save(message);
    }

    public List<ChatMessage> getLastMessages() {
        return
chatMessageRepository.findTop50ByOrderByTimestampDesc(
);
    }
}
import io.jsonwebtoken.Claims;
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.Role;
import org.mvar.social_elib_project.model.User;
import
org.mvar.social_elib_project.payload.request.auth.AuthRequest
;
import
org.mvar.social_elib_project.payload.request.auth.RegisterReq
uest;
import
org.mvar.social_elib_project.payload.response.AuthResponse;
import org.mvar.social_elib_project.repository.UserRepository;
import
org.springframework.security.authentication.AuthenticationMa
nager;
import
org.springframework.security.authentication.UsernamePasswor
dAuthenticationToken;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHol
der;
import
org.springframework.security.core.userdetails.UsernameNotFo
undException;
import
org.springframework.security.crypto.password.PasswordEncod
er;
import org.springframework.stereotype.Service;

import java.util.HashSet;

@Service
@RequiredArgsConstructor
public class AuthService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;
    private final TokenService tokenService;

    public AuthResponse register(RegisterRequest
registerRequest) {
```

```
if(userRepository.findUserByUsersname(registerRequest.usern
ame()).isPresent()) {
        throw new IllegalArgumentException("Username is
already in use");
    }

if(userRepository.findUserByEmail(registerRequest.email()).is
Present()) {
        throw new IllegalArgumentException("Email is already
in use");
    }
    User user = User.builder()
            .email(registerRequest.email())
            .usersname(registerRequest.username())

.password(passwordEncoder.encode(registerRequest.password(
)))
            .isActive(true)
            .role(Role.USER)
            .favouredItems(new HashSet<>())
            .expertAccomplishments(new HashSet<>())
            .userRating(0)
            .build();
    userRepository.save(user);

    String jwtToken = jwtService.generateToken(user);
    tokenService.createToken(jwtToken,
jwtService.extractClaim(jwtToken, Claims::getExpiration));

    return AuthResponse.builder()
            .token(jwtToken)
            .build();
  }

  public AuthResponse authenticate(AuthRequest
authRequest) {
    User user =
userRepository.findUserByEmail(authRequest.email())
            .orElseThrow(() -> new
UsernameNotFoundException(authRequest.email()));
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                authRequest.email(),
                authRequest.password()
            )
    );

    String jwtToken = jwtService.generateToken(user);
    tokenService.createToken(jwtToken,
jwtService.extractClaim(jwtToken, Claims::getExpiration));

    return AuthResponse.builder()
            .token(jwtToken)
            .build();
  }

  public void logout(String token) {
    tokenService.invalidateToken(token);
  }

  public User getAuthenticatedUser() {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication == null ||
!authentication.isAuthenticated()) {
        throw new IllegalStateException("User is not
authenticated");
    }
    String email = authentication.getName();
    return userRepository.findUserByEmail(email)
            .orElseThrow(() -> new
IllegalArgumentException("User with email not found: " +
email));
  }

  public boolean checkExpertRole() {
    Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    boolean isExpert =
authentication.getAuthorities().stream()
            .anyMatch(auth ->
auth.getAuthority().equals(Role.EXPERT.name()));
    if (!isExpert) {
        throw new IllegalStateException("User does not have
permission to add expert comment");
    }
    return isExpert;
  }
}
import lombok.RequiredArgsConstructor;
import org.mvar.social_elib_project.model.*;
import
org.mvar.social_elib_project.payload.request.admin.AdminCha
ngeRoleRequest;
import
org.mvar.social_elib_project.repository.CommentRepository;
import
org.mvar.social_elib_project.repository.ExpertCommentReposi
tory;
import org.mvar.social_elib_project.repository.ItemRepository;
import org.mvar.social_elib_project.repository.UserRepository;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHol
der;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
@RequiredArgsConstructor
public class AdminService {
  private final UserRepository userRepository;
  private final CommentRepository commentRepository;
  private final ItemRepository itemRepository;
  private final ExpertCommentRepository
expertCommentRepository;

  public void deleteUserByAdmin(String username) {
    adminAuth();
    User userToDelete =
userRepository.findUserByUsersname(username)
            .orElseThrow(() -> new
IllegalArgumentException("User not found: " + username));
    userRepository.delete(userToDelete);
  }

  public User updateUserRole(AdminChangeRoleRequest
request) {
    adminAuth();
    User userToUpdate =
userRepository.findUserByUsersname(request.user())
            .orElseThrow(() -> new
IllegalArgumentException("User not found: " +
request.user()));
    userToUpdate.setRole(request.role());
    return userRepository.save(userToUpdate);
  }
```

```
    public void deleteCommentByAdmin(long commentId) {
        adminAuth();
        Comment comment =
commentRepository.findCommentByCommentId(commentId)
                .orElseThrow(() -> new
IllegalArgumentException("Comment not found: " +
commentId));
        commentRepository.delete(comment);
    }

    public void deleteExpertCommentByAdmin(long
expertCommentId) {
        adminAuth();
        Optional<Item> itemOptional =
itemRepository.findAll().stream()
                .filter(item -> item.getExpertComment().stream()
                        .anyMatch(comment ->
comment.getExpertCommentId() == expertCommentId))
                .findFirst();
        if (itemOptional.isEmpty()) {
            throw new IllegalArgumentException("ExpertComment
not found in any item: " + expertCommentId);
        }
        Item item = itemOptional.get();
        ExpertComment commentToDelete =
item.getExpertComment().stream()
                .filter(comment -> comment.getExpertCommentId()
== expertCommentId)
                .findFirst()
                .orElseThrow(() -> new
IllegalArgumentException("ExpertComment not found: " +
expertCommentId));
        item.getExpertComment().remove(commentToDelete);
        itemRepository.save(item);
    }

    public void deleteItemByAdmin(long itemId) {
        adminAuth();
        Item item = itemRepository.findItemByItemId(itemId)
                .orElseThrow(() -> new
IllegalArgumentException("Item not found: " + itemId));
        itemRepository.delete(item);
        commentRepository.deleteAllByItemId(itemId);
        expertCommentRepository.deleteAllByItemId(itemId);
    }

    private void adminAuth() {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null ||
!authentication.isAuthenticated()) {
            throw new IllegalStateException("User is not
authenticated");
        }
        boolean isAdmin =
authentication.getAuthorities().stream()
                .anyMatch(auth ->
auth.getAuthority().equals(Role.ADMIN.name()));
        if (!isAdmin) {
            throw new SecurityException("User does not have
permission to update user roles");
        }
    }
}
```

Файли сторінок клієнтської частини:

```
export default async function AboutPage(){
    return (
        <div className="justify-items-center">
            <h1 className="text-5xl text-center">Вітаємо!</h1>
            <br></br>
            <h2 className="text-3xl text-center">Це - електронна
бібліотека із соціальною взаємодією. Тут ви можете:</h2>
            <br></br>
            <ul className="text-1xl">
                <li>Переглядати користувацькі матеріали
бібліотеки;</li>
                <li>Доповнювати бібліотеку власними
матеріалами;</li>
                <li>Залишати власні коментарі до предметів;</li>
                <li>Оцінювати предмети;</li>
                <li>Надавати особливий коментар (якщо ви маєте
статус експерта);</li>
            </ul>
        </div>
    )
}
"use client"

import React, {useEffect, useState} from "react";
import {addItem} from "@/service/api";
import { useRouter } from 'next/navigation';

export default function AddItemPage() {
    const [itemName, setItemName] = useState("")
    const [itemAuthor, setItemAuthor] = useState("")
    const [itemDescription, setItemDescription] = useState("")
    const [itemCategory, setItemCategory] = useState("")
    const [itemPublishDate, setItemPublishDate] = useState("")
    const [itemMaterialLink, setItemMaterialLink] =
useState("")
    const [itemImage, setItemImage] = useState<File |
null>(null)
    const [sourceType, setSourceType] = useState<"link" |
"file">("link");
    const [materialFile, setMaterialFile] = useState<File |
null>(null);

    const router = useRouter()
    useEffect(() => {
        const storedToken = localStorage.getItem('token')
        if (storedToken) {
            setLoggedIn(true);
        }
    }, []);

    const [loggedIn, setLoggedIn] = useState(false);

    const handleSubmit = async (e: React.FormEvent) => {
        e.preventDefault();
        const token = localStorage.getItem('token');
        if (!token) {
            alert("Щоб створити книгу, потрібно увійти!");
            return;
        }
        if (itemMaterialLink.includes('.ru')) {
            alert("Публікувати російські джерела заборонено!");
            return;
        }
        if (itemMaterialLink.includes('.by')) {
            alert("Публікувати білоруські джерела
заборонено!");
            return;
        }

        const formData = new FormData();
        formData.append("data", JSON.stringify({
            name: itemName,
```

```
        author: itemAuthor,
        description: itemDescription,
        category: itemCategory,
        publishDate: itemPublishDate,
        materialLink: sourceType === "link" ?
itemMaterialLink : ""
      }));
      if (itemImage) formData.append("image", itemImage);
      if (sourceType === "file" && materialFile)
formData.append("pdf", materialFile);

      const response = await addItem(formData, token);
      if (response) {
        setItemName("");
        setItemAuthor("");
        setItemDescription("");
        setItemCategory("");
        setItemPublishDate("");
        setItemMaterialLink("");
        setItemImage(null);
        router.push(`/catalog/${response.itemId}`);
      } else {
        alert("Не вдалося додати книгу");
      }
    };
    return (
      <div>
        {!loggedIn && (
          <p>Ви не авторизовані</p>
        )}
        {loggedIn && (
          <form onSubmit={handleSubmit} className="w-
[90%]">
            <h2 className='text-black mt-[-10%]'>Додати
предмет</h2>
            <br></br>
            <div className='text-white mt-[7%] w-[100%]
border-15 border-gray-600 bg-gray-600 rounded-lg'>
              <h3 className="">Назва</h3>
              <input name='name'
                  className='w-[100%] rounded-lg bg-white
border-2 border-solid border-black text-black'
                  onChange={(e) =>
setItemName(e.target.value)} required
                  type='text'
                  size={50}></input>
              <br></br>
              <h3 className='mt-[3%]'>Автор</h3>
              <input name='author'
                  className='w-[100%] rounded-lg bg-white
border-2 border-solid border-black text-black'
                  onChange={(e) =>
setItemAuthor(e.target.value)} required
                  type='text'
                  size={50}></input>
              <br></br>
              <h3 className='mt-[3%]'>Опис</h3>
              <textarea value={itemDescription}
                    className='w-[100%] rounded-lg bg-
white border-2 border-solid border-black text-black'
                  onChange={(e) =>
setItemDescription(e.target.value)}
                  required
                  rows={3}
                  maxLength={700}
                  placeholder={'Опис'}>
              </textarea>
              <h3 className='mt-[3%]'>Категорія</h3>
              <select name='category'
                    className='w-[100%] rounded-lg bg-
white border-2 border-solid border-black text-black'
                    onChange={(e) =>
setItemCategory(e.target.value)} required>
                  <option value="history">Історія</option>
                  <option
value="philosophy">Філософія</option>
                  <option value="art">Мистецтво</option>
                  <option
value="literature">Література</option>
                </select>
                <br></br>
                <h3 className='mt-[3%]'>Дата
публікації</h3>
                <input name='publishdate'
                    className='w-[100%] rounded-lg bg-white
border-2 border-solid border-black text-black'
                    onChange={(e) =>
setItemPublishDate(e.target.value)} required
                    type='date'
                    size={50}></input>
                <br></br>
                <h3 className='mt-[3%]'>Тип джерела</h3>
                <div className='flex gap-4'>
                  <label className='text-white'>
                    <input type="radio" value="link"
checked={sourceType === "link"}
                      onChange={() =>
setSourceType("link")}/>
                    Посилання
                  </label>
                  <label className='text-white'>
                    <input type="radio" value="file"
checked={sourceType === "file"}
                      onChange={() =>
setSourceType("file")}/>
                    PDF-файл
                  </label>
                </div>
                <br></br>
                {sourceType === "link" ? (
                  <>
                    <h3 className="">Посилання на
матеріал</h3>
                    <input name="materiallink"
                      pattern="^(?!.*\.(ru|by)(/|$)).*$"
                      title="Посилання не має бути з
ворожих джерел (.ru, .by)"
                      className="w-[100%] rounded-lg bg-
white border-2 border-solid border-black text-black"
                      onChange={(e) =>
setItemMaterialLink(e.target.value)}
                      required={sourceType === "link"}
                      type="url"
                      size={50}
                    />
                  </>
                ) : (
                  <>
                    <h3 className="">Завантажити PDF-
файл</h3>
                    <input
                      name="materialFile"
                      type="file"
                      accept="application/pdf"
                      className="w-[100%] rounded-lg bg-
white border-2 border-solid border-black text-black"
                      onChange={(e) => {
                        const file = e.target.files?.[0] ?? null;
```

```
                            setMaterialFile(file);
                        }}
                        required={sourceType === "file"}
                    />
                </>
            )}
            <br></br>
            <h3 className='mt-[3%]'>Зображення</h3>
            <input
                name='image'
                type='file'
                accept='image/*'
                className='w-[100%] rounded-lg bg-white
border-2 border-solid border-black text-black'
                onChange={(e) => {
                    const file = e.target.files?.[0] ?? null;
                    setItemImage(file);
                }}
            />
            <br></br>
        </div>
        <br></br>
        <button type="submit" className='text-white bg-
black rounded-lg border-3 border-black'>Додати
            матеріал
        </button>
    </form>
    )}
  </div>
 )
}
import AdminActions from
"@/components/admin/AdminActions";

export default async function AdminPage() {
  return (
    <div>
      <AdminActions/>
    </div>
  )
}
import { getItem } from "@/service/api";
import UpdateItemForm from
"@/components/items/UpdateItemForm";

interface EditItemPageProps {
  params: { id: string };
}

export default async function EditItemPage({ params }:
EditItemPageProps) {
  const itemId = params.id;

  if (!itemId) {
    return <div>Некоректний ID предмета</div>;
  }

  const item = await getItem(itemId);

  if (!item) {
    return <div>Предмет не знайдено</div>;
  }

  return (
    <div>
      <UpdateItemForm itemId={itemId} />
    </div>
  );
}
```

```
import { getItem, getItemComments } from "@/service/api";
import { notFound } from "next/navigation";
import React from "react";
import CommentForm from
"@/components/items/CommentForm";
import Link from "next/link";
import ExpertAddCommentButton from
"@/components/items/ExpertAddCommentButton";
import VoteItem from "@/components/items/VoteItem";
import CommentElement from
"@/components/items/Comment";
import ExpertCommentElement from
"@/components/items/ExpertCommentElement";
import UpdateItemButton from
"@/components/items/UpdateItemButton";
import DeleteItemButton from
"@/components/items/DeleteItemButton";
import ItemImage from "@/components/items/ItemImage";
import FavourItem from "@/components/items/FavourItem";

type Props = {
  params: {
    id: string;
  };
};

export default async function CatalogItemPage({params}:
Props) {

  const id = params.id

  const item = await getItem(id);
  const comments = await getItemComments(id);

  const categoryMap: Record<string, string> = {
    "history": "Історія",
    "philosophy": "Філософія",
    "art": "Мистецтво",
    "literature": "Література",
  };


  if (!item) {
    notFound();
  }


  return (
    <div
      className="text-[#35281D] items-center justify-items-
center mt-[-5%] p-8 pb-20 gap-16 sm:p-20 font-[family-
name:var(--font-geist-sans)]">
      <div
        className="w-[85%] rounded-xl border-4 border-
[#DBDBDB] bg-[#DBDBDB] items-center justify-items-center
grid grid-cols-2 gap-4">
        <div className="bg-[#DBDBDB] border-2 w-
[50%] min-h-[50%] mt-[5%]">
          {item.image ? (
            <ItemImage imageId={item.image}
altText="picture" />
          ) : (
            <p></p>
          )}
        </div>
        <div className="bg-[#E6E6E6] rounded-lg items-
center justify-items-center w-full mr-[5%]">
          <Link key={item.itemId}
href={`${item.itemId}/update_item`} passHref>
```

```
                <UpdateItemButton itemId={item.itemId}/>
              </Link>
              <DeleteItemButton itemId={item.itemId}/>
              <div className="ml-[80%]">
                <FavourItem itemId={id}/>
              </div>
              <h1 className="text-2xl font-bold mb-4 text-
center mt-[1%]">{item.name}</h1>
              <Link href={`/users/${item.user}`}>
              <p className="flex-1 pl-1"><strong>Додано
користувачем:</strong> {item.user}</p>
              </Link>
              <div className="bg-white text-1xl flex flex-col
rounded-lg">
                <p className="pl-1"><strong>ID
матеріалу:</strong> {item.itemId}</p>
                <p className="pl-
1"><strong>Автор:</strong> {item.author}</p>
                <p className="pl-1"><strong>Опис:</strong>
{item.description}</p>
                <Link key={item.category}
href={`/catalog/category/${item.category}`}>
                <p className="pl-
1"><strong>Категорія:</strong>
{categoryMap[item.category] ?? item.category}</p>
                </Link>
                <p className="pl-1"><strong>Дата
публікації:</strong> {item.publishDate}</p>
                <p className="pl-1">
                  <strong>Посилання на
публікацію:</strong>{" "}
                  {item.materialLink.startsWith("/files/pdf") ? (
                    <a

href={`http://localhost:8080${item.materialLink}`}
                      download
                      className="underline text-blue-600"
                    >
                      Завантажити PDF
                    </a>
                  ) : (
                    <a
                      href={item.materialLink}
                      target="_blank"
                      rel="noopener noreferrer"
                      className="underline text-blue-600"
                    >
                      {item.materialLink}
                    </a>
                  )}
                </p>
                <div className="flex flex-1 pl-1 w-[50%]">
                  <p className="w-
[40%]"><strong>Рейтинг:</strong> {item.rating}</p>
                  <VoteItem
                    initialRating={item.rating} itemId={id}/>
                </div>
              </div>
            </div>
            {item.expertComment &&
item.expertComment.length > 0 ? (
              <div className="bg-yellow-200 row-start-2 w-
full h-full col-span-2 rounded-lg flex flex-col overflow-auto p-
2">
                {item.expertComment.map((comment, index)
=> (
                  <ExpertCommentElement
                    key={comment.expertCommentId ??
index}
                    itemId={item.itemId}
                    comment={comment}
                  />
                ))}
              </div>
            ) : (
              <p><strong>Експертних коментарів поки що не
додано</strong></p>
            )}
            <ExpertAddCommentButton itemId={item.itemId}/>
          </div>
          <div className="mt-8">
            <h2 className="text-xl font-semibold mb-
4">Коментарі</h2>
            <CommentForm itemId={item.itemId}/>
            {comments && comments.length > 0 ? (
              <ul className="flex flex-col-reverse">
                {comments.map((comment) => (
                  comment && comment.commentId ? (
                    <CommentElement
key={comment.commentId} itemId={item.itemId}
comment={comment}/>
                  ) : null
                ))}
              </ul>

            ) : (
              <p>Коментарів поки немає.</p>
            )}
          </div>
        </div>

    );
}
import React from "react";
import { getItemsByCategory } from "@/service/api";
import ItemCard from "@/components/items/ItemCard";

const categoryMap: Record<string, string> = {
  "history": "Історія",
  "philosophy": "Філософія",
  "art": "Мистецтво",
  "literature": "Література",
};

export default async function CategoryPage({ params }: {
params: { id: string } }) {
  const categoryId = params.id;

  const items = await getItemsByCategory(categoryId);

  if (!items || items.length === 0) {
    return (
      <div className="p-8 text-center">
        <h1 className="text-2xl font-semibold mb-4">
          Категорія: {categoryMap[categoryId] ??
categoryId}
        </h1>
        <p>Матеріали для цієї категорії не знайдено.</p>
      </div>
    );
  }

  const localizedCategory = categoryMap[items[0].category]
?? items[0].category;

  return (
    <div className="p-8">
      <h1 className="text-2xl font-bold mb-6">Категорія:
```

```
{localizedCategory}</h1>
        <div className="grid grid-cols-4 gap-4">
          {items.map((item) => (
            <ItemCard key={item.itemId} item={item} />
          ))}
        </div>
      </div>
    );
}
"use client"

import React, { useEffect, useState } from "react";
import { getCatalog } from "@/service/api";
import ItemCard from "@/components/items/ItemCard";

export default function Catalog() {
    const [items, setItems] = useState([]);

    useEffect(() => {
        const fetchData = async () => {
            try {
                const responseData = await getCatalog();
                setItems(responseData);
            } catch (error) {
                console.error('Error fetching data:', error);
            }
        };
        fetchData();
    }, []);

    return (
        <div className="w-full h-full grid grid-cols-4 grid-
flow-row gap-0">
            {items.length > 0 && items.map((item) => (
                <ItemCard key={item.itemId} item={item}/>
            ))}
        </div>
    );
}
import ProfilePage from "@/components/users/ProfilePage";

export default async function UserPage({params}) {
    return (
        <div>
            <ProfilePage params={params}/>
        </div>
    )
}
'use client';

import React, { useEffect, useState } from "react";
import { getCatalog } from "@/service/api";
import ItemCard from "@/components/items/ItemCard";

export default function Home() {
    const [items, setItems] = useState([]);
    const [token, setToken] = useState("");

    useEffect(() => {
        const fetchData = async () => {
            try {
                const responseData = await getCatalog();
                setItems(responseData);
            } catch (error) {
                console.error("Error fetching data:", error);
            }
        };

        const storedToken = localStorage.getItem("token");
        if (storedToken) {
            setToken(storedToken);
        }

        fetchData();
    }, []);

    return (
        <div className="min-h-screen font-[family-name:var(--
font-geist-sans)] p-4">
            <h1 className="text-2xl font-semibold mb-
4">Вітаємо!</h1>
            <h2 className="text-xl font-medium mb-
2">Найсвіжіші матеріали:</h2>
            <div className="grid grid-cols-4 gap-4 mb-6">
                {items
                    .sort((a, b) => new Date(b.creationDate) - new
Date(a.creationDate))
                    .slice(0, 4)
                    .map((item) => (
                        <ItemCard key={item.itemId} item={item} />
                    ))}
            </div>

            <h2 className="text-xl font-medium mb-
2">Матеріали з найвищими оцінками:</h2>
            <div className="grid grid-cols-4 gap-4 mb-10">
                {items
                    .sort((a, b) => b.rating - a.rating)
                    .slice(0, 4)
                    .map((item) => (
                        <ItemCard key={item.itemId} item={item} />
                    ))}
            </div>
        </div>
    );
}
```