

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка
до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ОБМІНУ КНИГ

Виконав: студент 4 курсу, групи КНТ-520
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

СОЛОМКА А.С.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
“ ” 2024 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СОЛОМКА Артур Сергійович
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи для обміну книг

керівник проєкту (роботи) К. Т. Н., доцент, СКРУПСЬКИЙ Степан Юрійович
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “01” квітня 2024 року №105

2. Строк подання студентом проєкту (роботи) 26 травня 2024 року

3. Вихідні дані до проєкту (роботи) наявний API для отримання списку країн та міст, швидкість відповіді якого не перевищує 10 мс

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Вибір технології реалізації системи;
2) Моделювання системи;
3) Програмна розробка системи;
4) Тестування системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
ER-модель бази даних;
Алгоритм підтвердження обміну;
Сайт системи для обміну книг (сторінки).

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С.Ю., к.т.н., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 01.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних технологій для реалізації системи	01.03.2024 р.	
2	Розробка структури системи	15.03.2024 р.	
3	Розробка алгоритму роботи системи	20.03.2024 р.	
4	Розробка серверної частини	01.04.2024 р.	
5	Розробка клієнтської частини	10.04.2024 р.	
6	Оформлення ПЗ	01.05.2024 р.	
7	Оформлення графічного матеріалу	10.05.2024 р.	
8	Оформлення допоміжного матеріалу	17.05.2024 р.	

Студент  Артур СОЛОМКА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)  Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 84 сторінки, 20 рисунка, 15 таблиць, 17 джерел

ANGULAR, JAVA, MYSQL, ORM HIBERNATE, REST, SPRING BOOT, TOMCAT, TYPESCRIPT

Об'єкт розробки – комп'ютерна системи для обміну книг.

Мета проекту – розробка розподіленої комп'ютерної системи яка спростить комунікацію між людьми та надасть зручні інструменти для проведення обмінів книгами.

В ході розробки було проаналізовано сучасні інструменти для реалізації системи, проведено моделювання додатку та спроектовано базу даних.

Серверна частина розроблена за допомогою мови програмування Java та програмної платформи Spring Boot приймає запити від клієнтів, обробляє їх, робить запити до бази даних та надсилає відповіді назад.

Клієнтська частина розроблена за допомогою мови програмування TypeScript та програмної платформи Angular являє собою користувацький інтерфейс для спрощеної комунікації з системою, яка надсилає запити до сервера та відображає його відповідь.

ЗМІСТ

Скорочення та умовні позначки	7
Вступ.....	8
1 Вибір технологій реалізації системи	9
1.1 Архітектура для розробки системи	9
1.2 Вибір системи керування базами даних	11
1.2.1 MySQL	11
1.2.2 PostgreSQL	12
1.2.3 Redis	14
1.2.4 MongoDB	15
1.3 Вибір мови програмування	17
1.3.1 Java	17
1.3.2 C#	19
1.3.3 Python.....	20
1.3.4 JavaScript	21
1.4 Вибір технології для проектування користувацького інтерфейсу	23
1.4.1 Angular.....	23
1.4.2 React	25
1.5 Виновник з вибору технологій.....	25
2 Моделювання системи	26
2.1 Постановка задачі розробки системи	26
2.1.1 Функції додатку	27
2.1.2 Характеристика користувача.....	27
2.1.3 Вимоги до інтерфейсу.....	28
2.1.4 Системні вимоги.....	29
2.1.5 Обмеження	29
2.2 Функціональні можливості користувачів	30
2.3 Проектування бази даних	32

2.4 Діаграми послідовностей.....	36
2.5 Визначення, акроніми, скорочення	39
3 Програмна розробка системи.....	41
3.1 Розробка серверної частини	41
3.1.1 Ініціалізація Spring Boot додатку.....	41
3.1.2 Базове налаштування Spring Boot додатку	43
3.1.3 Створення сутностей та їх зв'язків.....	47
3.1.4 Створення сервісних компонентів.....	53
3.1.5 Створення REST контролерів	59
3.1.6 Безпека додатку	61
3.2 Розробка клієнтської частини	65
3.2.1 Ініціалізація та налаштування Angular додатку	65
3.2.2 Створення клієнтських сервісів.....	65
3.2.3 Створення компонентів клієнтського інтерйесу.....	67
3.2.4 Налаштування маршрутизації між копонентами.....	69
4 Випробовування системи	71
4.1 Unit тестування	71
4.1.1 Тестування репозиторіїв.....	71
4.1.2 Тестування сервісів	73
4.2 Перевірка роботи системи.....	74
Висновки	79
Перелік джерел посилання	80
Додаток А Код JWT сервісу та фільтру	82

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API	– Application Programming Interface, Інтерфейс програмування додатків
CRUD	– Create read update delete, Створити зчитати оновити видалити
DTO	– Data Transfer Object, Об'єкт передачі даних
HTTP	– Hypertext Transfer Protocol, Протокол передачі гіпертексту
JPA	– Java Persistence API
JPQL	– Java Persistence Query Language, мова запитів Java Persistence
JSON	– JavaScript Object Notation, Нотований JavaScript об'єкт
JWT	– JSON Web Token, JSON веб токен
REST	– Representational State Transfer, Передача представницького стану
SQL	– Structured Query Language, Мова структурованих запитів
БД	– База даних
СКБД	– Система керування базами даних

ВСТУП

В сучасному світі зростає зацікавленість у читанні та обміні знаннями, але залишається проблема доступності книг для всіх бажаючих. Крім того, деякі старі книги, особливо рідкісні або маловідомі видання, не були відцифровані та недоступні в електронному форматі. Це створює додаткову потребу у фізичному обміні таких видань, які можуть бути цінними для колекціонерів та дослідників літератури. Інтернет-платформи для обміну книг стають дедалі популярнішими, оскільки вони дозволяють легко і швидко знайти потрібні твори, поділитися своїми книжками та отримати доступ до широкого асортименту літератури без значних витрат. Подібні системи не лише полегшують доступ до книжок, а й сприяють культурній взаємодії, розвитку читацької активності та збільшенню читацької аудиторії.

Дипломна робота має за мету розробку комп'ютерної системи, що сприятиме обміну книжковими творами між користувачами у віртуальному середовищі.

Для успішної реалізації системи буде здійснено вибір технологій, які оптимально відповідатимуть потребам проекту. Визначивши найбільш підходящу архітектуру для розробки системи, буде обрано систему керування базами даних, мову програмування та технології для проектування користувацького інтерфейсу.

Під час моделювання системи буде ретельно вивчено потреби та вимоги користувачів, визначено призначення системи та сформульовано задачу розробки. Важливим етапом буде визначення функціональних можливостей додатку, характеристик користувачів, вимог до інтерфейсу, системних вимог та обмежень.

Мета дипломної роботи полягає в створенні функціонального та зручного сервісу, що дозволить користувачам обмінюватися книгами, сприяючи тим самим поширенню культури читання та обміну знаннями. Після реалізації системи планується її тестування для забезпечення високої якості та ефективності роботи.

1 ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ СИСТЕМИ

Перед розробкою програмного забезпечення слід визначитися з її архітектурою, обрати систему керування базами даних, мову програмування та програмну платформу. Підбір відповідних технологій є ключовим етапом, оскільки він гарантує, що розроблювана система буде відповідати встановленим вимогам.

1.1 Архітектура для розробки системи

Функціонування мережі Інтернет засноване на архітектурі клієнт-сервер [1]. Клієнт є комп'ютер на стороні користувача, який надсилає запит до сервера з ціллю отримання певної інформації або виконання певних функцій сервера. Сервер виступає у ролі більш потужнішого комп'ютера або обладнання з програмним кодом та базою даних, чия логіка роботи забезпечує видачу файлів, контенту, ресурсів, які на ньому зберігаються за запитом клієнта. Клієнт-серверна архітектура зображена на рис. 1.1.

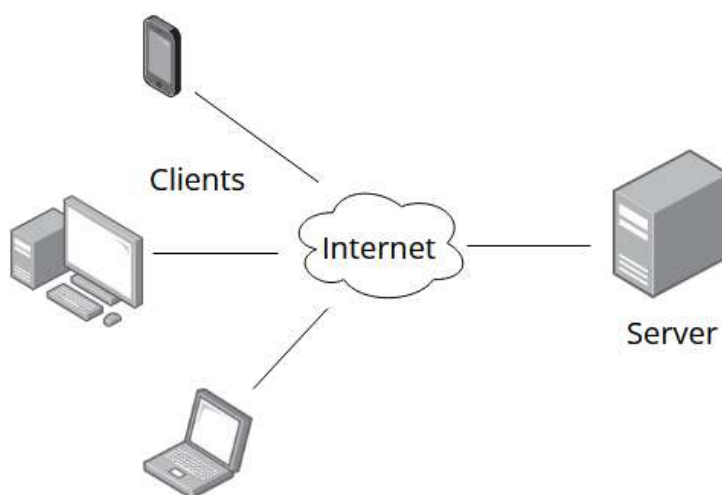


Рисунок 1.1 – Клієнт-серверна архітектура

Слід зазначити, що в основі взаємодії між клієнтом і сервером є те, що її ініціатором є клієнт. Запит клієнтів зазвичай представляє собою маніпуляцію з даними (їх додавання, отримання, оновлення, видалення), саме тому потрібно використовувати базу даних.

Взаємодія між клієнтом та сервером відбувається за допомогою мережеских протоколів, такими як HTTP, FTP, TCP, IP або UDP. Протокол зазвичай обирають під вид послуг, які надають сервери. Так як відбувається розробка веб-додатку, то буде використовуватися протокол HTTP та стек протоколів TCP/IP. Слід також додати про протокол HTTPS, який є розширенням для протоколу HTTP, що робить його безпечним. Справа в тому що HTTP передає дані у відкритому вигляді, HTTPS вирішує цю проблему об'єднуючись з протоколом TLS або SSL.

Сервер отримуватиме HTTP повідомлення, розпаковуватиме його, дістане потрібну інформацію та передасть програмі на обробку. Результат обробки упакується назад у HTTP відповідь, яку буде надіслано клієнтові. З цього випливає, що програма, яка розроблюється для сервера повинна мати наступні особливості:

- запуск здійснюється сервером;
- має забезпечити взаємодію з сервером;
- має забезпечити відправку результаті за запитом клієнта.

До серверних мов програмування, які можуть забезпечити ці особливості, можна віднести: Python, Java, C#, JavaScript та інші.

Клієнтська ж програма являє собою браузер.

Перевагою такого підходу є зниження витрат на супровід додатку розробниками, а також дозволить кожному клієнту постійно користуватися самою останньою версією програми, оскільки саме вона завжди буде знаходитися на сервері [1].

1.2 Вибір системи керування базами даних

При виборі системи керування базами даних (СКБД) для проектування бази даних (БД) слід враховувати різноманітні критерії. Зокрема, варто звернути увагу на функціональні можливості СКБД при рішенні поставленої задачі, об'єм баз даних, які можуть обробляти вибрана СКБД, наявність засобів проектування додатків, підтримку сучасних мов програмування, вимоги до апаратного та програмного забезпечення, а також фінансові витрати при проектуванні за допомогою обраної СКБД.

1.2.1 MySQL

MySQL є найпопулярнішою реляційною системою керування базами даних з відкритим вихідним кодом, яка надає прості інструменти для адміністрування та налаштування. Процес розробки з MySQL зосереджується на ефективній реалізації функцій, необхідних більшості людей. Це означає, що MySQL все ще має менший функціонал ніж його головні конкуренти [2]. Тим не менш СКБД має наступні переваги, які надають змогу залишатися конкурентоспроможною:

- може бути встановлена без будь-яких специфічних налаштувань;
- не є вимогливою до апаратного забезпечення та не навантажує системні ресурси, що дає можливість запустити СКБД на застарілому обладнанні;
- швидкість отримання інформації, що зробило фаворитом веб-адміністрування;
- увага до стандартів SQL;
- може просто стати частиною великої програмної системи. Найпопулярніші мови програмування надають бібліотеки для використання MySQL, до них можна віднести C, C++, C#, Java, Python, Perl, Ruby. Також СКБД підтримує стандарт Open Database Connectivity (ODBC), роблячи її доступною навіть якщо MySQL-специфічний функціонал не є доступним;

- має гнучку та безпечну систему привілеїв і паролів, яка забезпечує перевірку на основі хоста. Безпека паролів базується на шифруванні всього трафіку паролів під час підключення до сервера[3];

- є безкоштовною для використання.

До недоліків MySQL відносять:

- погана спроможність оброблення одночасних транзакцій, що призводить до зниження продуктивності в години пік. Для вирішення проблеми потрібна оптимізація запитів до бази даних та апаратних ресурсів. Ще одним способом є перехід до механізму зберігання, який підтримує блокування на рівні рядків, наприклад InnoDB;

- проблеми з продуктивністю великомасштабних баз даних. MySQL може мати проблеми з ефективним керуванням базами даних, розмір яких досягає терабайтів. Для пом'якшення проблеми з масштабованістю, слід розбити велику базу даних на менші, більш керовані сегменти, які будуть розподілені між кількома серверами;

- простота MySQL може стати проблемою, у разі необхідності розширеного функціоналу. Тільки інтеграція з іншим інструментами може цьому зарадити;

- обмежені функції безпеки у виданні для спільноти, такі як відсутність маскування даних та детальний аудит. Допоможе оновлення до корпоративної версії або провадження безпеки на рівні додатків і використання сторонніх інструментів для шифрування та аудиту.

Виходячи з переваг та недоліків MySQL виявиться оптимальним варіантом для середніх та невеликих систем. Простота використання, ефективність роботи в обмежених ресурсах, системні привілеїв та захист паролів робить СКБД ідеальним рішенням для проектів з обмеженими обсягами даних.

1.2.2 PostgreSQL

PostgreSQL являє собою реляційну об'єктно-орієнтовану систему керування базами даних з відкритим вихідним кодом, розроблена факультетом комп'ютерних наук Каліфорнійського університету в Берклі [4]. PostgreSQL виграв кілька нагород, у тому числі нагороду Linux Journal Editor's Choice Award

за найкращу базу даних тричі (за 2000, 2003 та 2004 рік) і нагороду Linux New Media Award за найкращу систему керування базами даних 2004 року [4].

PostgreSQL має наступні переваги:

- перша перевага випливає з того, що PostgreSQL є об'єктно-орієнтованою, а це значить що вона дозволяє визначати власні типи даних, перенавантажувати функції з різними типами даних аргументів, визначити відносини успадкування між таблицями;

- дотримання принципів ACID [5], що означає атомарність, послідовність, ізоляцію та стійкість. Вони надають ключові гарантії, що база даних буде підтримувати транзакції задля збереження цілісності даних та уникнення помилок дійсності даних;

- наявність функціоналу пов'язаний з безпекою, який можна використовувати безкоштовно. PostgreSQL має міжнародне визнання з точки зору своєї здатності забезпечувати безпеку [5];

- підтримка значної кількості мов програмування C, C++, C#, Java, Python, Perl, Ruby;

- відповідає стандартам SQL;

- є безкоштовною для використання.

В PostgreSQL наявні наступні недоліки:

- через багатий функціонал СКБД може бути складнішою у налаштування та оптимізації порівняно з іншими, особливо для користувачів, які тільки розпочали вчити адміністрування баз даних, що може призвести до уповільнення розробки програмного забезпечення;

- наявні проблеми продуктивності та відновленням резервних копій. Не рідко коли звичайний запит до бази даних може бути повільнішим за інші СКБД;

- PostgreSQL споживає більше системних ресурсів, таких як процесор і пам'ять, порівняно з іншими СКБД.

Попри ряд переваг, які визначені у PostgreSQL, необхідно врахувати його недоліки, які можуть суттєво вплинути на ефективність системи. Один із значущих аспектів – це складність у налаштування та оптимізації через багатий

функціонал СКБД, що може негативно вплинути на швидкість розробки програмного забезпечення. Додатково, виникають питання щодо продуктивності. Низька швидкість обробки запитів та велике споживання системних ресурсів, можуть вплинути на загальне враження від системи. В цьому випадку слід зробити висновок, що PostgreSQL може виявитися потужним рішенням для великих проектів з великим обсягом даних та розробниками, які володіють достатнім досвідом для її оптимізації та налаштування.

1.2.3 Redis

Redis – це дуже швидка не реляційна (або NoSQL) база даних, яка зберігає відображення ключів на п'ять різних типів значень [6]. СКБД має відкритий вихідний код. Redis підтримує постійне зберігання в пам'ять на диску, реплікацію для масштабування читання та розділення клієнтської сторони для масштабування запису. СКБД немає таблиць, і немає визначеного або зафіксованого базою даних способу пов'язання даних з іншими даними. Так само як memcached (система кешування типу «ключ-значення» різноманітних об'єктів в оперативній пам'яті), Redis може зберігати відображення ключів на значення і навіть досягти схожих рівнів продуктивності [6].

До значних переваг СКБД можна віднести:

- усі дані знаходяться в пам'яті, що дозволяє отримувати доступ до даних з низькою затримкою та високою пропускнуою здатністю. На відміну від традиційних баз даних, в яких дані зберігаються на диску, в сховищах даних в пам'яті немає потреби звертатися до диску, що зменшує затримку до мікросекунд, що призводить до виконання більшої кількості операцій й швидкої реакції;
- дозволяє використовувати просту структуру команд на відміну від мов запитів традиційних баз даних, що призводить до зменшення рядків коду. Наприклад, можна використовувати структуру даних хешу Redis для переміщення даних в сховище лише однією командою;
- використовує архітектуру з головним сервером та репліками і підтримує асинхронну реплікацію, де дані можуть бути репліковані на кілька серверів-

реплік. Це забезпечує швидке відновлення при відмові головного сервера, а також продуктивність, оскільки запити будуть розділені між серверами;

- підтримка значної кількості мов програмування C, C++, C#, Java, Python, Perl, Ruby;

- є безкоштовною для використання.

До недоліків Redis відносять:

- потужність обмежується наявною ОЗП. Якщо набір даних більший, ніж доступна ОЗП, це призводить до проблем з продуктивністю;

- транзакції, які включають кілька ключів, можуть не бути атомарними;

- не підтримує вторинні індекси, які є корисними для запитів до даних за атрибутами, відмінними від основного ключа, що призводить до того, що деякі типи запитів виявляються менш ефективним або навіть неможливими без додаткового моделювання даних;

- відсутні будь-які засоби контролю доступу, які повинні надаватися окремим рівнем авторизації. Також не підтримує жодного механізму шифрування, це повинно бути реалізовано за допомогою окремого рівня, такого як SSL-проксі.

Хоча Redis має безсумнівні переваги для використання в розробці, такі як висока швидкодія, простота використання та підтримка різних структур даних, існують обставини, коли вибір цього рішення може бути не найкращим. По-перше, Redis підходить для задач кешування і зберігання невеликих обсягів даних, і вартість його масштабування може бути значною. Збільшення обсягу даних може вимагати значних витрат на обладнання та інфраструктуру. По-друге, Redis не надає вбудованої підтримки для запитів, які вимагають складних операцій для веб-сайтів з вимогами до обробки та аналізу інформації.

1.2.4 MongoDB

MongoDB – це документ-орієнтована систему керування базами даних, яка використовує модуль даних на основі документів. Документ-орієнтована база даних заміняє концепцію «ряд» більш гнучкою моделлю – «документ» [7]. Дозволяючи вбудовані документи та масиви, підхід, орієнтований на документи,

робить можливим представлення складних ієрархічних відносин за допомогою одного запису.

Розглянемо основні переваги MongoDB:

- підтримує загальні вторинні індекси, що дозволяють виконувати різноманітні швидкі запити, а також надає можливості унікальної, складеної, геопросторової та повнотекстової індексації;
- підтримує «конвеєр агрегації», який дозволяє створити складні агрегації з простих елементів і дозволяє базі даних оптимізувати їх;
- підтримує колекції з часом життя для даних, які повинні застаріти через певний час, наприклад, сесії. Також підтримує колекції з фіксованим розмірів, які корисні для зберігання нещодавніх даних, таких як логи;
- підтримує простий у використанні протокол для зберігання великих файлів та метаданих файлів;
- відсутні заздалегідь визначеної схеми. Ключі та значення документа не мають фіксованих типів або розмірів. Без фіксованих схем додавання чи видалення за необхідності стає простішим.

Недоліки MongoDB:

- поєднання документів може бути дуже втомлюючим завданням, оскільки не підтримується з'єднання, як в реляційних баз даних;
- обмежує розмір документа лише 16 МБ. Також обмежена вкладеність документів, яка обмежена лише 100 рівнями в глибину;
- дублювання даних може призвести до пошкоджень, оскільки воно не відповідає стандартам ACID;
- вимагає великої кількості пам'яті через відсутність функціоналу з'єднань, що призводить до дублювання даних, що в свою чергу призводить до збільшення зайвої резервності даних і використання значного обсягу пам'яті.

MongoDB безсумнівно має значні переваги у вигляді гнучкості сховища даних та здатності обробки великих обсягів інформації, проте відсутність стандартизації у порівнянні з реляційними базами даних ускладнює взаємодію з даними та роботи з ними в умовах, коли потрібна точність та строга структура.

Для проектів, де необхідна висока продуктивність та складна обробка транзакцій, може виявитися, що реляційні бази даних є більш оптимальним варіантом. Також адаптація до MongoDB може вимагати додаткового часу для розробника.

1.3 Вибір мови програмування

Є велика кількість факторів, які потрібно враховувати при визначенні, яку мову програмування (або мови) використовувати при розробці системи. Навіть за умов, що досвідчені розробники вже мають власний інструментарій улюблених мов і технологій для різних проектів, вони все одно проходять через серію питань, щоб ухвалити остаточне рішення.

Кожна мова програмування має свої переваги та недоліки, і вибір правильної мови є критичним для успіху проекту. При виборі мови слід врахувати тип розроблюваного додатку, доступність бібліотек та фреймворків, безпека використання, продуктивність та швидкість розробки.

1.3.1 Java

Java – це високорівнева, мультиплатформна, об’єктно-орієнтована мова програмування зі строгою типізацією, вперше випущеною Sun Microsystems в 1995 році [8]. Надаючи надійну платформу, на основі якої будується багато послуг та додатків, Java стала основною мовою на яку покладаються нові інноваційні продукти та цифрові сервіси. Java широко використовувана мова програмування для написання веб-додатків й залишається популярним вибором серед розробників протягом двох десятиліть [8]. Популярність мови зумовлена наступними перевагами:

- хоч жодна мова програмування не має вбудованого захисту від усіх видів вразливостей, Java пропонує високий рівень захисту, запобігаючи конкретним недолікам безпеки, які можна знайти в деяких інших мовах. Наприклад,

відсутність вказівників може запобігти несанкціонованому доступу, долаючи проблему, яка часто виникає при програмуванні на мові C;

- використовує автоматичне управління пам'яттю, що спрощує роботу програміста та зменшує ймовірність виникнення помилок, пов'язаних з витоком пам'яті або її неправильним використанням;

- підтримує роботу з потоками, що робить можливим одночасне виконання різних частин програми та підвищує продуктивність;

- існує широка спільнота розробників та велика кількість бібліотек та фреймворків, що полегшує розробку програм.

Spring є найпопулярнішим фреймворком з відкритим вихідним кодом для розробки веб-додатків за допомогою Java [9]. Spring Framework визначається своєю універсальністю та розширюваністю, пропонуючи широкі можливості для розробки Java додатків. Його переваги включають в себе гнучкість конфігурацій, вбудовану підтримку транзакцій та безпеку (зокрема, завдяки Spring Security), а також ефективний контейнер управління об'єктів названих бінами [9]. З іншого боку, це включає в себе додатковий шар абстракції, що негативно впливає на продуктивність, складність конфігурації. За останні роки інструменти, такі як Spring Boot, значно спростили процес розгортання та конфігурування, забезпечуючи швидкий старт проектів.

Перейдемо до недоліків Java:

- потребує інтерпретації під час виконання, що дозволяє йому працювати на будь-якій операційній системі, але це також робить його менш продуктивним у порівнянні з мовами, такими як C та C++;

- програма на Java витрачає більше пам'яті, оскільки вона виконується на JVM.

Розробка веб-сайту за допомогою Java має свої вагомі переваги, які роблять цей вибір привабливим для багатьох розробників. Java безсумнівно є мовою програмування з високим рівнем надійності і стійкості, що робить її ідеальним вибором для веб-розробки, а такі інструменти як Spring Boot, дозволяють значно спрости та пришвидшити процес розробки, що нівелює всі недоліки Java.

1.3.2 C#

C# - це об'єктно-орієнтована, компонентно-орієнтована мова програмування зі строгою типізацією розроблена компанією Microsoft. C# надає мовні конструкції для прямої підтримки цих концепцій, роблячи C# природною мовою для створення та використання програмних компонентів [10]. З часу свого виникнення C# додав можливості для підтримки нового функціоналу та виникаючих практик проектування програмного забезпечення. У своєму ядрі C# - це об'єктно-орієнтована мова, в якій розробник визначає типи та їхню поведінку [10]. Популярність C# продовжує зростати завдяки наступним перевагам:

- інтегрована з платформою .NET, що сприяє високій міжмовній інтеграції, багатофункціональності та ефективності виконання програм;
- підтримує асинхронне програмування та можливості багатозадачності, що полегшує роботу з асинхронними операціями;
- гармонійно взаємодіє з іншими технологіями Microsoft, такими як SQL Server, Azure Cloud Services, Excel, та багато інших;
- подібно Java використовує автоматичне управління пам'яттю;
- надає вбудовані засоби безпеки, які допомагають захищати додаток від атак та вразливостей;
- існує широка спільнота розробників та велика кількість бібліотек та фреймворків.

ASP.NET Core – це найпопулярніший фреймворк для розробки веб-додатків на мові програмування C# [11]. Він вирізняється високою продуктивністю, швидкістю роботи та підтримкою хмарних технологій. Фреймворк є відкритим та активно розвивається спільнотою, що дозволяє швидко отримувати оновлення та виправлення помилок. Однак вартість навчання для незнайомих з технологією розробників, може стати значним недоліком, оскільки фреймворк вимагає розуміння власних концепцій. Також, для деяких старих проєктів, перехід на ASP.NET Core може бути складним через несумісність деяких API з попередніми версіями.

До недоліків C# відносять:

- у порівнянні з мовами, які компілюються безпосередньо в машинний код, C# може бути менш ефективним з точки зору швидкодії;
- в окремих випадках існують спроби атак, такі як Deniel of Service, або використання певних конструкцій мови, які може призвести до вразливостей безпеки;
- додатки, написані на C#, можуть мати великі розміри файлів через включення виконавчого файлу та інших необхідних компонентів .NET;
- розробка додатку для інших платформ, відмінних до Windows, може бути важчим і залежить від того, яку підтримку надає .NET Core або інші інструменти.

Хоча C# і ASP.NET Core є ефективними інструментами для розробки веб-додатку, існують певні недоліки, які роблять їх використання не найкращим вибором. Залежність від Microsoft може значно ускладнити масштабування проекту. Також слід враховувати, що веб-розробка швидко змінюється, і інші популярні інструменти, такі як Java з Spring Boot, надають набагато більший функціонал і можливості.

1.3.3 Python

Python – це інтерпретована, об'єктно-орієнтована високорівнева мова програмування з динамічною семантикою [12]. Основними напрямками розробки є веб та машинне навчання.

Python має високорівневі вбудовані структури даних, спільно з динамічною типізацією та динамічним зв'язуванням, що робить мову привабливою для швидкої розробки додатків, а також для використання як мови сценарії або мови для з'єднання існуючих компонентів між собою. Має спрощений синтаксис порівняно з C-подібними мовами програмування. Інтерпритатор та обширна стандартна бібліотека доступні у вигляді вихідних або бінарних файлів безкоштовно і може бути вільно розповсюджені [12]. Ще однією перевагою мови є її активна спільнота та велика кількість фреймворків.

Найпопулярніший фреймворк для розробки веб-застосунків на Python є Django. Фреймворк має високий рівень абстракції, що спрощує роботу з базами даних та шаблонами, а також вбудовану аутентифікацію та авторизацію, що

робить розробку безпеки додаку більш простою [13]. Однак є доволі обмеженим в гнучкості для деяких проектів, великий обсяг коду, що може бути надмірним для малих проектів, і має великий час вивчення для новачків, оскільки Django накладає власні концепції та структури на розробку.

Основні недоліки Python:

- порівняно з деякими іншими мовами програмування, може бути менш ефективним з точки зору швидкодії;
- програми на Python використовують більше оперативної пам'яті;
- недорозвинений та примітивний доступ до даних порівняно з такими технологіями як JDBC;

Слід також зазначити, що спрощений синтаксис може також стати недоліком, особливо для розробників, які звикли до С-подібних мов.

Python є популярною мовою програмування з чисельними перевагами, проте варто враховувати певні обмеження, які можуть зробити його менш практичним для розробки веб-систем у деяких випадках. Швидкодія та продуктивність Python можуть виявитися недостатніми порівняно з іншими мовами, такими як Java або C#. Ці недоліки роблять мову менш вигідним вибором для розробки.

1.3.4 JavaScript

JavaScript – це легка інтерпретована мова програмування з функціями першого класу (функції мови обробляється так само, як і будь-яка інша змінна). Динамічні можливості JavaScript включають конструювання об'єктів під час виконання, списки змінних параметрів, змінні функції, динамічне створення сценаріїв, інтроспекцію об'єктів, а також відновлення вихідного коду.

Основи JavaScript на серверній стороні були закладені у 2008 році, коли Google випустив V8, JavaScript-двигун з відкритим вихідним кодом, який став частиною веб-браузера Google Chrome [14]. Рік потому Раян Даль випустив Node.js, в який він додав цикл подій та низькорівневі JavaScript API для файлової системи та драйвери для апаратного забезпечення [14]. І завдяки V8, він може працювати на найважливіших обчислювальних платформах світу, від серверів до планшетів та смартфонів.

Node.js став популярним за короткий проміжок часу завдяки наступним перевагам:

- для великих проектів Node.js може бути легко масштабованим, завдяки можливості створення кластерів;
- прискорює процесу розробки та зменшує витрати і зусилля, необхідні для створення проекту, оскільки JavaScript стане єдиною мовою для клієнта та сервера;
- відмінна продуктивність та здатність ефективно обробляти великі обсяги даних одночасно, завдяки V8, який допомагає компілювати код безпосередньо в машинний код;
- має вбудований модуль кешування, який допомагає зменшити загальний час завантаження;
- має активну та велику спільноту розробників.

Однак Node.js має складну асинхронну модель програмування, що відлякує розробників, які віддають перевагу лінійному блокуючому введенню та виведенню. Відсутність бібліотечної підтримки у порівнянні з іншими мовами програмування призводить до необхідності користуватися загальними бібліотеками для різних завдань. Нестабільні зміни у API Node.js із виходом нових версій створюють проблеми сумісності. При обробці важких обчислювальних завдань Node.js може втрачати продуктивність через використання всієї потужності процесора. Навіть з доданням модуля «worker threads» у 2018 році для запуску кількох потоків одночасно, розробникам все ще потрібно шукати інші інструменти для вирішення проблем, пов'язаних із великими обчислювальними завданнями.

1.4 Вибір технології для проектування користувацького інтерфейсу

Існує два шаблони проектування веб-додатків: односторінковий та багатосторінковий.

Односторінковий (Single Page Application, SPA) – реалізація веб-застосунку, який завантажує лише один документ, а потім, за потреби, оновлює вміст цього документа за допомогою JavaScript API, що дозволяє не перезавантажувати сторінку браузера при зміні контенту. SPA кешує будь-які локальні дані, що дозволяє працювати навіть в автономному режимі. Є швидким, оскільки більшість ресурсів завантажується лише один раз протягом усього життєвого циклу застосунку.

Багатосторінковий (Multi-Page Application, MPA) – реалізація веб-застосунку, який має більше ніж один документ зі статичною інформацією та посилання на інші документи. Кожна зміна вимагає від сервера відображення нової сторінки в браузері, що призводить до перезавантаження сторінки. Це ідеальний підхід для користувачів, які потребують карти для навігації в застосунку з великою кількістю інформації. При такому проектуванні SEO оптимізація є простішою.

Розроблювана система буде використовувати шаблон SPA, для реалізації якого найчастіше використовують такий інструмент як React або Angular.

1.4.1 Angular

Angular – це фреймворк для дизайну та розробки додатків, а також платформа для створення складних односторінкових додатків [15]. Мовою розробки є TypeScript, яка компілюється в JavaScript.

Архітектура Angular-додатка ґрунтується на кількох основних концепціях. Основні будівельні блоки фреймворку – це Angular-компоненти [15].

Компоненти визначають представлення (views), які представляють собою набори елементів екрану, які Angular може вибрати та модифікувати відповідно логіки [16].

Компоненти та служби (services), які забезпечують функціональність фонових операцій, не пов'язаних безпосередньо з представленням, наприклад, отримання даних [16]. Такі служби можна впроваджувати компоненти як залежності, що робить код модульним, перевикористовуваним та ефективним.

Компоненти та служби – це класи, позначені декораторами (decorators). Ці декоратори надають метадані, які вкажуть Angular, як їх використовувати [16].

Зазвичай компоненти додатки визначають багато представлень, які організовані в ієрархічному порядку. Angular надає сервіс Router для допомоги у визначенні шляхів навігації між представленнями. Router забезпечує високорівнені можливості навігації прямо в браузері.

До переваг Angular можна віднести:

- підтримує двустороннє зв'язування даних, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних в моделі та навпаки;
- дозволяє розбити додаток на модулі, що сприяє кращому організації коду і сприяє відокремленню функціональних частин додатку;
- надає багато вбудованих інструментів та функціоналу для розробки маршрутизації, валідації форм та інші;
- має велику спільноту розробників та відмінну документацію.

Недоліки Angular:

- додатки мають великий розмір;
- не так сильно абстрагується від DOM (Document object model) порівняно з React.

Великий функціонал та незначні недоліки роблять Angular кращим варіантом для розробку клієнтського інтерфейсу.

1.4.2 React

React – це JavaScript-бібліотека для створення інтерфейсів користувача. React використовує віртуальний DOM, який є внутрішній представленням реального DOM [17]. Цей віртуальний DOM дозволяє React ефективно оновлювати та відтворювати лише ті компоненти, які зазнали змін, замість повного відтворення всієї сторінки. Крім того технологія сприяє концепції

повторного використання компонентів, які є автономними модулями, які можна використовувати в різних частинах додатку [17]. Цей модульний підхід полегшує побудову складних інтерфейсів користувача, розбиваючи їх на менші, повторно використововувані компоненти.

Однак React не є повноцінним фреймворком. Розробники повинні поєднувати його з бібліотекою такою як Redux або React Router, щоб отримувати функціональність, схожу на Angular. React не є реактивним (в контексті реактивного програмування та архітектури, які є більш подійно-орієнтованими, стійкими та відзивчивими) «з коробки» [17]. Розробники повинні використовувати інші інструменти, такі як Reactive Extensions, для композиції асинхронних потоків даних із спостережуваними.

Окрім того технологія є все ще новою, основний API змінюються, а найкращі практики використання все ще розвиваються, що не робить React привабливим для використання в розробці.

1.5 Висновки з вибору технологій

На даному етапі було проаналізовано основні переваги та недоліки сучасних технологій для розробки систем в результаті чого, було визначено, які інструменти будуть використовуватися для реалізації комп'ютерної системи та її архітектуру.

Архітектурою системи було обрано клієнт-серверну. Перевагою такого підходу стане спрощення розробки та підтримання додатку.

Через простоту використання та невимогливість до апаратних ресурсів було обрано систему керування базами даних MySQL.

Завдяки своїй надійності та стійкості Java, було обрано як серверну мову програмування з фреймворком Spring Boot, що значно пришвидшить процес розробки та розгортання веб-додатку.

Через значну кількість вбудованих інструментів та функціоналу для створення користувацького інтерфейсу було обрано фреймворк Angular. Таким чином мовою програмування клієнтської частини стане TypeScript.

2 МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Постановка задачі розробки системи

BookSwar – це онлайн-платформа для обміну книг, яка підтримує функціонал, як для внутрішніх сторін (адміністраторів, модераторів) так і для зовнішніх осіб (користувачів).

Платформа має бути загальною доступною, без надання дозволу на користування та повинна однаково працювати на таких популярних браузерях як: Chrome, Safari, Firefox, Opera. Час відповіді на запити користувачів не повинно перевищувати 5 секунд. Для користування застосунком, користувач повинен мати персональний комп'ютер або смартфон з доступом до мережі Інтернет, обладнаний веб-браузером.

Платформа надасть змогу спростити комунікацію між читачами спільноти, розповсюдити літературу, підвищить надійність обмінів. Для цього під час розробки застосунку, слід вирішити наступні задачі:

- надати можливості для взаємодії з іншими користувачами спільноти, такі як написання повідомлення, запропонування обміну;
- надати прості інструменти керування списком власних книг з детальною інформацією та можливістю їх відслідковування;
- надати можливість писати рецензії та залишати оцінки книгам;
- реалізувати систему рейтингу, за якою користувачі будуть вирішувати чи будуть проводити обмін з певним користувачем;
- забезпечити зручний та гнучкий пошук книг.

2.1.1 Функції додатку

Онлайн-платформа «BookSwar» має виконувати наступний перелік функцій:

- можливість додати свою книгу. Користувачі зможуть додати свої книги, використовуючи зручну форму на платформі. Форма повинна містити необхідні поля для введення інформації про книгу, таку як назва, автор, жанр, її стан;
- надати можливість відслідковувати книгу за унікальним ідентифікатором. Кожна книга повинна мати унікальний ідентифікатор, що полегшить контроль за місцезнаходженням книги;
- надати можливість користувачам підтверджувати обмін. Функція, яка підтвердить що обмін між користувачами відбувся, а власник книги змінився;
- надати користувачам можливість оцінити книгу та залишити рецензію;
- надати користувачам інструменти для зручного пошуку книг. Пошук книги повинен бути можливий за її назвою, автором, жанром, мовою написання, її стану;
- видача балів за підтверджений обмін книг, що надасть користувачу змогу не тільки відправляти, але й отримувати книги;
- дати можливість написати повідомлення іншому користувачу;
- забезпечити повний контроль внутрішнім контентом адміністраторам. Для адміністраторів повинні бути доступні інструменти для моніторингу та управління контентом та профілями користувачів на платформі;
- надати інструментарій модераторам для втручання до змісту залишених коментарів або іншого контенту певних користувачів.

2.1.2 Характеристика користувача

Типовий користувач BookSwar – це літературний ентузіаст, який цінує можливість збагачення свого читацького досвіду, вивчення нових творів, має доступ до Інтернету та має веб-браузер.

Передбачається, що користувач достатньо знайомий зі смартфоном або комп'ютером, щоб працювати з браузером, клавіатурою та мишею, і здатен переглядати прості веб-сайти.

Для визначення того, як програмне забезпечення буде взаємодіяти з різними групами користувачів та врахувати їх потреби, слід встановити ролі користувачів на сайті (табл. 2.1).

Таблиця 2.1 – Ролі користувачів онлайн платформи

Роль	Опис ролі
Гість	Користувач, який не є частиною спільноти, або ще не автентифікований.
Читач	Користувач, який вже зареєстрований на платформі та вже є авторизований. Хочє додати нову книгу або обмінюватися книгами з іншими користувачами.
Модератор	Користувач, який вже зареєстрований на платформі та вже є авторизований. Має можливості для редакції або видалення створеного контенту користувачів (рецензії, додані книги).
Адміністратор	Користувач, який вже зареєстрований на платформі та вже є авторизований. Має повний контроль над обліковими записами користувачів та внутрішнім контентом платформи.

2.1.3 Вимоги до інтерфейсу

Для забезпечення максимального комфорту користувачів при використанні платформи, інтерфейс повинен відповідати низці ключових вимог:

- бути адаптивним для різних пристроїв та розмірів екрану;
- бути сумісним з всіма популярними веб-браузерами Chrome, Safari, Firefox, Opera;

- забезпечити інтуїтивність інтерфейсу, уникнути перевантаженості зайвими елементами та забезпечити зручне користування основними функціями;
- надати зручні форми для взаємодії з системою;
- мати гнучкий інструмент для пошуку, який надасть змогу здійснювати швидкий та точний пошук з можливістю фільтрації;
- надати зручний інструментарій для модераторів та адміністраторів для виконання їх обов'язків.

Кастомізація інтерфейсу користувачами не передбачається.

2.1.4 Системні вимоги

Рекомендовані системні вимоги для серверу:

- 4 ядерний процесор з частотою 2 ГГц;
- оперативна пам'ять на 8 ГБ;
- HDD на 200 ГБ доступного місця;
- швидкість Інтернет з'єднання 100 Мбіт/с.

Вимоги для серверу можуть змінюватися під впливом розширення функціональності, збільшення обсягу даних або зростання числа користувачів.

Рекомендовані системні вимоги для клієнта:

- операційна система Windows, macOS або Linux;
- 2 ядерний процесор з частотою 1.6 ГГц;
- оперативна пам'ять на 512 МБ;
- веб-браузер Crome, Chrome, Safari, Firefox, Opera;
- швидкість Інтернет з'єднання 50 Мбіт/с.

2.1.5 Обмеження

Користувачі сайту можуть мати тільки одну роль. При реєстрації користувачеві автоматично видається роль «Читач», для зміни ролі потрібне втручання адміністрації.

Сторінка з інструментарієм адміністраторів повинна бути приватною, тобто будь-який користувач який не має ролі «Адміністратор» не повинен мати можливість потрапити на дану сторінку.

Користувачі сайту з роллю «Гість» має право перебувати на всіх публічних або не персоналізованих сторінках (наприклад, список бажаного), але не має можливості стати учасником обміну, додати власну книгу на сайт, залишити рецензію, написати повідомлення іншому користувачу.

2.2 Функціональні можливості користувачів

Виходячи з постановки задачі можна зробити висновок, що в системі наявні наступні ролі, які мають відповідний набір функціональних можливостей та повноважень: «Гість», «Читач», «Модератор», «Адміністратор».

«Гість» матиме найменший набір функціоналу, який наведений на рисунку 2.1.



Рисунок 2.1 – Функціонал користувача з роллю «Гість»

Користувач з роллю «Читач» в свою чергу вже матиме увесь необхідний функціонал для повноцінного користування сайтом, що надасть змогу стати учасником обміну.

Функціонал користувача з роллю «Читач» наведений на рисунку 2.2.

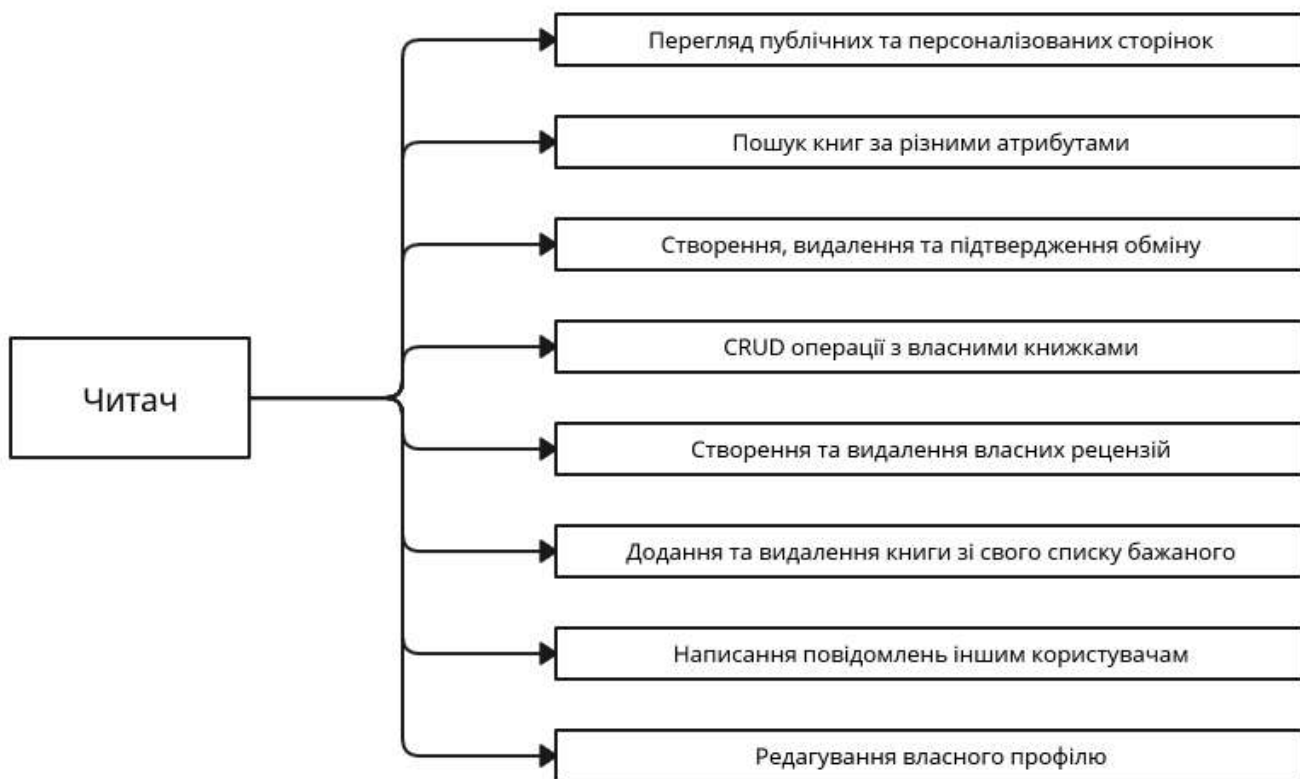


Рисунок 2.2 – Функціонал користувача з роллю «Читач»

Роль «Модератор» надасть весь функціонал ролі «Читач», а також додаткові повноваження, що зображено на рисунку 2.3.

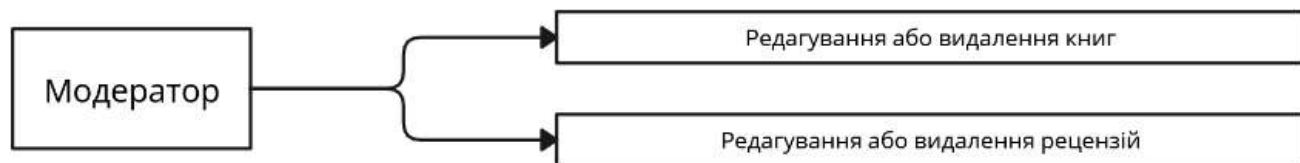


Рисунок 2.3 – Повноваження користувача з роллю «Модератор»

Роль «Адміністратор» вміщує в собі найбільший функціонал, в який в ходять функціонал ролі «Читач», повноваження ролі «Модератор», а також власні, які нададуть повний контроль над вмістом сайту.

Повноваження та функціонал ролі «Адміністратор» наведено на рисунку 2.4.

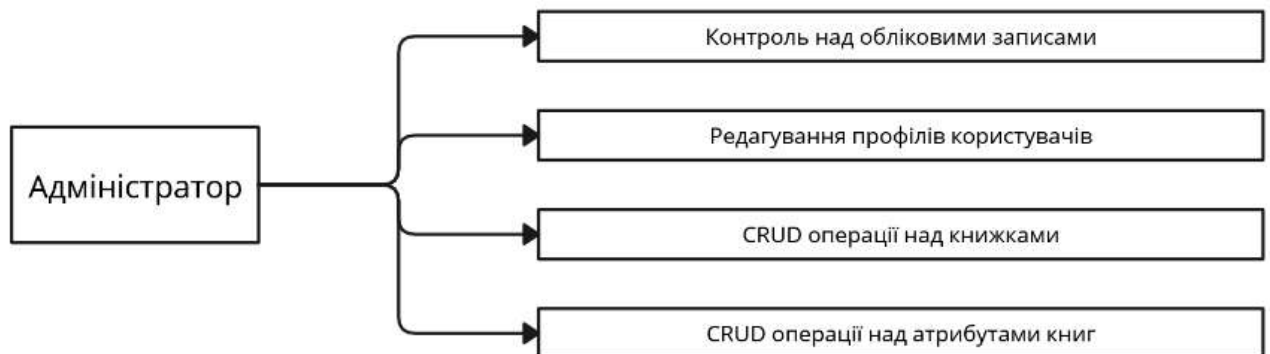


Рисунок 2.4 – Повноваження та функціонал ролі «Адміністратор»

2.3 Проектування бази даних

Проектування бази даних є важливим передумовою перед розробкою будь-якої інформаційної системи. Це ключовий етап, який визначає структуру, організацію та зв'язки між даними, що будуть використовуватися в системі. Важливо провести цей процес перед початком розробки, оскільки правильно спроектована база даних дозволяє забезпечити оптимальну продуктивність, швидкий доступ до інформації та забезпечити надійність та безпеку даних.

При розробці концептуальної моделі бази даних комп'ютерної системи необхідно, в першу чергу, визначити сутності. Кожна сутність представляється назвою, переліком атрибутів і первинним ключем. Визначається зв'язки між таблицями, а також обмеження цілісності даних. Спираючись на попередній аналіз, можна виділити наступні сутності: «Книги» (табл. 2.2), «Обміни» (табл. 2.3), «Жанри» (табл. 2.4), «Мови» (табл. 2.5), «Якість» (табл. 2.6), «Статус» (табл. 2.7), «Рецензії» (табл. 2.8), «Токени» (табл. 2.9), «Нотатки» (табл. 2.10), «Списки_бажаного» (табл. 2.11), «Користувачі» (табл. 2.12), «Чати» (табл. 2.13), «Повідомлення» (табл. 2.14), «Книги_Жанри» (табл. 2.15).

Таблиця 2.2 – Сутність «Книги»

Атрибут	Тип даних	Примітка
book_id	BIGINT	Первинний ключ
owner_id	BIGINT	
quality_id	BIGINT	
language_id	BIGINT	
status_id	BIGINT	
title	VARCHAR(50)	
author	VARCHAR(50)	
description	VARCHAR(1000)	
photo	LOB	

Таблиця 2.3 – Сутність «Обміни»

Атрибут	Тип даних	Примітка
exchange_id	BIGINT	Первинний ключ
book_id	BIGINT	
initiator_id	BIGINT	
recipient_id	BIGINT	
confirmed	BOOLEAN	

Таблиця 2.4 – Сутність «Жанри»

Атрибут	Тип даних	Примітка
genre_id	BIGINT	Первинний ключ
genre	VARCHAR(255)	

Таблиця 2.5 – Сутність «Мови»

Атрибут	Тип даних	Примітка
language_id	BIGINT	Первинний ключ
language	VARCHAR(255)	

Таблиця 2.6 – Сутність «Якість»

Атрибут	Тип даних	Примітка
quality_id	BIGINT	Первинний ключ
quality	VARCHAR(255)	

Таблиця 2.7 – Сутність «Статус»

Атрибут	Тип даних	Примітка
status_id	BIGINT	Первинний ключ
status	VARCHAR(255)	

Таблиця 2.8 – Сутність «Рецензії»

Атрибут	Тип даних	Примітка
book_id	BIGINT	Складний ключ
user_id	BIGINT	Складний ключ
rating	INT	
review	VARCHAR(1000)	

Таблиця 2.9 – Сутність «Токени»

Атрибут	Тип даних	Примітка
token_id	BIGINT	Первинний ключ
user_id	BIGINT	
token	VARCHAR(255)	
expired	BOOLEAN	
revoked	BOOLEAN	

Таблиця 2.10 – Сутність «Нотатки»

Атрибут	Тип даних	Примітка
note_id	BIGINT	Первинний ключ
book_id	BIGINT	
country	VARCHAR(255)	
city	VARCHAR(255)	
date	DATE	

Таблиця 2.11 – Сутність «Списки_бажаного»

Атрибут	Тип даних	Примітка
book_id	BIGINT	Складний ключ
user_id	BIGINT	Складний ключ

Таблиця 2.12 – Сутність «Користувачі»

Атрибут	Тип даних	Примітка
user_id	BIGINT	Первинний ключ
email	VARCHAR(255)	
nickname	VARCHAR(35)	
password	VARCHAR(255)	
country	VARCHAR(255)	
city	VARCHAR(255)	
points	INT	
registration_date	DATE	
role	ENUM('READER', 'MODERATOR', 'ADMINISTRATOR')	
photo	LONGBLOB	
activity	BOOLEAN	

Таблиця 2.13 – Сутність «Чати»

Атрибут	Тип даних	Примітка
chat_room_id	BIGINT	Первинний ключ
first_participant_id	BIGINT	
second_participant_id	BIGINT	

Таблиця 2.14 – Сутність «Повідомлення»

Атрибут	Тип даних	Примітка
message_id	BIGINT	Первинний ключ
chat_room_id	BIGINT	
sender_id	BIGINT	
content	VARCHAR(500)	
timestamp	DATETIME(6)	

Таблиця 2.15 – Сутність «Книги_Жанри»

Атрибут	Тип даних	Примітка
book_id	BIGINT	Складний ключ
genre_id	BIGINT	Складний ключ

Таким чином всього база даних налічує у собі 14 сутностей.

Описавши сутності бази стає можливим створення моделі «сутність-зв'язок» (ER-модель) – модель даних, яка вказує на те, як різні сутності зв'язані один між одним в системі. На рисунку 2.5 зображено створену ER-модель.

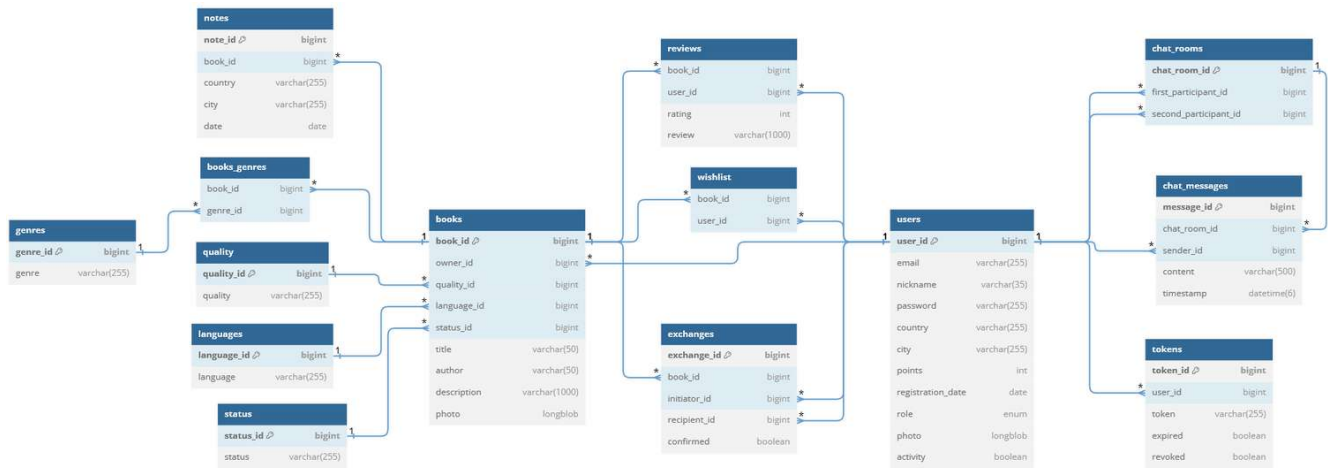


Рисунок 2.5 – ER-модель бази даних

2.4 Діаграми послідовностей

Діаграми послідовностей — це діаграми взаємодії, які детально описують, як виконуються операції. Вони фіксують взаємодію між об'єктами в контексті співпраці.

В нашому випадку об'єктами є клієнт, Tomcat (оскільки Spring Boot включає попередньо налаштований вбудований веб-сервер за замовчуванням), SMTP сервер (якщо знадобиться відправити повідомлення на пошту користувачів), Spring Boot, MySQL.

Діаграми послідовності зосереджені на часі, і вони візуально показують порядок взаємодії за допомогою вертикальної осі діаграми, щоб відобразити час, які повідомлення надсилаються та коли.

На рисунку 2.6 зображено діаграму послідовностей реєстрації користувача.

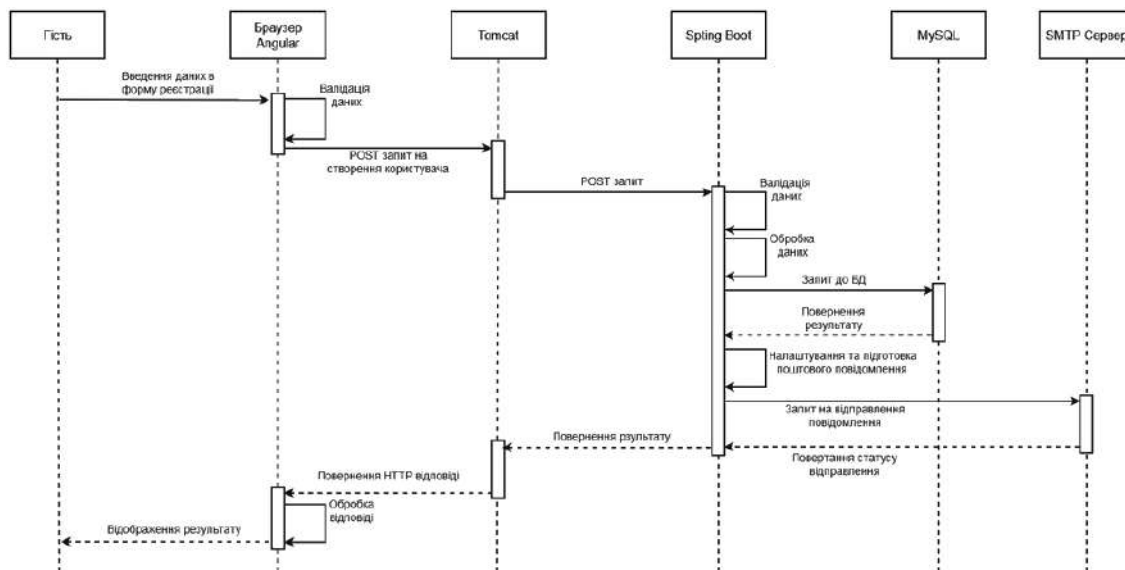


Рисунок 2.6 – Діаграма послідовностей прецеденту «Реєстрація»

При реєстрації користувач спочатку заповнює усі необхідні поля форми такі як: псевдонім, пошта та пароль. Після валідації даних браузер здійснить запит на збереження даних на сервер, який збереже дані в БД та за допомогою протоколу SMTP відправить повідомлення на вказану пошту клієнта для підтвердження реєстрації. Користувачу залишиться тільки відкрити отриманий лист на пошті та перейти по посиланню впродовж 24 годин, після чого його акаунт стане активним. Діаграма послідовностей активації акаунту зображено на рисунку 2.7.

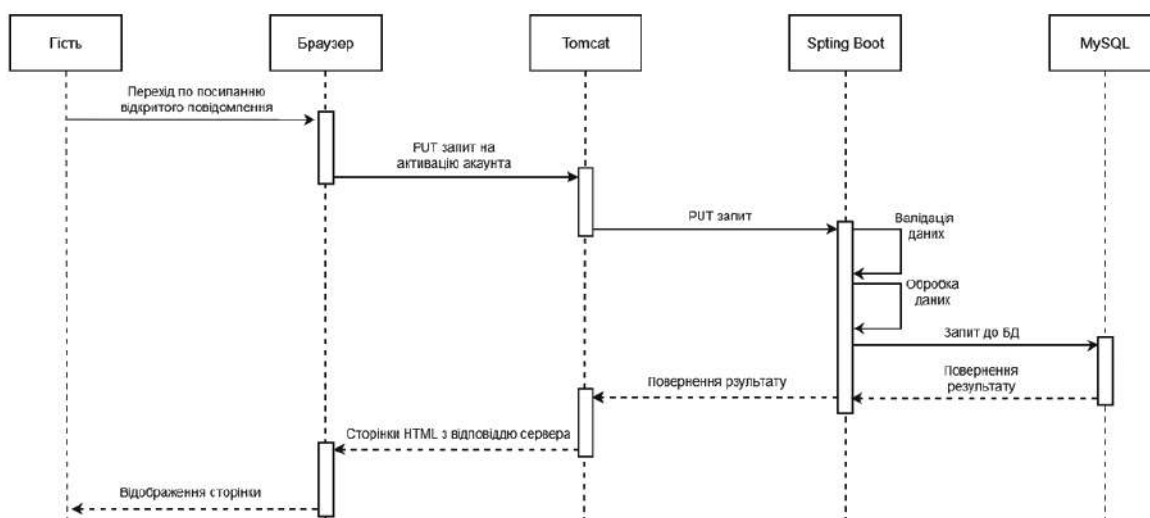


Рисунок 2.7 - Діаграма послідовностей прецеденту «Активация акаунта»

Для створення обміну користувачу слід відкрити сторінку зі змістом, який відобразить інформацію стосовно певної книги. Для цього користувач матиме можливість пошуку книги за різними критеріями або без них. Якщо користувач побачив під час пошуку потрібну йому книгу він матиме можливість перейти по посиланню, яке буде містити id книги, після чого й отримає усю необхідну інформацію. Далі, якщо книга має статус який підтверджує, що вона готова до обміну, користувач зможе натиснути кнопку “Exchange” і надіслати запит на створення нового обміну на сервер.

Діаграма послідовності створення нового обміну зображено на рисунку 2.8, для того, щоб на перенавантажувати діаграму, будемо вважати, що користувач вже знайшов необхідну книгу.

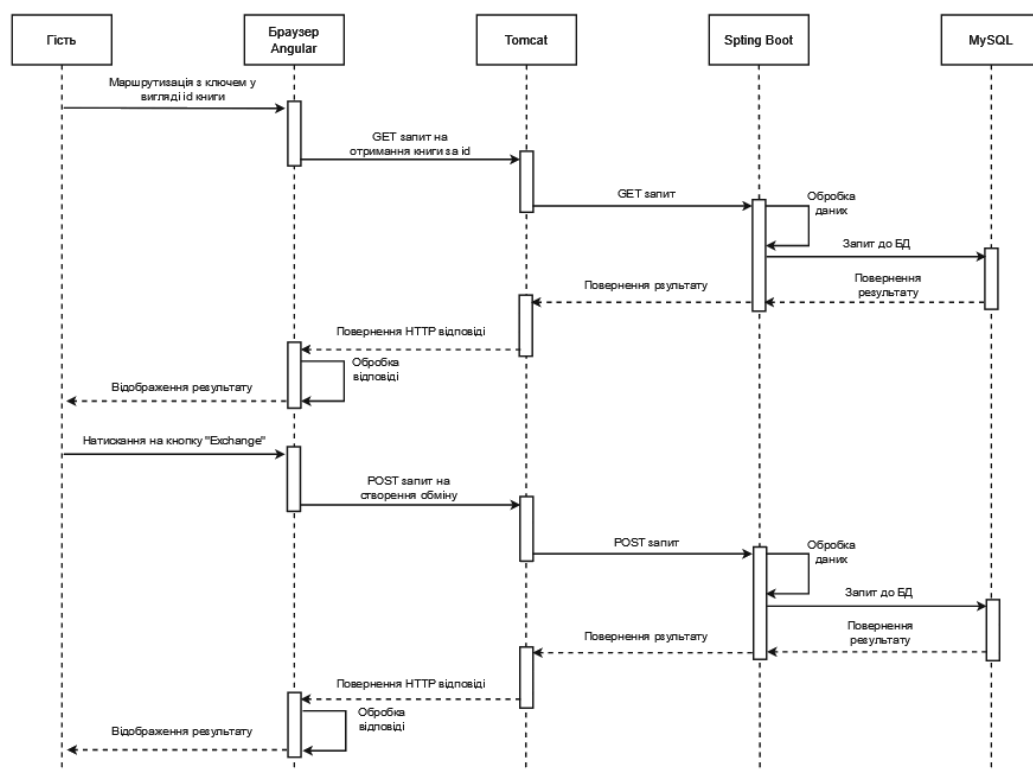


Рисунок 2.8 – Діаграма послідовностей «Створення обміну»

Далі користувач, який є власником книги, зможе підтвердити обмін, після чого книга змінить свого власника. Під час підтвердження обміну сервер переконається, що користувач є власником книги, є учасником обміну та не є його ініціатором.

2.5 Визначення, акроніми, скорочення

API (Application Programming Interface) — це набір правил і протоколів для побудови та інтеграції програмного забезпечення. API дозволяє різним програмам взаємодіяти одна з одною, обмінюючись даними та функціями.

Bean – об'єкт Java, життєвим циклом яким керує контейнер Spring.

CRUD – операції Create (створення), Read (читання), Update (оновлення) та Delete (видалення), які зазвичай виконуються з даними в системах баз даних або програмах.

DTO (Data Transfer Object) – це об'єкт, який використовується для передачі даних між прошарками програми або між програмами. Зазвичай DTO містить лише поля для зберігання даних та не містить бізнес-логіки. Використання DTO спрощує передачу даних із великою кількістю полів та дозволяє уникнути передачі зайвих деталей між компонентами програми.

Hibernate – це фреймворк для роботи з базами даних у Java-додатках. Він надає об'єктно-реляційне відображення, що дозволяє програмістам працювати з об'єктами Java, не звертаючись безпосередньо до таблиць бази даних із SQL.

HTTP (Hypertext Transfer Protocol) – це протокол передачі даних в мережі Інтернет. Він використовується для взаємодії між клієнтами та серверами. HTTP визначає спосіб, яким запити відправляються від клієнта до сервера та відповіді від сервера до клієнта.

JPA (Java Persistence API) – це стандарт Java для управління об'єктно-реляційним відображенням даних у Java-додатках. Він надає способи роботи з об'єктами в базі даних, не прив'язуючись до конкретної реалізації бази даних.

JPQL (Java Persistence Query Language) – це мова запитів, яка використовується для виконання запитів до об'єктно-реляційних баз даних за допомогою Java Persistence API.

JSON (JavaScript Object Notation) – це легкий формат обміну даними, який використовується для передачі структурованих даних між веб-сервером і веб-клієнтом, а також між різними компонентами програмного забезпечення.

JWT (JSON Web Token) – це компактний, самостійний та зашифрований формат для передачі відкритих даних між двома сторонами в безпечний спосіб. JWT зазвичай використовується для автентифікації та авторизації веб-додатків.

Open Source (відкритий код) – програмне забезпечення, чиї вихідні тексти доступні для використання, зміни та поширення.

REST (Representational State Transfer) – це архітектурний стиль для розробки веб-сервісів, що базується на принципах простоти, стандартних інтерфейсів та безстандартного взаємодії.

RESTful – системи, побудовані за архітектурним стилем REST.

SQL (Structured Query Language) – це мова запитів, яка використовується для керування та маніпулювання даними у реляційних базах даних.

Автентифікація – це процес перевірки ідентичності користувача для підтвердження, що він дійсно той, за кого себе видає. Це зазвичай включає в себе представлення облікових даних, таких як ім'я користувача та пароль, для перевірки збережених у системі облікових записів.

Авторизація – це процес визначення прав доступу користувача до певних ресурсів чи функціональності системи після успішної автентифікації.

Анотація – це спеціальний тип метаданих, який може бути використаний для надання додаткової інформації про клас, метод, змінну або інший елемент програми.

БД – база даних.

Сервлет – це програмна компонента веб-додатка, написана на мові програмування Java, яка використовується для обробки та відповіді на HTTP-запити від клієнтів.

СКБД – системи керування базами даних.

Фреймворк – це набір загальних правил, структур, бібліотек та інструментів, які використовуються для розробки програмного забезпечення.

3 ПРОГРАМНА РОЗРОБКА СИСТЕМИ

3.1 Розробка серверної частини

На даному етапі буде створено REST API серверний додаток, до якого потім зможуть звертатись клієнти.

Для цього буде здійснено ініціалізацію проекту Spring Boot з переліком доданих залежностей та їх призначенням. Буде проведено базове налаштування Spring Boot додатку, а саме: налаштування взаємодії з базою даних, веб налаштування, налаштування SMTP серверу, налаштування WebSocket та двигуна шаблонів Thymeleaf.

Буде розглянуто приклади створення сутностей бази даних з їх зв'язками, створення сервісних компонентів, які будуть відповідати за логіку додатку, а також створення REST контролерів для обміну даними з клієнтами.

Також буде приділено увагу безпеці додатку, що включає процес авторизації та аунтентифікації за допомогою JWT (JSON Web Token) та безпечному зберіганню даних.

3.1.1 Ініціалізація Spring Boot додатку

Для ініціалізації проекту буде використано Spring Initializer – це веб-сайт або веб-інструмент, який використовується для створення проектів Spring Boot. Проекти також можна створювати вручну, проте це є відчутно складніший процес, який може займати більше часу.

При ініціалізації проекту були встановлені такі параметри як: тип проекту – Maven, версія Spring Boot – 3.2.5, мова програмування – Java 17.

Також з огляду на функціонал проекту було додано наступні залежності:

а) Lombok – Java бібліотека анотацій, яка допоможе значно зменшити час на написання шаблонного коду (наприклад гетери та сетери);

б) Thymeleaf – Java двигун шаблонів, який знадобиться для відправлення шаблонних HTML сторінок клієнтам;

в) Spring Security – фреймворк, який спрямований на надання аунтентифікації та авторизації для Java-додатків;

г) SQL залежності:

1) Spring Data JPA – частина широкої родини Spring Data (проект який включає в себе багато під проектів, що спеціалізуються на доступі до даних), що спрощує впровадження репозиторіїв на основі JPA (Java Persistence API);

2) MySQL Driver – це конкретна реалізація JDBC (Java Database Connectivity), спеціально призначена для взаємодії з MySQL базами даних;

3) H2 Database – вбудована база даних з відкрити вихідним кодом, яка знаходиться в пам'яті, а не на диску. Цей інструмент буде використовуватися при тестуванні.

г) WebSocket – протокол, який допоможе обробляти повідомлення між користувачами в реальному часі;

д) I/O залежності:

1) Validation – це повністю сумісна реалізація JSR-303/309 (Java Specification Requests, які визначають стандарт Java Bean Validation API), яка дозволяє виражати та перевіряти обмеження застосунку;

2) Java Mail Sender – це інтерфейс наданий фреймворком Spring, який служить абстракцією для надсилання повідомлень електронної пошти. Замість того, щоб безпосередньо мати справу зі складністю JavaMail API, розробники можуть взаємодіяти з JavaMailSender за допомогою набору чітко визначених методів і властивостей, що робить процес надсилання електронних листів більш простим і менш схильним до помилок;

е) Spring Web – фреймворк, який використовується для побудови веб, включно з RESTful, застосунків на Java, основним компонентом якого є Spring Web MVC (Model-View-Controller), який забезпечує надійну та гнучку архітектуру для розробки веб додатків. Включає в себе веб-сервер Tomcat.

Також слід додати, що за замовченням в проект було додано залежність “spring-boot-starter-test”, яка включає в себе такі інструменти для проведення

тестів: JUnit, Spring Test & Spring Boot Test, AssertJ, Hamcrest, Mockito, JSONassert, JsonPath, Awaitility.

На рисунку 3.1 зображено усі налаштування ініціалізації проекту.

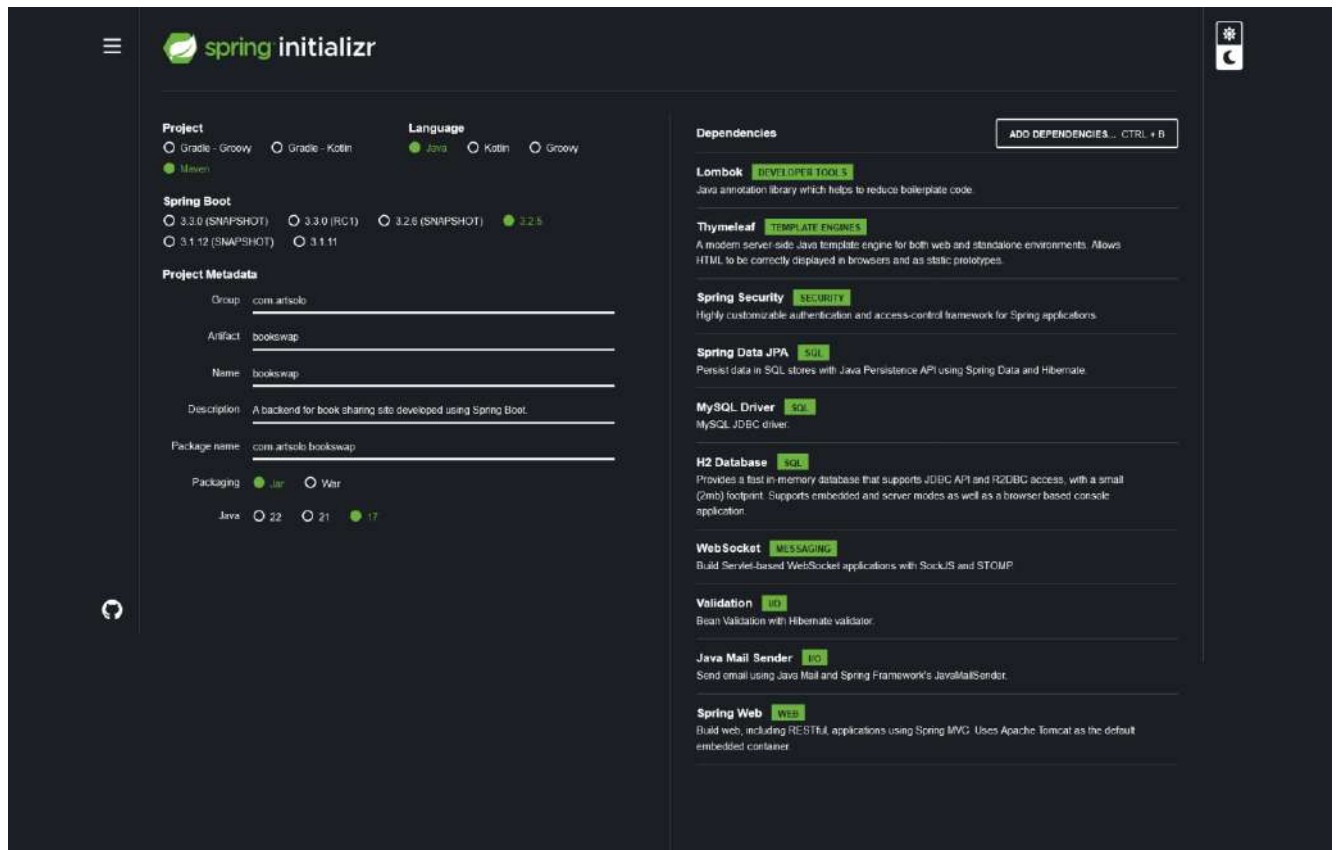


Рисунок 3.1 – Ініціалізація Spring Boot проекту

Після генерації проекту було отримано .zip файл з проектом, який вже є робочим додатком, який можна запустити в обраному середовищі розробки.

3.1.2 Базове налаштування Spring Boot додатку

Налаштування Spring Boot додатку починається з файлу “application.properties”. Цей файл є ключовим для встановлення властивостей проекту. У ньому ми визначаємо широкий спектр параметрів, включаючи зміну порту, конфігурацію підключення до бази даних, налаштування зв'язку з сервером та багато іншого. Далі буде розглянуто встановлені властивості для проекту.

За замовченням порт, який використовує сервер, є 8080. Його можна змінити як показано в лістингу 3.1.

Лістинг 3.1 – Встановлення порту сервера на 8181

```
server.port = 8181
```

Тепер слід налаштувати підключення до бази даних. Єдине, що перед цим слід зробити, так це створити базу даних, у нашому випадку, вона матиме назву “bookswap”. На лістингу 3.2 зображено встановлені властивості підключення.

Лістинг 3.2 – Налаштування підключення до бази даних

```
spring.datasource.url = jdbc:mysql://localhost:3306/bookswap
spring.datasource.username = root
spring.datasource.password = password
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
```

Як показано в лістингу 3.3, можна також вказати максимальним розмір запитів.

Лістинг 3.3 – Встановлення максимального розміру запиту

```
spring.servlet.multipart.enabled = true
spring.servlet.multipart.file-size-threshold = 2KB
spring.servlet.multipart.max-file-size = 100MB
spring.servlet.multipart.max-request-size= 100MB
```

Далі для того, щоб зробити процес розробки більш зручним, слід встановити властивості специфічні для Spring Data JPA, як вказано в лістингу 3.4.

Лістинг 3.4 – Встановлення властивостей Spring Data JPA

```
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.format_sql = true
```

Перша властивість встановлена в “update”, вказує на те що Hibernate автоматично виконуватиме оновлення таблиць бази даних (створення нових таблиць, обмежень, проте не видалення) на основі моделі даних визначеної в коді

додатку без втручання розробника. Останні дві властивості додадуть та стилізують вивід здійснених запитів до бази даних під час роботи додатку.

Наступним кроком буде встановлення властивостей Spring Mail, для надсилання повідомлень поштою, сюди ввійде також налаштування SMTP сервера, лістинг 3.5.

Лістинг 3.5 – Встановлення властивостей Spring Mail

```
spring.mail.host = smtp.gmail.com
spring.mail.port = 587
spring.mail.username = username@gmail.com
spring.mail.password = password
spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.starttls.enable = true
```

Залишилось лише встановити налаштування для Thymeleaf, для відправлення встановлених HTML шаблонів, які будуть надсилатися клієнтам. Для цього було створено клас, обернутий анотацією @Configuration. Анотація @Configuration у Spring є частиною основного фреймворку Spring. Анотація Spring Configuration вказує на те, що в класі є методи визначення @Bean. Таким чином, контейнер Spring може обробляти клас і створювати Spring Beans для використання в додатку.

Spring Beans - це об'єкти, які управляються контейнером Spring. Вони є основними будівельними блоками додатків, побудованих на основі Spring Framework. Spring контейнер керує життєвим циклом цих об'єктів, включаючи їх створення, конфігурацію та знищення.

Spring Beans дозволяють легко управляти залежностями між компонентами додатку та забезпечують інверсію керування, що сприяє створенню більш масштабованих, гнучких та легко тестових додатків.

В лістингу 3.6 наведено конфігураційний клас “ThymeleafConfig”. У цьому класі визначено два методи, які позначені анотацією @Bean. Кожен з цих методів повертає об'єкт, який Spring контейнер буде управляти як бін (Spring Bean).

Лістинг 3.6 – Налаштування Thymeleaf

```

@Configuration
public class ThymeleafConfig {
    @Bean
    public ClassLoaderTemplateResolver htmlTemplateResolver() {
        ClassLoaderTemplateResolver emailTemplateResolver = new
        ClassLoaderTemplateResolver();

        emailTemplateResolver.setPrefix("/templates/");
        emailTemplateResolver.setSuffix(".html");

        emailTemplateResolver.setTemplateMode(TemplateMode.HTML);

        emailTemplateResolver.setCharacterEncoding(StandardCharsets.UTF
        _8.name());
        return emailTemplateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine = new
        SpringTemplateEngine();

        templateEngine.addTemplateResolver(htmlTemplateResolver());
        return templateEngine;
    }
}

```

Перший метод `htmlTemplateResolver()` створює та налаштовує об'єкт `ClassLoaderTemplateResolver`, який використовується для розв'язання шаблонів HTML. Він встановлює префікс та суфікс для шляхів шаблонів, визначає режим шаблону як HTML та встановлює кодування символів.

Другий метод `templateEngine()` створює та налаштовує об'єкт `SpringTemplateEngine`, який використовується для обробки шаблонів Thymeleaf.

Для завершення базового налаштування залишилось тільки налаштувати WebSocket. Для цього було створено клас “`WebSocketConfiguration`”, який окрім анотації `@Configuration`, також обернутий у анотацію `@EnableWebSocketMessageBroker`, яка включає підтримку обробку повідомлень WebSocket з використанням брокера повідомлень STOMP. Також клас реалізує інтерфейс “`WebSocketMessageBrokerConfigurer`”, що надає змогу для налаштування методів обробки повідомлень.

В лістингу 3.7 наведено конфігураційний клас “WebSocketConfiguration”.

Лістинг 3.7 – Клас “WebSocketConfiguration”

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfiguration implements
WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry)
{registry.addEndpoint("/ws").setAllowedOrigins("http://localhost:4200
").withSockJS();
    }
    @Override
    public void configureMessageBroker(MessageBrokerRegistry
registry) {
        registry.setApplicationDestinationPrefixes("/app");
        registry.enableSimpleBroker("/user");
        registry.setUserDestinationPrefix("/user");
    }
}
```

Метод `registerStompEndpoints()`, додає кінцеву точку `/ws`, яка буде використана для підключення клієнтів `WebSocket`, дозволяє підключення з `Angular` клієнта (`http://localhost:4200`), а також додає підтримку `SockJS`.

Метод `setApplicationDestinationPrefixes("/app")` вказує, що повідомлення, які починаються з префікса `/app`, будуть направлятися до методів обробників повідомлень, які будуть реалізовані в контролерах. Метод `enableSimpleBroker("/user")` включає простий брокер повідомлень з призначенням `/user`, що означає, що повідомлення, що починаються з цього префіксу, будуть оброблятися брокером повідомлень і доставлятися користувачам. Метод `setUserDestinationPrefix("/user")` встановлює префікс для повідомлень, що призначені конкретному користувачу.

На цьому базове налаштування додатку можна вважати завершеним.

3.1.3 Створення сутностей та їх зв'язків

Створення сутностей та зв'язків між ними буде розглядатися на прикладі наступних таблиць: «Книги», «Користувачі», та «Списки_бажаного». Цього вистачить для створення уявлення над процесом побудови бази даних для системи обміну книгами, включаючи визначення атрибутів для кожної сутності,

встановлення первинних і зовнішніх ключів, а також реалізацію зв'язків між таблицями.

В лістингу 3.8 наведено створення сутності «Книги».

Лістинг 3.8 – Створення сутності «Книги»

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "books")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "book_id", nullable = false)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "owner_id")
    private User owner;
    @Column(length = 50)
    private String title;
    @Column(length = 50)
    private String author;
    @Column(length = 1000)
    private String description;
    @ManyToMany
    @JoinTable(
        joinColumns = @JoinColumn(name = "book_id"),
        inverseJoinColumns = @JoinColumn(name = "genre_id")
    )
    private List<Genre> genres;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "quality_id")
    private Quality quality;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "status_id")
    private Status status;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "language_id")
    private Language language;
    @Lob
    @Column(columnDefinition="LONGBLOB")
    private byte[] photo;
    @OneToMany(mappedBy = "book", orphanRemoval = true)
    private List<Review> reviews;
    @OneToMany(mappedBy = "book", orphanRemoval = true)
    private List<Wishlist> wishlist;
    @OneToMany(mappedBy = "book", orphanRemoval = true)
    private List<Exchange> exchanges;
    @OneToMany(mappedBy = "book", orphanRemoval = true)
    private List<Note> notes;
}
```

Детально розглянемо код:

- анотація `@Data` належить бібліотеці Lombok, автоматично генерує гетери та сетери для всіх полів класу, а також методи `toString()`, `equals(Object o)` та `hashCode()`;
- анотації `@AllArgsConstructor` та `@NoArgsConstructor` належить бібліотеці Lombok, автоматично генерує конструктор з аргументами для всіх полів та без аргументів, відповідно;
- анотація `@Builder` належить бібліотеці Lombok, автоматично генерує патерн “Builder”, який забезпечує зручний та безпечний спосіб створення об'єктів класу, особливо коли клас має багато полів або коли потрібно створювати об'єкти з різними комбінаціями полів;
- анотація `@Entity` вказує, що клас є сутністю JPA;
- за замовченням назва сутності буде збігатися з назвою класу, анотація `@Table` явно вказує, що сутність повинна зберігатися під назвою “books”;
- анотація `@Id` вказує, що поле є первинним ключем;
- анотація `@GeneratedValue(strategy = GenerationType.IDENTITY)`, вказує, що значення первинного ключа автоматично генерується базою даних (автоінкремент);
- за допомогою анотації `@Column` вказуються властивості поля таблиці;
- для створення зв'язку “багато до одного” використовується анотація з відповідною назвою `@ManyToOne`, `@JoinColumn` в свою чергу визначає колонку в таблиці, яка буде використано як зовнішній ключ;
- для створення зв'язку “багато до багатьох” використовується анотація з відповідною назвою `@ManyToMany`, `@JoinTable` вказує на проміжну таблицю, що використовуватиметься для збереження зв'язку між таблицями;
- для створення зв'язку “один до багатьох” використовується анотація з відповідною назвою `@OneToMany`, де `mappedBy` вказує, що, в даному випадку, зв'язок керується полем `book` в іншій сутності.

Тепер розглянемо приклад створення зв'язків між таблицями зі складеними ключами.

В лістингу 3.9 наведено приклад створення складеного ключа.

Лістинг 3.9 – Складений ключ

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Embeddable
public class CompositeKey implements Serializable {
    private Long user_id;
    private Long book_id;
}
```

Анотація `@Embeddable`, вказує що клас є вбудованим типом, і може використовуватися як частина іншої сутності. В той час як реалізація інтерфейсу “`Serializable`” дозволяє класу бути перетвореним у байтів потік, що необхідно для використання його як ключа в JPA.

Тепер зв’яжемо таблицю «Книги» з «Списики_бажаного» (лістинг 3.10).

Лістинг 3.10 – Створення сутності «Списики_бажаного»

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "wishlist")
public class Wishlist {
    @EmbeddedId
    private CompositeKey wishlistId;

    @ManyToOne
    @MapsId("user_id")
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @MapsId("book_id")
    @JoinColumn(name = "book_id")
    private Book book;
}
```

Анотація `@EmbeddedId` використовується для вказівки, що певне поле є вбудованим композитним ключем. Це означає, що клас, який представляє цей

ключ, буде вбудований в сутність і його поля будуть використовуватися як частина первинного ключа.

Анотація `@MapsId("user_id")` вказує, що поле `user_id` в `CompositeKey` повинно бути використане як зовнішній ключ для зв'язку з сутністю `User`. В свою чергу анотація `@MapsId("book_id")` вказує, що поле `book_id` в `CompositeKey` повинно бути використане як зовнішній ключ для зв'язку з сутністю `Book`.

Тепер розглянемо сутність «Користувачі». Ця сутність є важливим компонентом при авторизації та автентифікації в системі користувачів. В лістингу 3.11 наведено створенні сутності «Користувачі».

Лістинг 3.11 – Створення сутності «Користувачі»

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id", nullable = false)
    private Long id;
    @Column(unique = true, length = 35)
    private String nickname; private String email;
    private String password; private Integer points;
    @Lob
    @Column(columnDefinition="BLOB")
    private byte[] photo;
    private Boolean activity;
    @Column(name = "registration_date")
    private LocalDate registrationDate;
    private String country;
    private String city;
    @Enumerated(EnumType.STRING)
    private Role role;
    @OneToMany(mappedBy="user", orphanRemoval = true, cascade =
CascadeType.ALL)
    private List<Token> tokens;
    @OneToMany(mappedBy = "owner", orphanRemoval = true)
    private List<Book> books;
    @OneToMany(mappedBy = "user", orphanRemoval = true)
    private List<Wishlist> wishlist;
    @OneToMany(mappedBy = "user", orphanRemoval = true)
    private List<Review> reviews;
    @OneToMany(mappedBy = "initiator", orphanRemoval = true)
    private List<Exchange> initiations;
    @OneToMany(mappedBy = "recipient", orphanRemoval = true)
    private List<Exchange> recipients;}
```

Анотація `@Enumerated(EnumType.STRING)` використовується для вказівки JPA, як перетворити значення перерахування (Enum) на рядковий тип бази даних під час збереження в базі даних. Давайте розглянемо, як це працює у контексті класу User.

Оскільки сутність User представляє користувача системи, який виконує важливі дії та має права доступу, вона повинна реалізовувати інтерфейс UserDetails, який визначає необхідні методи для отримання інформації про користувача, необхідної для аутентифікації та авторизації в системі. Реалізація інтересу дозволить перенавантажити наступні методи наведені в лістингу 3.12.

Лістинг 3.12 – Реалізація інтерфейсу UserDetails класом User

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@Table(name = "users")
public class User implements UserDetails {
    ...
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getUsername() {return email;}

    @Override
    public boolean isAccountNonExpired() {return true;}

    @Override
    public boolean isAccountNonLocked() {return true;}

    @Override
    public boolean isCredentialsNonExpired() {return true;}

    @Override
    public boolean isEnabled() {return activity;}
}
```

Після створення всіх необхідних класів можна здійснити спробу запуску додатку, і у випадку коректної конфігурації, Hibernate автоматично згенерує SQL запити для створення таблиць та встановлення їх зв'язків у базі даних.

Результат запуску додатку зображено на рисунку 3.2.

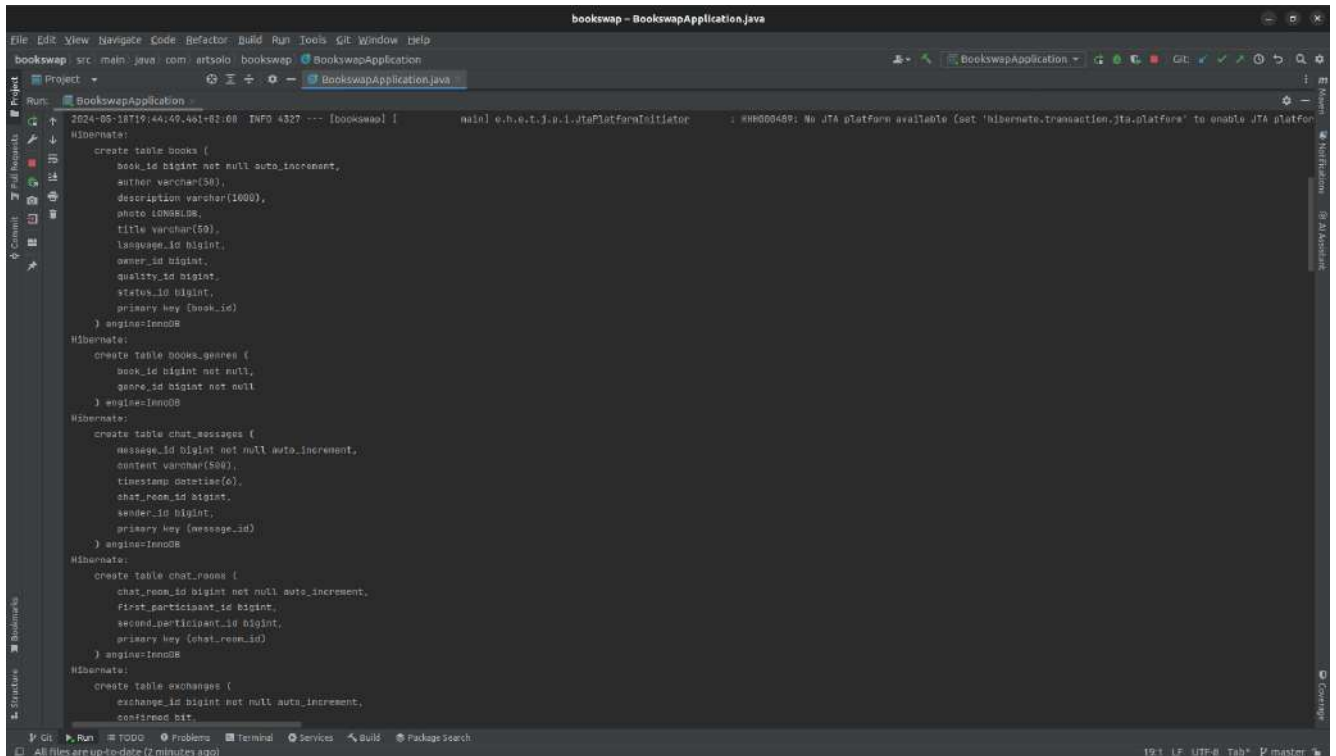


Рисунок 3.2 – Результат запуску додатку

За результатом запуску програми можна зробити висновок, що всі параметри для сутностей були встановлені правильно і Hibernate зміг їх вдало створити. Це також можна перевірити звернувшись до бази даних через консоль, або через MySQL Workbench (за наявності).

3.1.4 Створення сервісних компонентів

Створення сервісних компонентів буде показано за прикладом сутності «Обміни».

В Spring Boot сервісні компоненти – це класи, які містять анотацію `@Service` та використовуються для написання бізнес логіки на окремому рівні, відділеному від REST контролерів.

Анотація `@Service` в свою чергу анотована у `@Component` (анотація, що допомагає Spring сканувати класи та реєструвати їх як біни) й використовується для вказівки того, що клас містить бізнес логіку і, на даний момент, вона не має

жодних особливостей (тому можна використати замість неї анотацію `@Component`), проте в майбутньому це може змінитись.

Перед впровадженням сервісу слід створити усі необхідні інтерфейси репозиторіїв Spring Data JPA, які призначені для обробки операцій доступу до даних певних сутностей. Приклад такого інтерфейсу наведено в лістингу 3.13.

Лістинг 3.13 – Інтерфейс “ExchangeRepository”

```
public interface ExchangeRepository extends
JpaRepository<Exchange, Long> {
    List<Exchange> findAllByInitiatorId(Long initiatorId);
    List<Exchange> findAllByRecipientId(Long recipientId);
    List<Exchange> findAllByBookId(Long bookId);
}
```

Клас наслідує інтерфейс `JpaRepository` з параметрами `Exchange` як типом сутності та `Long` як типом первинного ключа. Інтерфейсом для наслідування `JpaRepository` було обрано через найбільший функціонал, порівняно з `CrudRepository`, який надає тільки CRUD функціонал, та `PagingAndSortingRepository`, який надає методи для пагінації та сортування (`JpaRepository` наслідує `PagingAndSortingRepository`, який в свою чергу наслідує `CrudRepository`).

Через угоду про іменування методів, Spring Data JPA стає зручним механізмом для створення користувацьких методів. Ця функція дозволяє розробникам визначати методи запиту безпосередньо в інтерфейсі репозиторію без необхідності написання JPQL (Java Persistence Query Language), як це показано в лістингу 3.14, тож суть методу можна зрозуміти з назви.

Spring Data JPA аналізує назву методу та парсить його для створення відповідного запиту. Ім'я методу ділиться на частини: префікс (наприклад, `like`, `find`, `read`, `query`, `count`, `get`), тему (наприклад, `All`, `First`, `Distinct`) та умову (наприклад, `by`, `and`, `or`). Після умови ім'я методу визначає поля сутності, які використовуються в запиті.

Проте для більш комплексних запитів, можна скористатися анотацією `@Query` для написання власних запитів за допомогою JPQL. Приклад наведено в лістингу 3.14.

Лістинг 3.14 – Приклад використання `@Query`

```
public interface TokenRepository extends JpaRepository<Token,
Long> {
    @Query("""
select t from Token t inner join User u on t.user.id = u.id
where u.id = :userId and (t.expired = false or t.revoked = false)
""")
    List<Token> findAllValidTokensByUser(Long userId);
    Optional<Token> findByToken(String token);
}
```

Останнє, що слід зробити перед створення сервісу для обмінів, так це створити DTO (Data Transfer Object) для сутності «Обміни». DTO дозволяють налаштувати дані, які надсилаються клієнту, тобто можна включити лише необхідні поля, змінити назви полів або обчислити похідні дані. DTO сутності «Обміни» наведено в лістингу 3.15.

Лістинг 3.15 – DTO сутності «Обміни»

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ExchangeResponse {
    private Long id;
    private Long initiatorId;
    private String initiator;
    private Long recipientId;
    private String recipient;
    private Long bookId;
    private String book;
    private Boolean confirmed;
}
```

Тепер можна перейти до логіки обмінів. Так було створено клас “ExchangeService” анотований у `@RequiredArgsConstructor` та `@Service`.

Ін'єкція залежностей на основі конструктора є більш рекомендованим варіантом оскільки є безпечнішим та краще піддається тестуванню ніж польова ін'єкція, або на основі сетерів. Для цього буде використовуватися анотація `@RequiredArgsConstructor` бібліотеки Lombok. Ця анотація автоматично генерує конструктор із обов'язковими аргументами, тобто полів `final` та поля, позначені `@NonNull`. Обов'язковими полями класу “ExchangeService” стане “ExchangeRepository” та “BookService”.

Методи для створення та видалення обміну наведено в лістингу 3.16.

Лістинг 3.16 – Методи створення та видалення обміну

```
public void createNewExchange(User initiator, Book book) {
    Exchange exchange = Exchange.builder()
        .initiator(initiator)
        .recipient(book.getOwner())
        .book(book)
        .confirmed(Boolean.FALSE)
        .build();
    exchangeRepository.save(exchange);
}

public boolean deleteExchange(Exchange exchange) {
    exchangeRepository.deleteById(exchange.getId());
    return !exchangeRepository.existsById(exchange.getId());
}
```

Суть використаних методів `save()`, `deleteById()` та `existsById()` можна зрозуміти з назви, лише додам, що метод `save()` можна використовувати й для оновлення даних.

Методи отримання списку ініційованих обмінів користувача наведено в лістингу 3.17.

Лістинг 3.17 – Метод для отримання списку ініційованих обмінів

```
public List<ExchangeResponse>
getAllUserInitiateExchanges(User user) {
    List<Exchange> exchanges =
exchangeRepository.findAllByInitiatorId(user.getId());
    return
exchanges.stream().map(this::getExchangeResponse).collect(Collectors.toList());
}
```

Метод для отримання списку обмінів користувача в яких він є отримувачем ініційованого обміну буде працювати за таким же принципом, тільки замість методу `findAllByInitiatorId(Long id)` класу `ExchangeRepository` буде використовуватися `findAllByRecipientId(Long id)`.

Метод для підтвердження обміну наведено в лістингу 3.18.

Лістинг 3.18 – Метод підтвердження обміну

```
@Transactional
public boolean confirmExchange(Exchange exchange) {
    if (bookService.changeBookOwner(exchange.getBook(),
exchange.getInitiator())) {
        if (changeExchangeConfirmStatus(exchange)) {
            setNewRecipientForExchanges(
exchangeRepository.findAllByBookId(exchange.getBook().getId()),
exchange.getInitiator()
);
            return true;
        }
    }
    return false;
}
```

Анотація `@Transactional` у Spring використовується для керування межами транзакцій декларативним способом. Транзакції гарантують, що всі серії операцій будуть успішними або невдалими, зберігаючи цілісність і послідовність даних. В кінці методу всі обміни прив'язані до книги, яку власник передає іншому користувачу, будуть “передані” (лістинг 3.19) новому власнику, який тепер стане отримувачем цих обмінів і зможе передати книгу одному з ініціаторів.

Лістинг 3.19 – Метод зміни отримувача ініційованого обміну

```
public void setNewRecipientForExchanges(List<Exchange>
exchanges, User newRecipient) {
    exchanges.forEach(exchange -> {
        if
(!exchange.getInitiator().getId().equals(newRecipient.getId())) {
            exchange.setRecipient(newRecipient);
            exchangeRepository.save(exchange);
        }
    });
}
```

На рисунку 3.3 зображено алгоритм підтвердження обміну.

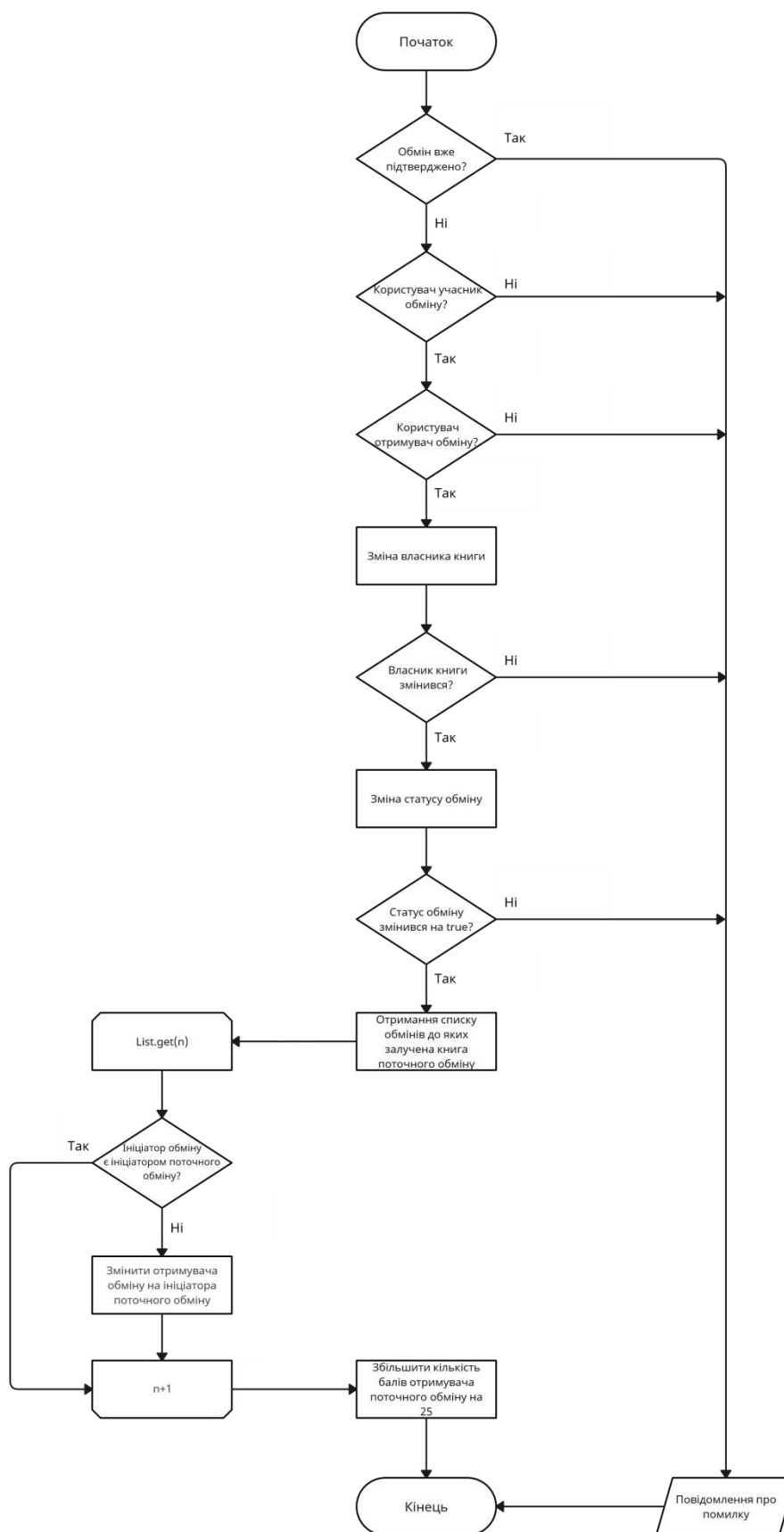


Рисунок 3.3 – Алгоритм підтвердження обміну

Також були створені допоміжні методи, які будуть викликані REST контролером в майбутньому для слідування алгоритму підтвердження обміну, їх наведено в лістингу 3.20.

Лістинг 3.20 – Допоміжні методи

```
public boolean isExchangeConfirmed(Exchange exchange) {
    return exchange.getConfirmed();
}

public boolean isUserRecipientOfExchange(Exchange exchange, User
providedRecipient) {
    User actualRecipient = exchange.getRecipient();
    return actualRecipient.getId().equals(providedRecipient.getId());
}

public boolean isUserParticipantOfExchange(Exchange exchange,
User user) {
    User recipient = exchange.getRecipient();
    User initiator = exchange.getInitiator();
    return (user.getId().equals(recipient.getId()) ||
user.getId().equals(initiator.getId()));
}
```

На цьому процес написання логіки обміну книг можна вважати закінченим.

3.1.5 Створення REST контролерів

Анотація Spring `@RestController` — це анотація, яка застосовується до класу, щоб позначити його як обробник запитів. Анотація використовується для створення веб-служб RESTful за допомогою Spring MVC та піклується про зіставлення даних запиту з визначеним методом обробки запиту. Коли тіло відповіді генерується з методу обробника, воно перетворює його у відповідь JSON або XML. `@RestController` анотована у `@Controller`, який в свою чергу, як і `@Service`, анотований у `@Component`, проте на відміну від `@Service`, її замінити не можна, оскільки тільки з цією анотацією можна використовувати `@RequestMapping`.

Анотація `@RequestMapping` використовується для зіставлення запитів на методи контролерів. Вона має різні атрибути для відповідності за URL, HTTP-методом, параметрами запиту, заголовками та типами медіа. Анотацію можна

використовувати на рівні класу для спільних зіставлень або на рівні методу для звуження до конкретного зіставлення кінцевої точки.

Існують також варіанти `@RequestMapping`, специфічні для HTTP-методів: `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, `@PatchMapping`.

Надалі буде приведено приклад створення REST контролера для впровадження кінцевих точок, для взаємодії з сервісом, який вміщує в собі логіку обміну книгами.

Було створено клас, який анотовано `@RestController`, `@RequiredArgsConstructor` та `@RequestMapping("/api/v1/exchange")`.

Далі наведено впроваджений метод для отримання усіх ініційованих обмінів поточного користувача (лістинг 3.21).

Лістинг 3.21 – Метод для отримання списку ініційованих обмінів

```
@GetMapping("/get/initiation")
public
ResponseEntity<SuccessResponse<List<ExchangeResponse>>>
getAllInitiateExchanges(Principal currentUser) {
    User user = (User)
    ((UsernamePasswordAuthenticationToken) currentUser).getPrincipal();
    return ResponseEntity.ok().body(new
    SuccessResponse<>(exchangeService.getAllUserInitiateExchanges(user))
    );
}
```

Таким чином, щоб отримати список, знадобиться здійснити запит “`http://localhost:8080/api/v1/exchange/get/initiation`”, після чого сервер надійшло відповідь у форматі JSON.

Анотація `@PathVariable` у Spring MVC використовується для вилучення значень із шляху URI та прив'язки їх до параметрів методу в контролері. Вона є важливою частиною обробки RESTful веб-сервісів у Spring, дозволяючи динамічне та гнучке зіставлення URL.

Дана анотація була використана при створенні кінцевої точки для підтвердження обміну (лістинг 3.22).

Лістинг 3.22 – Підтвердження обміну

```
@PostMapping("/confirm/{id}")
public ResponseEntity<?> confirmExchangeById(@PathVariable Long
id, Principal currentUser) {
    Exchange exchange = exchangeService.getExchangeById(id);
    User user = (User) ((UsernamePasswordAuthenticationToken)
currentUser).getPrincipal();
    if (!exchangeService.isExchangeConfirmed(exchange)) {
        if (exchangeService.isUserRecipientOfExchange(exchange, user)){
            if (exchangeService.confirmExchange(exchange))
                userService.increaseUserPoints(25, exchange.getRecipient());
            return
                ResponseEntity.ok().body(MessageResponse.builder().message("Exc
change was confirmed successfully").build());}return
                ResponseEntity.badRequest().body(ErrorResponse.builder().error(
new ErrorDescription(HttpStatus.BAD_REQUEST.value(), "Failed to
confirm exchange")).build());}return
                ResponseEntity.badRequest().body(ErrorResponse.builder().error(
new ErrorDescription(HttpStatus.BAD_REQUEST.value(), "You are
not recipient of exchange")).build());}return
                ResponseEntity.badRequest().body(ErrorResponse.builder().error(
new ErrorDescription(HttpStatus.BAD_REQUEST.value(), "Exchange
is already confirmed")).build());
    }
}
```

Таким чином для підтвердження обміну з ідентифікатором 3 користувачу доведеться надіслати запит “ <http://localhost:8080/api/v1/exchange/confirm/3>”.

Також замість `@PathVariable` можна скористатися `@RequestParam`. Запит виглядав би таким чином: <http://localhost:8080/api/v1/exchange/confirm?id=3>.

Тож можна зробити висновок, що створення RESTful API за допомогою Spring Boot є доволі простим.

3.1.6 Безпека додатку

Основою безпеки Spring Boot проекту є Spring Security. Як і в усіх проектах Spring, справжня потужність Spring Security виявляється у тому, наскільки легко його можна розширити для відповідності власним вимогам.

У Spring MVC всі вхідні HTTP-запити перенаправляються через один сервлет, відомий як Dispatcher Servlet. Цей сервлет направляє ці запити до контролерів, де визначається кінцеві точки додатка. На цьому етапі втручається Spring Security, вводячи серію захисних фільтрів (Security Filter Chain) перед тим, як HTTP-запити досягають Dispatcher Servlet. Це означає, що кожен вхідний запит

буде проходити через ці фільтри по одному. Такий підхід дозволяє перевіряти стани аутентифікації та авторизації до того, як запит досягне Dispatcher Servlet, а отже й контролерів. В цілому, це основна функція Spring Security.

Типом авторизація та автентифікації було обрано JWT, який стане першим фільтром в серії.

JWT (JSON Web Token) - це компактний, самостійний та зашифрований формат для передачі відкритих даних між двома сторонами в безпечний спосіб. JWT зазвичай використовується для автентифікації та авторизації веб-додатків. Він складається з трьох частин: заголовка, заявки та підпису. Заголовок містить інформацію про тип токена та алгоритм шифрування. Заявка містить дані, які треба передати, такі як ідентифікатор користувача або час видачі токена. Підпис використовується для перевірки цілісності та автентичності токена. JWT може бути використаний для створення сеансів, зберігання користувацької інформації або передачі даних між веб-додатками. Він часто використовується в сучасних системах автентифікації, таких як OAuth та OpenID Connect.

Алгоритмом шифрування стане HS256. Заявка буде містити пошту користувача, яка для кожного є унікальною, а також час вичерпання токена у 24 години з моменту генерації. Для підписання буде використано секретний ключ. Сервіс для генерації, валідації та вилучення інформації з токена наведено в додатку А.

Для впровадження JWT авторизації та автентифікації потрібно створити власний фільтр, який стане першим в серії фільтрів.

Клас “JwtAuthenticationFilter” наслідуватиме клас “OncePerRequestFilter” й перенавантажить головний метод doFilterInternal(), який міститиме обробку всіх вхідних HTTP запитів. Метод намагатиметься отримати JWT з заголовка Authorization HTTP-запиту. Якщо заголовок відсутній або не починається з “Bearer ”, фільтр продовжує роботу наступного фільтра в ланцюжку без подальшої обробки. У разі вилучення електронної пошти і якщо в “SecurityContextHolder” немає поточної аутентифікації, фільтр виконує валідацію токена. У разі не відповідності вимогам токена, повертається помилка, в іншому

випадку токен встановлюється у “SecurityContextHolder” та управління передається наступному фільтру. Код класу “JwtAuthenticationFilter” наведено в додатку А.

Загальна схема роботи Spring Security зі впровадженням фільтром зображена на схемі рисунку 3.4.

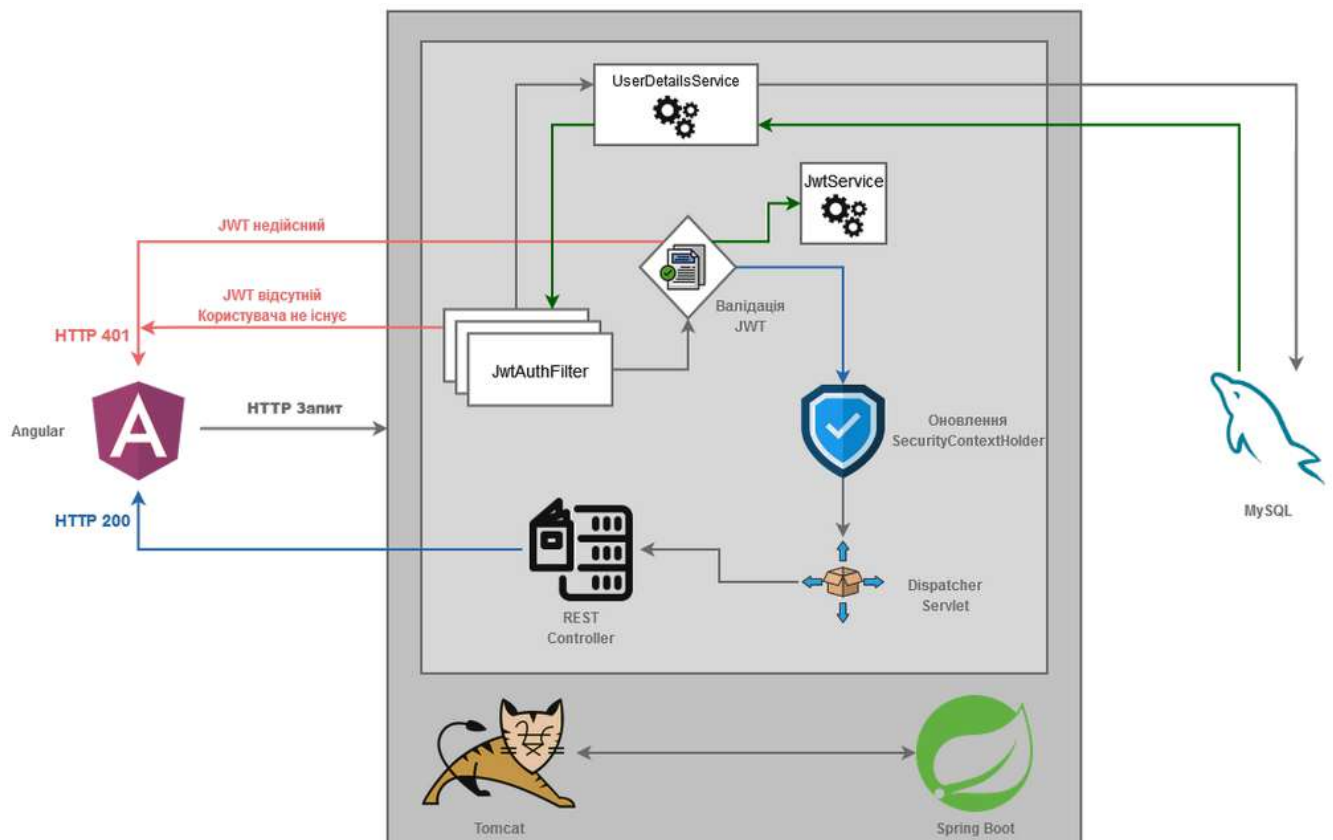


Рисунок 3.4 – Схема роботи Spring Security

Що стосується безпечного зберігання даних в базі даних то фреймворк надає такі інтерфейси як “PasswordEncoder” та “TextEncryptor”.

PasswordEncoder - це інтерфейс у Spring Security, що забезпечує механізм шифрування паролів. Основна його задача - хешування паролів перед збереженням у базу даних та перевірка паролів під час аутентифікації.

TextEncryptor - це інтерфейс у Spring Security, що забезпечує механізм шифрування та розшифрування текстових даних. Використовується для захисту конфіденційної інформації, яка зберігається у вигляді тексту.

Перед їх використанням слід створити біни як наведено в лістингу 3.23.

Лістинг 3.23 – Створення бінів “PasswordEncoder” та “TextEncryptor”

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public TextEncryptor textEncryptor() {
    return Encryptors.text(
        "1d598942b3b007c39ac690a1a727aa0196259ab6ee12cb20fefb91db19123951",
        "96ab0a75ecf23a58"
    );
}
```

Як наведено в лістинг PasswordEncoder використовуватиме алгоритм BCrypt для хешування паролів. BCryptPasswordEncoder широко рекомендований вибір для безпечного хешування паролів в Spring Security. Він відповідає за генерацію випадкових солей для кожного пароля. TextEncryptor в свою чергу створює текстовий шифратор, який використовує пароль та сіль для шифрування.

Тепер для шифрування паролю, перед доданням до бази даних, слід скористатися методом `passwordEncoder.encode(rawPassword)`, а при авторизація користувача використовувати метод `passwordEncoder.matches(CharSequence rawPassword, String encodedPassword)`, який порівнює наданий користувачем пароль з зашифрованим.

Для зберігання текстових даних, наприклад повідомлень користувачів, слід скористатися методом `textEncryptor.encrypt(text)`, а для розшифровки `textEncryptor.decrypt(encryptedText)`.

3.2 Розробка клієнтської частини

На даному етапі буде створено клієнтський інтерфейс для зручної взаємодії з системою.

Буде розглянуто приклади створення сервісів, які будуть надсилати запити до серверу та повертати від нього відповіді, створення компонентів, які будуть використовуватися для опису логіку взаємодії з користувацьким інтерфейсом, а також налаштування маршрутизації між компонентами.

3.2.1 Ініціалізація та налаштування Angular додатку

Для ініціалізації проекту потрібно ввести команду: `ng new angular-tour-of-heroes`. Після чого будуть встановлені усі необхідне пакети та залежності, які потрібні Angular.

Щоб спростити процес створення візуально приємного дизайну, було додано бібліотеку Angular Material. Для цього було введено команду: `ng add @angular/material`. Angular Material пропонує на вибір чотири теми, а також можливість створити власну, з яких було обрано “Azure & Blue”.

Процес розробки користувацького інтерфейсу також було спрощено за допомогою додання Tailwind – CSS фреймворк, який дозволяє стилізувати веб-сторінки, не виходячи з файлу HTML. Для цього було прописано команди: `npm install -D tailwindcss postcss autoprefixer` та `npm tailwindcss init`.

Для запуску Angular додатку треба прописати команду: `ng serve --open`. Після чого програма буде запущена використовуючи порт 4200 за замовченням і для того щоб відкрити її з браузера, слід звернутися за URL: `http://localhost:4200/`.

3.2.2 Створення клієнтських сервісів

У контексті Angular, сервіс - це об'єкт, який забезпечує спільну функціональність для різних частин додатку. Сервіси використовуються для організації та спільного використання логіки, яка не пов'язана безпосередньо з відображенням інтерфейсу користувача.

Розглянемо створення сервісу на прикладі взаємодії з сервером, пов'язаної з обмінами.

Для створення сервісу було прописано команду: `ng generate service exchange`. Після чого були створені файли `“exchange.service.ts”` та `“exchange.service.spec.ts”`, один для написання логіки, інший для тестування відповідно.

В лістингу 3.24 наведено методи для створення обміну та отримання ініційованих обмінів.

Лістинг 3.24 – Методи створеного сервісу

```
@Injectable({
  providedIn: 'root'
})
export class ExchangeService {

  private baseUrl = 'http://localhost:8080';

  constructor(private http:HttpClient) { }

  createNewExchange(bookId:number) {
    const headers = new HttpHeaders({
      Authorization:`Bearer ${localStorage.getItem("token")}`
    }).set('Content-Type','application/json');
    return this.http.post(this.baseUrl +
"/api/v1/exchange/create/" + bookId, null, {headers});
  }

  getAllInitiateExchanges() {
    const headers = new HttpHeaders({
      Authorization:`Bearer ${localStorage.getItem("token")}`
    }).set('Content-Type','application/json');
    return this.http.get<any>(this.baseUrl +
"/api/v1/exchange/get/initiation", {headers});
  }
}
```

Анотація `@Injectable` означає, що Angular автоматично надає цей сервіс в корені застосунку (singleton), тобто він буде створений один раз і доступний в усіх компонентах і сервісах, які його інжектують.

Поле `baseUrl` вміщує у собі базовий URL для серверного API, до якого будуть надходити запити. Поле методів `headers` являє собою HTTP-заголовки, які

включають токен авторизації типу Bearer, який було вилучено з локального сховища (що і буде використано на стороні серверу для ідентифікації користувача), та встановлюють формат JSON як тип тіла запиту.

Метод `createNewExchange` виконує метод POST для створення нового обміну, в той час як `getAllInitiateExchanges()` виконує метод GET для отримання всі ініційованих обмінів користувача.

REST методи PUT та DELETE будуть створюватися за таким самим принципом.

3.2.3 Створення компонентів клієнтського інтерфейсу

В контексті Angular, компонент є основним будівельним блоком додатку. Компонент відповідає за відображення певної частини користувацького інтерфейсу і управління відповідною логікою. Компоненти в Angular є незалежними та повторно використовуваними частинами коду, які можна комбінувати для створення складних додатків.

Розглянемо приклад створення компоненту, який буде відображати список обмінів та надавати інтерактивні елементи для керування ними.

Для створення компоненту було прописано команду: `ng generate component exchanges-component`. Після чого було створено чотири файли: `exchanges.component.ts` (файл класу компонента), `exchanges.component.html` (шаблон компонента), `exchanges.component.css` (стили компонента), `exchanges.component.spec.ts` (тестовий файл компонента). Створений компонент наведено в лістингу 3.25.

Лістинг 3.25 – Створений компонент

```
@Component({
  selector: 'app-exchanges',
  standalone: true,
  imports: [CommonModule, MatButtonModule, MatIconModule,
MatTabsModule, MatTableModule, RouterModule],
  templateUrl: './exchanges.component.html',
  styleUrls: ['./exchanges.component.scss']
})
export class ExchangesComponent implements OnInit {...}
```

Компонент реалізує інтерфейсу OnInit, що вимагає реалізації методу ngOnInit. Цей метод викликається Angular, коли компонент ініціалізується.

Декоратор @Component використовується для визначення метаданих компоненту:

- selector - визначає ім'я HTML-тега, який буде використовуватися для відображення цього компонента
- standalone - властивість вказує, що компонент є самостійним. Самостійні компоненти можуть бути визначені без включення в модуль.
- imports - масив модулів, які імпортуються і використовуються всередині компонента. Це дозволяє компоненту використовувати функціональність та компоненти з цих модулів. Наприклад, CommonModule надає директиви Angular (як ngIf та ngFor), а модулі Angular Material надають різні компоненти бібліотеки.
- templateUrl - шлях до файлу HTML-шаблону, який визначає структуру відображення компонента.
- styleUrls - шлях до файлу SCSS, який містить стилі для цього компонента.

В лістинг 3.26 наведено метод для отримання списку обмінів, що показує приклад взаємодії контролера з сервісами.

Лістинг 3.26 – Приклад взаємодії компонента з сервісом

```
getInitiateExchangesData() {
    this.exchangeService.getAllInitiateExchanges().subscribe({
        next: (response: SuccessResponse<Exchange[]>) => {
            this.initiateExchanges = response.data;
        },
        error: (error: ErrorResponse) => {
            if (error.error.error.message) {
                this.responseMessage = error.error.error.message;
            } else {
                this.responseMessage = "Unexpected error occurred";
            }
            this.snackbarService.openSnackBar(this.responseMessage,
"error");
        }
    });
}
```

Метод `subscribe` підписується на `getAllInitiateExchanges()` для обробки відповіді або помилки. Використовується об'єкт з двома методами: `next` для обробки успішної відповіді і `error` для обробки помилки. У разі успішної відповіді отримані обміни було додано у масив `initiateExchanges`, у разі помилки буде виведено повідомлення про це.

Тож тепер реалізація методу `ngOnInit` буде виглядати так як наведено в лістингу 3.27.

Лістинг 3.27 – Реалізація методу `ngOnInit`

```
ngOnInit(): void {
    if (this.authService.isAuthenticated()) {
        this.getInitiateExchangesData();
        this.getRecipientExchangesData();
    } else {
        this.router.navigate(['/auth']);
    }
}
```

Метод ініціалізує масиви зі списками ініційованих обмінів та обмінів в який користувач став одержувачем обміну, у разі якщо користувач автентифікований (тобто в локальному сховищі вже є збережений токен), у іншому випадку його буде направлено до компоненту автентифікації, для цього було використано `router`, про що буде далі.

3.2.4 Налаштування маршрутизації між компонентами

Маршрутизація в Angular забезпечує навігацію між різними компонентами додатку. Вона дозволяє користувачам переміщуватися між різними представленнями або сторінками додатку, використовуючи URL-адреси. Angular Router є модулем, який керує маршрутизацією. `Routes` масив об'єктів маршрутів, які визначають шляхи та відповідні їм компоненти. `RouterOutlet` директива, яка визначає місце на сторінці, де буде відображатися обраний компонент. `RouterLink` директива, яка використовується для створення посилань для навігації між маршрутами.

В лістинг 3.28 наведено налаштування маршрутизації між компонентами розробленого додатку.

Лістинг 3.28 – Налаштування маршрутизації

```
export const routes: Routes = [
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  { path: 'home', component: HomeComponent },
  { path: 'auth', component: AuthComponent },
  { path: 'search', component: SearchComponent },
  { path: 'library', component: LibraryComponent },
  { path: 'wishlist', component: WishlistComponent },
  { path: 'exchanges', component: ExchangesComponent },
  { path: 'chatting', component: ChatComponent },
  { path: 'profile', component: UserProfileComponent },
  { path: 'user/:id/profile', component: UserProfileComponent },
},
  { path: 'book/:id', component: BookOverviewComponent },
  { path: 'book/:id/location', component:
BookLocationHistoryComponent },
  { path: 'new-password', component: NewPasswordFormComponent },
},
  { path: '**', component: PageNotFoundComponent }
];
```

Так, якщо звернуться з URL “http://localhost:4200/” Angular автоматично відкриє компонент HomeComponent. Якщо буде здійснено звернення за URL, який не було налаштовано, то буде відкрито компонент PageNotFoundComponent. В головному шаблоні додатку буде використано RouterOutlet, як наведено в лістингу 3.29.

Лістинг 3.29 – HTML шаблон головного компоненту додатку

```
<div>
  <div>
    <app-navbar></app-navbar>
    <router-outlet></router-outlet>
  </div>
  <app-footer></app-footer>
</div>
```

Так головний контент постійно буде змінюватися, а компоненти інтерфейсу для навігації та компонент `footer` будуть лишатися на місці. Щоб перевести користувача на інший компонент потрібно скористатися модулем `Route` за прикладом: `this.router.navigate(['/auth'])`. Для додання посилання на компонент в HTML тезі `<a></a>`, слід скористатися `RouterLink` за прикладом: `<a [routerLink]='[/book]'></a>`.

На цьому можна зробити висновок, що налаштування навігації в Angular є доволі простим завданням.

4 ВИПРОБОВУВАННЯ СИСТЕМИ

4.1 Unit тестування

Unit тестування (від англ. "unit testing") – це процес перевірки окремих компонентів програмного забезпечення для визначення їх коректності. Кожен компонент, або "юніт", зазвичай є найменшою тестованою частиною програми, наприклад, функція, метод або клас.

Unit тести дозволяють виявити помилки на ранніх етапах розробки. Це дозволяє швидко знайти і виправити проблеми до того, як вони вплинуть на інші частини системи. Наявність таких тестів допомагає впевнено вносити зміни до коду. Якщо зміна викликає збій у тестах, це сигналізує про те, що щось пішло не так, і потрібно переглянути зміни. Unit тести можуть слугувати додатковою документацією для коду. Вони показують, як повинна працювати кожна окрема частина системи.

4.1.1 Тестування репозиторіїв

В загалом основні методи репозиторіїв, які тестують є ті, які були впроваджені розробником.

Класова анотація `@DataJpaTest` є спеціалізованою анотацією, що надається Spring Boot для тестування JPA репозиторіїв. Вона спрощує налаштування

середовища для тестування доступу до даних, фокусуючись на компонентах, які працюють з JPA.

В тестах використовується також `@BeforeEach`, яка буде виконувати збереження даних в базі даних H2 перед кожним тестом, а також `@AfterEach`, яка буде видаляти усі дані з бази після його виконання.

В лістингу 4.1 наведено код тестування створених методів репозиторію сутності «Обміни».

Лістинг 4.1 – Тестування методів репозиторію “ExchangeRepository”

```
@Test
void foundAllFirstUserInitiations() {
    List<Exchange> exchanges =
exchangeRepository.findAllByInitiatorId(firstUser.getId());

    assertThat(exchanges.size()).isEqualTo(1);

    exchanges.forEach(exchange ->
assertThat(exchange.getInitiator().getId()).isEqualTo(firstUser
.getId()));
}

@Test
void foundAllSecondUserRecipientExchanges() {
    List<Exchange> exchanges =
exchangeRepository.findAllByRecipientId(secondUser.getId());

    assertThat(exchanges.size()).isEqualTo(2);

    exchanges.forEach(exchange ->
assertThat(exchange.getRecipient().getId()).isEqualTo(secondUse
r.getId()));
}

@Test
void foundAllExchangesByBookId() {
    List<Exchange> exchanges =
exchangeRepository.findAllByBookId(secondBook.getId());
    assertThat(exchanges.size()).isEqualTo(2);

    exchanges.forEach(exchange ->
assertThat(exchange.getBook().getId()).isEqualTo(secondBook.get
Id()));
}
```

Тести перевіряють, що повернений список відповідає вимогам.

4.1.2 Тестування сервісів

Тестування сервісів відбувається за допомогою Mockito - фреймворку для створення тестових дублікатів (mock-об'єктів) у Java.

`@ExtendWith(MockitoExtension.class)` анотація вказує JUnit використовувати розширення Mockito для керування життєвим циклом Mockito. Вона забезпечує автоматичну ініціалізацію мок-об'єктів, а також обробку анотацій `@Mock` та `@InjectMocks`.

Клас для тестування сервісу для обробки обмінів наведено в лістингу 4.2.

Лістинг 4.2 – Клас “ExchangeServiceTest”

```
@ExtendWith(MockitoExtension.class)
class ExchangeServiceTest {
    @Mock
    private ExchangeRepository exchangeRepository;
    @Mock
    private BookService bookService;
    @InjectMocks
    private ExchangeService exchangeService;
}
```

Анотація `@Mock` вказує що клас є мок-об'єктом, який імітуватиме поведінку реального репозиторію (`ExchangeRepository`), або сервісу (`ExchangeService`), дозволяючи тестувати `ExchangeService` у ізоляції від реальної бази даних та контролювати виклики та результати сервісів.

Анотація `@InjectMocks` вказує Mockito автоматично створити екземпляр `ExchangeService` та інжектувати у нього всі об'єкти, позначені анотацією `@Mock`.

В лістингу 4.3 наведено впроваджений метод для тестування створення нового обміну.

Лістинг 4.3 – Тестування створення обміну

```
@Test
void newExchangeCreated() {
    User bookOwner = User.builder().id(1L).build();
    User initiator = User.builder().id(2L).build();
    Book book =
    Book.builder().id(4L).owner(bookOwner).build();
```

```

        exchangeService.createNewExchange(initiator, book);
        ArgumentCaptor<Exchange> exchangeArgumentCaptor =
ArgumentCaptor.forClass(Exchange.class);

verify(exchangeRepository).save(exchangeArgumentCaptor.capture());
        Exchange capturedExchange =
exchangeArgumentCaptor.getValue();

assertThat(capturedExchange.getInitiator()).isEqualTo(initiator);

assertThat(capturedExchange.getRecipient()).isEqualTo(bookOwner);
        assertThat(capturedExchange.getBook()).isEqualTo(book);
        assertThat(capturedExchange.getConfirmed()).isFalse();
    }

```

Створюються об'єкти User для власника книги (bookOwner) та ініціатора обміну (initiator), а також об'єкт Book, який належить власнику книги. Викликається метод createNewExchange на exchangeService з переданими ініціатором та книгою. Також використовується ArgumentCaptor для перехоплення об'єкта Exchange, збереженого у exchangeRepository. Далі виконуються перевірки, щоб переконатися, що збережений об'єкт Exchange має очікувані значення: ініціатором є initiator, отримувачем є bookOwner, книгою є book, а підтвердження (confirmed) має значення false.

4.2 Перевірка роботи системи

Розроблений додаток має багатий функціонал, але одним з головних є пошук та обмін книгами.

Для створення обміну користувачу слід знайти потрібну книгу, відкрити її профіль і натиснути на кнопку “Exchange” після чого буде створено новий обмін в якому ініціатором стане поточний користувач, який і натиснув кнопку “Exchange”.

Фрагмент сторінки пошуку книги зображена на рисунку 4.1.

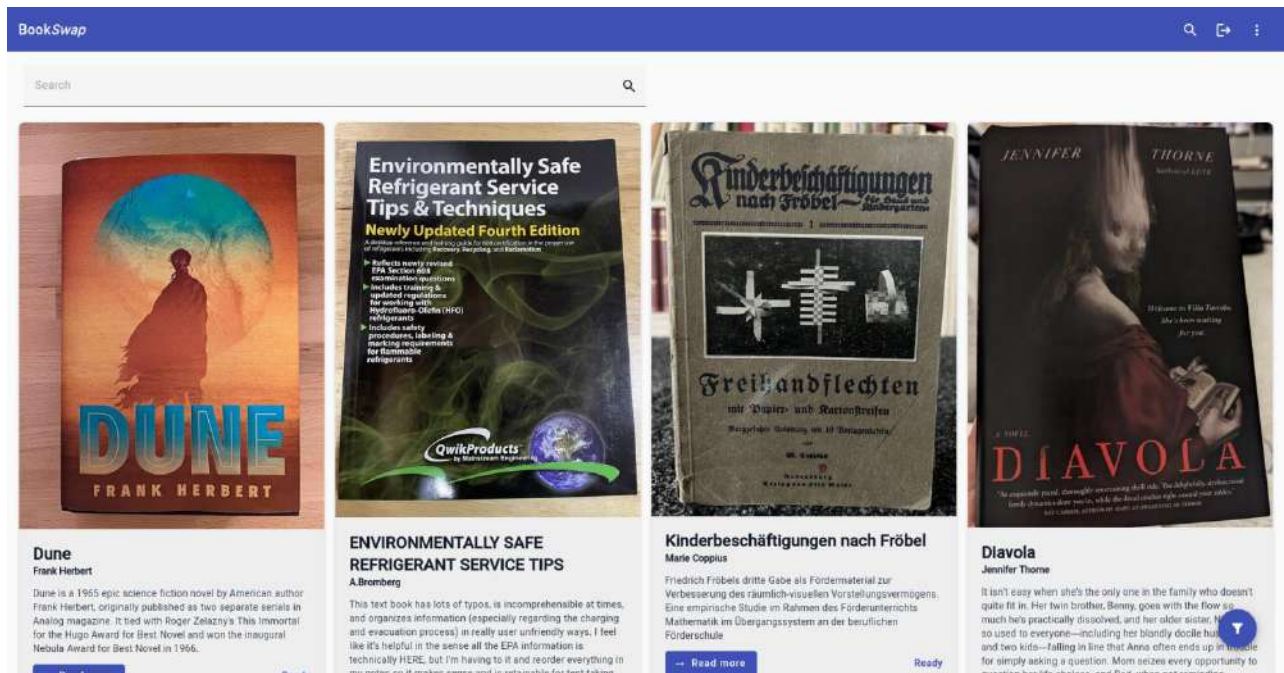


Рисунок 4.1 – Фрагмент сторінки для пошуку книги

Побачивши потрібну книгу користувач може натиснути по зображенню, або по кнопці “Read more” після чого відкриється сторінка профілю книги, фрагмент якої зображено на рисунку 4.2.

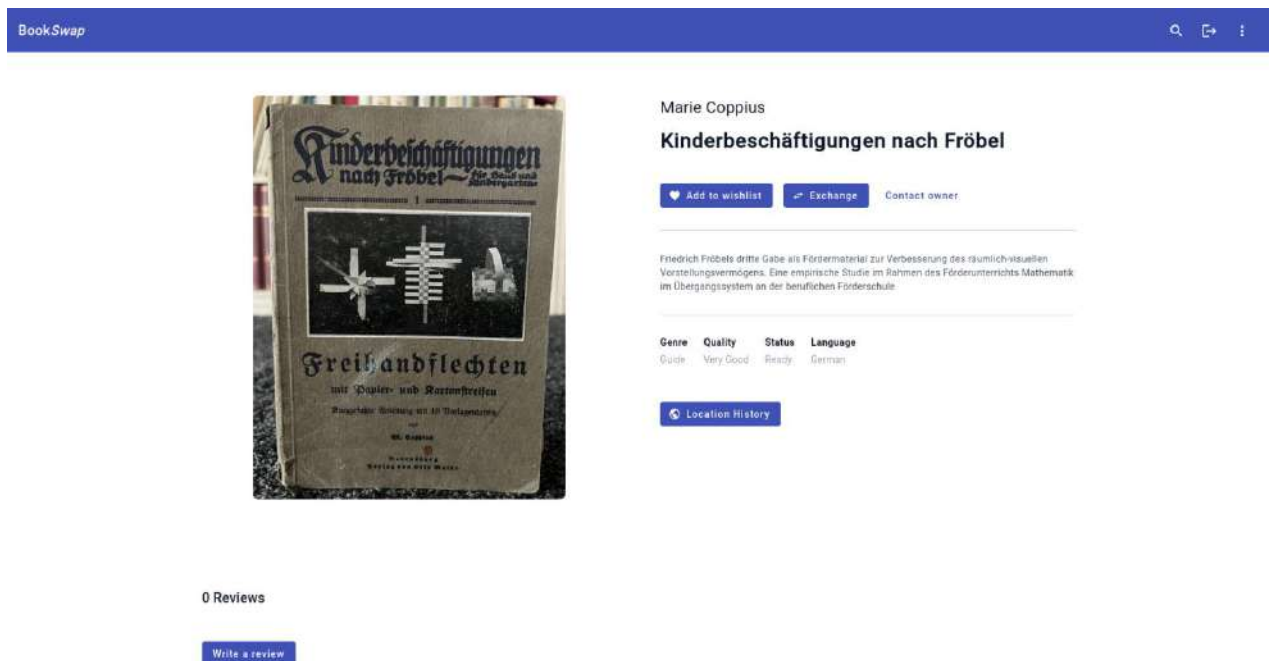
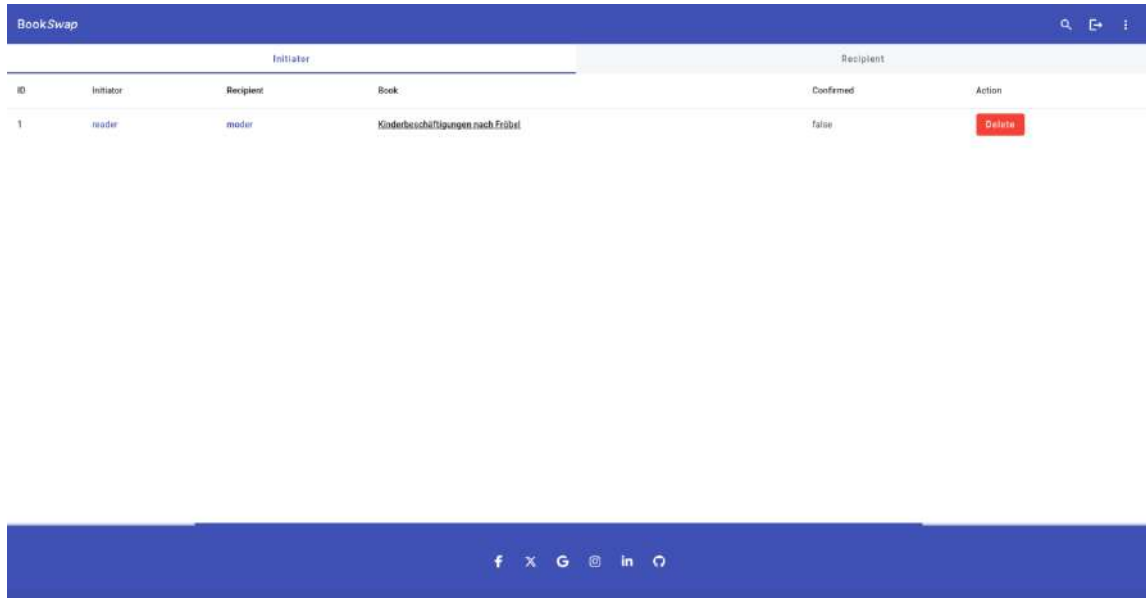


Рисунок 4.2 – Фрагмент сторінки профілю книги

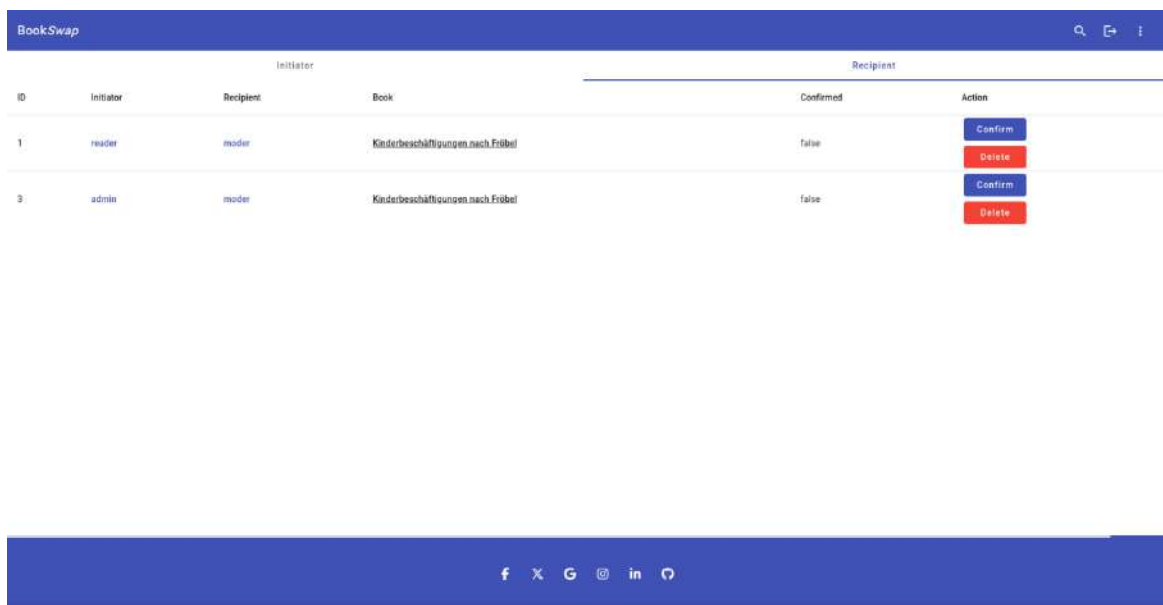
Тепер користувач має натиснути кнопку “Exchange” та перейти на сторінку “Exchanges”, де в таблиці “Initiator”, як зображено на рисунку 4.3, буде виведено створений обмін.



BookSwap					
Initiator				Recipient	
ID	Initiator	Recipient	Book	Confirmed	Action
1	reader	moder	Kinderbeschäftigungen nach Fröbel	false	Delete

Рисунок 4.3 – Фрагмент сторінки з обмінами користувача

Ввійдемо в інший обліковий запис користувача, який був вказаний як отримувач обміну, перейдемо на сторінку “Exchange” та відкриємо вкладку “Recipient” (рисунок 4.4).



BookSwap					
Initiator				Recipient	
ID	Initiator	Recipient	Book	Confirmed	Action
1	reader	moder	Kinderbeschäftigungen nach Fröbel	false	Confirm Delete
3	admin	moder	Kinderbeschäftigungen nach Fröbel	false	Confirm Delete

Рисунок 4.4 – Фрагмент сторінки з обмінами іншого користувача

Бачимо, що отримані обміни можна підтвердити, після чого книга змінить свого власника, або видалити. При спробі підтвердження обміну, система впевниться чи справді користувач хоче це зробити (рисунок 4.5).

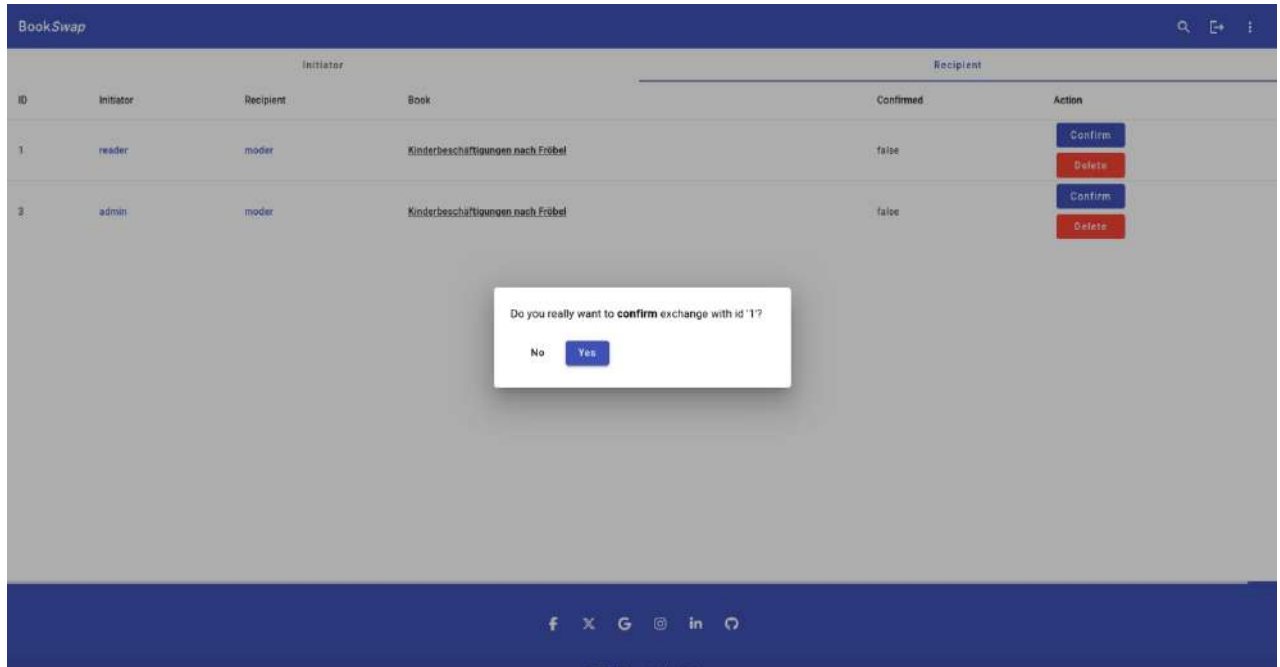


Рисунок 4.5 – Запит на підтвердження обміну

Результат підтвердження обміну зображено на рисунку 4.6.

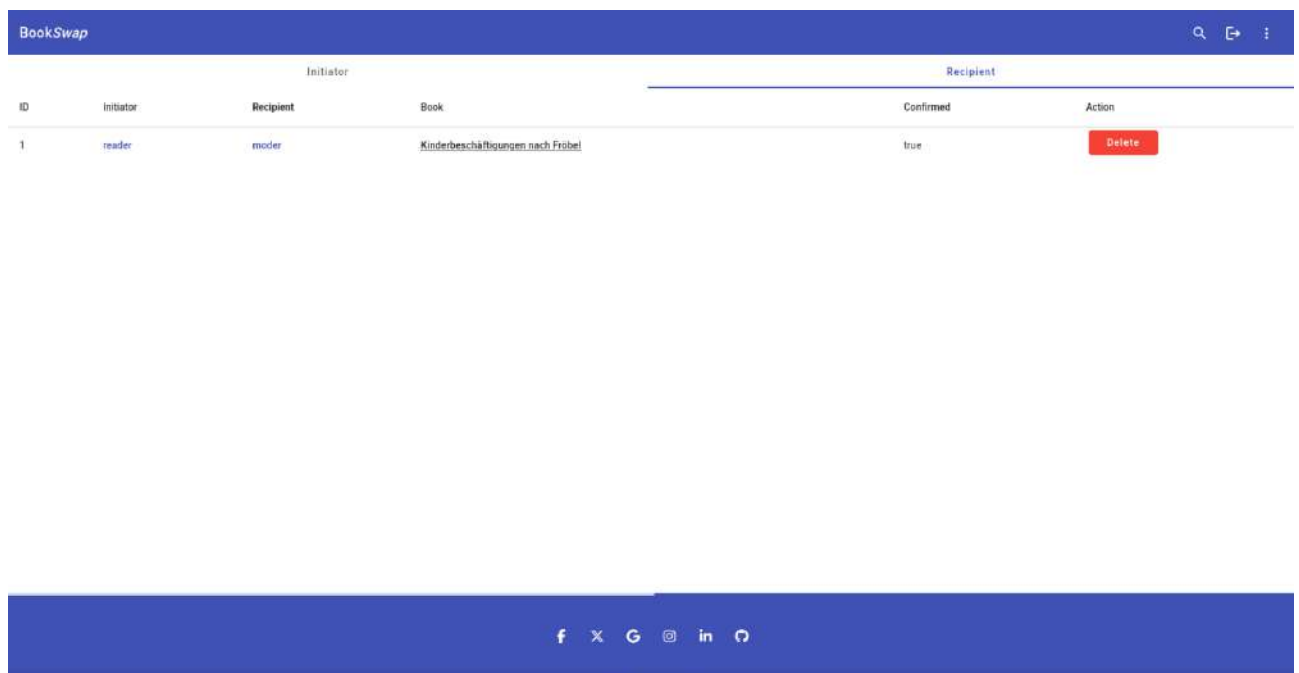


Рисунок 4.6 – Результат підтвердження обміну

Бачимо, що залишився тільки один обмін, який було підтверджено, інший обмін змінив отримувача на ініціатора вже підтвердженого поточного обміну. В цьому можна переконатись знову змінивши обліковий запис на попереднього користувача з якого було здійснено ініціацію обміну (рисунок 4.7).

Initiator				Recipient	
ID	Initiator	Recipient	Book	Confirmed	Action
3	admin	reader	Kinderbeschäftigungen nach Fröbel	false	Confirm Delete

Рисунок 4.7 – Результат зміни отримувача обміну

Тепер користувач може видалити обмін, або так само підтвердити його, і книга знову змінить власника.

ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено розподілену комп'ютерну систему для обміну книг, яка включає в себе сервер для обробки та відповіді на запити клієнтів, а також користувацький інтерфейс для спрощення взаємодії з системою. Головною метою роботи було створити комп'ютерну систему, що дозволить спростити комунікацію між користувачами, а також надати зручні механізми для сприяння процесу обміну книг.

Для досягнення цієї мети було використано різні технології та інструменти розробки. Серверною мовою програмування було використано Java, що забезпечить надійність та стійкість системи. Для прискорення та спрощення розробки було використано фреймворк Spring Boot, що також забезпечить розширюваність системи та її безпеку. Також для реалізації користувацького інтерфейсу було використано фреймворк Angular, який забезпечив багатий функціонал та велику кількість інструментів при розробці інтерфейсу. У випадку з Angular мовою програмування клієнтської частини став TypeScript.

В процесі розробки було спроектовано базу даних додатку, системою керування якої стало MySQL.

Розроблена система, окрім надання можливості для обміну книг, надає багатий функціонал в який входить: пошук книг за різними атрибутами, керування власними книгами у бібліотеці, написання рецензій та повідомлень іншим користувач, редагування списку бажаного. Саме це зробить її конкурентоспроможною.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. О.С Бунке, Серверні Web-Технології: Київ КПІ ім. Ігоря Сікорського 2023. 109с.;
2. Seyed M.M. “Saied” Tahaghoghi and Hugh E. Williams, Learning MySQL: © 2007 O’Reilly Media. All rights reserved. Printed in the United States of America. 600p.;
3. MySQL Documentation. MySQL. URL: <https://dev.mysql.com/doc/> (дата звернення: 20.04.2024);
4. Regina O. Obe and Leo S. Hsu, PostgreSQL: Up and Running, Second Edition: © 2015 Regina Obe and Leo Hsu. All rights reserved. Printed in the United States of America. 211p.;
5. PostgreSQL Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/> (дата звернення: 20.04.2024);
6. Josiah L. Carlson, Redis in Action: ©2013 by Manning Publications Co. All rights reserved. 294p.;
7. Kristina Chodorow, MongoDB: The Definitive Guide: Copyright © 2013 Kristina Chodorow. All rights reserved. Printed in the United States of America. 409p.;
8. What is Java? Java. URL: https://www.java.com/en/download/help/whatis_java.html (дата звернення: 20.04.2024);
9. Spring Framework Documentation. Spring By VMware Tanzu. URL: <https://spring.io/projects/spring-framework> (дата звернення: 20.04.2024);
10. A tour of the C# language. Dotnet Microsoft. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/> (дата звернення: 20.04.2024);
11. What is ASP.NET? Dotnet Microsoft. URL: <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet/> (дата звернення: 20.04.2024);

12. Python Documentation. Python Software Foundation. URL: <https://www.python.org/doc/> (дата звернення: 20.04.2024);
13. Django Documentation. Django Software Foundation. URL: <https://docs.djangoproject.com/en/5.0/> (дата звернення: 20.04.2024);
14. Patrick Mulder and Kelsey Breseman, Node.js for Embedded Systems: Copyright © 2017 Patrick Mulder and Kelsey Breseman. All rights reserved. 200p.;
15. Shyam Seshadri and Brad Green, AngularJS: Up And Running: Copyright © 2014 Shyam Seshadri and Brad Green. All rights reserved. Printed in the United States of America. 282p.;
16. Angular Documentation. Angular. URL: <https://angular.io/docs> (дата звернення: 20.04.2024);
17. Azat Mardan, React Quickly: ©2017 by Manning Publications Co. All rights reserved. 496p.

ДОДАТОК А

КОД JWT СЕРВІСУ ТА ФІЛЬТРУ

Лістинг В.1 – Код JWT сервісу

```

@Service
public class JwtService {

    private static final byte[] SECRET_KEY =
Decoders.BASE64.decode("1d590942b3b007c79ac697a1a727aa0196239ab
6ee12cb20fefb91db19113951");

    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);
    }

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

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

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

    public boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
}

```

```

    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

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

```

Лістинг В.2 – Код JWT фільтру

```

@Component
public class JwtAuthenticationFilter extends
OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final TokenRepository tokenRepository;

    public JwtAuthenticationFilter(JwtService jwtService,
UserDetailsService userDetailsService, TokenRepository
tokenRepository) {
        this.jwtService = jwtService;
        this.userDetailsService = userDetailsService;
        this.tokenRepository = tokenRepository;
    }

    @Override
    protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain) throws
ServletException, IOException {

        final String authHeader =
request.getHeader("Authorization");
        final String jwt;
        final String email;

        if (authHeader == null ||
!authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        try {

```

```

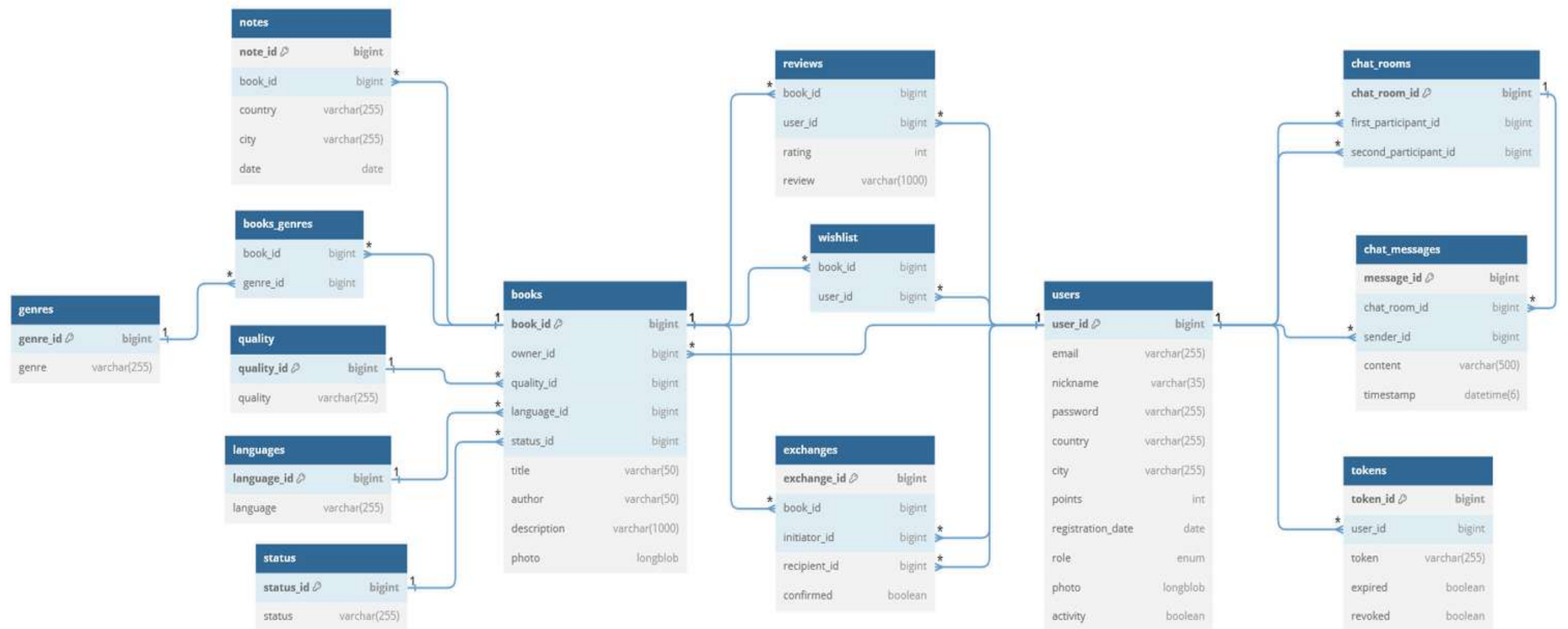
        jwt = authHeader.substring(7);
        email = jwtService.extractEmail(jwt);

        if (email != null &&
SecurityContextHolder.getContext().getAuthentication() == null)
        {
            UserDetails userDetails =
this.userDetailsService.loadUserByUsername(email);
            boolean isDbTokenValid =
tokenRepository.findByToken(jwt)
                .map(t -> !t.isExpired() &&
!t.isRevoked())
                .orElse(false);
            if (jwtService.isTokenValid(jwt, userDetails)
&& isDbTokenValid) {
                UsernamePasswordAuthenticationToken
authToken = new UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities()
                );
                authToken.setDetails(
                    new
WebAuthenticationDetailsSource().buildDetails(request)
                );
SecurityContextHolder.getContext().setAuthentication(authToken)
;
            }
        }
    } catch (ExpiredJwtException ex) {
        response.setContentType("application/json");

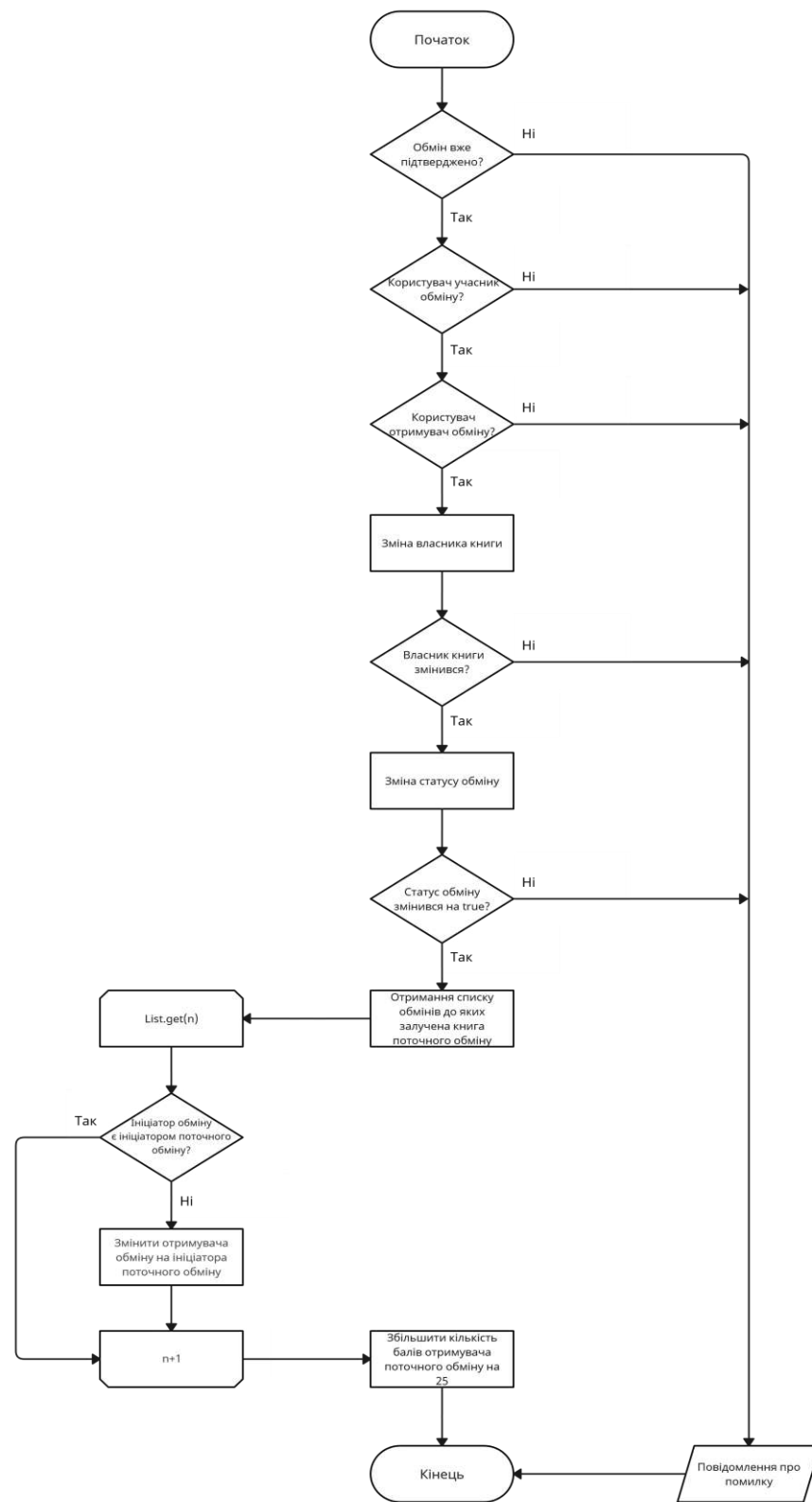
response.setStatus(HttpStatus.UNAUTHORIZED.value());
        ObjectMapper objectMapper = new ObjectMapper();
        ErrorDescription errorDescription = new
ErrorDescription(
            HttpStatus.UNAUTHORIZED.value(),
            "Session has expired",
            null
        );
        objectMapper.writeValue(response.getWriter(), new
ErrorResponse(errorDescription));
        return;
    }

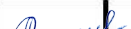
    filterChain.doFilter(request, response);
}
}

```



					13.02070849.00050 ПЛ1										
					Розробка комп'ютерної системи для обміну книг					Лит.		Маса	Масшт.		
Зам.	Лист.	№ докум.	Підп.	Дата											
Розроб.		Соломка А.С.													
Перев.		Скрупський С.Ю.													
Т.контр.															
Н.контр.		Польська О.В.			ER-модель бази даних					НУ «Запорізька політехніка»					
Затв.		Кудерметов Р.К.								КНТ-520					



					13.02070849.00050 ПЛ2								
					Розробка комп'ютерної системи для обміну книг				Лит.		Маса	Масшт.	
Зам.	Лист.	№ докум.	Підп.	Дата									
Розроб.		Соломка А.С.											
Перев.		Скрупський С.Ю.											
Т.контр.					Алгоритм підтвердження обміну				Лист 1		Листів 1		
Н.контр.		Польська О.В.											
Затв.		Кудерметов Р.К.											
									НУ «Запорізька політехніка» КНТ-520				

BookSwap

Search

Dune

Frank Herbert

Dune is a 1965 epic science fiction novel by American author Frank Herbert, originally published as two separate serials in Analog magazine. It tied with Roger Zelazny's This Immortal for the Hugo Award for Best Novel and won the inaugural Nebula Award for Best Novel in 1966.

Read more

Ready

Environmentally Safe Refrigerant Service Tips & Techniques

Newly Updated Fourth Edition

A. Bromberg

This text book has lots of typos, is incomprehensible at times, and organizes information (especially regarding the charging and evacuation process) in really user unfriendly ways. I feel like it's helpful in the sense all the EPA information is technically HERE, but I'm having to it and reword everything in my notes so it makes sense and is retainable for test taking.

Read more

Buy

Kinderbeschäftigungen nach Fröbel

Marie Coppius

Friedrich Fröbels dritte Gabe als Fördermaterial zur Verbesserung des räumlich-visuellen Vorstellungsvermögens. Eine empirische Studie im Rahmen des Förderunterrichts Mathematik im Übergangssystem an der beruflichen Förderschule

Read more

Ready

Diavola

Jennifer Thorne

It isn't easy when she's the only one in the family who doesn't quite fit in. Her twin brother, Benny, goes with the flow so much he's practically dissolved, and her older sister, N, so used to everyone—including her blandly docile half and two kids—falling in line that Anna often ends up in trouble for simply asking a question. Mom seizes every opportunity to question her life choices, and Dad, when not reminding everyone who paid for this vacation, just wants some peace and quiet.

Read more

Ready

Joyride

Stephen Crye

Joyride by Stephen Crye is a classic slasher novel in tradition with films like House On Sorority Row, Scream, The Blaming and more. The novel is fast-paced and has numerous characters dying in disturbing, graphic and bloody ways.

Read more

Not Ready

Unlocking Butlerstown

Dr. Eddy Fitzgerald

UNLOCKING BUTLERSTOWN edited by Dr. Eddy Fitzgerald. Published in 2016 by Butlerstown History Group. Large format, card covers. Illustrated throughout. 125 pages. Local history aspects of Butlerstown, Co. Waterford and its vicinity.

Read more

Not Ready

Ernest the Fierce Mouse

Philip Rowe

When Little Brown Mouse is left behind when the harvesters destroy her home, Hoppy Hare and his friends disguise her as a flower so she can find her way to where the harvest mice spend the winter

Read more

Ready

When Lights Burn Low

Helen L. Toner

A book that has been read but is in good condition. Very minimal damage to the cover including scuff marks, but no holes or tears. The dust jacket for hard covers may not be included. Binding has minimal wear. The majority of pages are undamaged with minimal creasing or tearing, minimal pencil underlining of text, no highlighting of text, no writing in margins. No missing pages.

Read more

Ready

In the Land of Ju-Ju

Robert Leighton

Here is a story which I dedicate to you in the hope that it may teach you something of the difficulties and perils that are faced by our fellow-countrymen in the work of civilising and opening up commerce with the native races of West Africa, and in their efforts to abolish the worship of Ju-Ju and the heath practice of human sacrifice.

Read more

Ready

Some Curious Plants

Duncan, F.M. & L.T.

Concentrating on out of print books and modern non-fiction. Photographs of all books are provided where possible, if not showing in the listings please contact me and I will be happy to email copies. All books are posted wrapped in bubble wrap and cardboard, with additional jiffy bag, by 1st class mail where possible. I am happy to quote additional insurance if required. Overseas books will be sent by standard airmail, I regret that I do not post surface mail.

Read more

Ready

Items per page: 101 - 10 of 12

f x G @ in

© 2024 Copyright: Art Solo

BookSwap

Search

Marie Coppius

Kinderbeschäftigungen nach Fröbel

Add to wishlist

Exchange

Contact owner

Friedrich Fröbels dritte Gabe als Fördermaterial zur Verbesserung des räumlich-visuellen Vorstellungsvermögens. Eine empirische Studie im Rahmen des Förderunterrichts Mathematik im Übergangssystem an der beruflichen Förderschule

Genre

Guide

Quality

Very Good

Status

Ready

Language

German

Location History

0 Reviews

Write a review

There are no reviews, but you can be the first!

BookSwap

Initiator

Recipient

ID	Initiator	Recipient	Book	Confirmed	Action
1	reader	moder	Kinderbeschäftigungen nach Fröbel	false	Confirm Delete
3	admin	moder	Kinderbeschäftigungen nach Fröbel	false	Confirm Delete

f x G @ in

© 2024 Copyright: Art Solo

						13.02070849.00050 ПЛЗ		
Зам.	Лист.	№ докум.	Підп.	Дата	Розробка комп'ютерної системи для обміну книг	Лист.	Маса	Масшт.
Розроб.	Перев.	Соломка А.С.	Скруський С.Ю.					
Т.контр.								
Н.контр.	Польська О.В.	Кудерметов Р.К.			Сторінки сайту	НУ «Запорізька політехніка» КНТ-520		