Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Шидлюх Костянтин Олександрович,
студент групи АІ-232

КУРСОВА РОБОТА

Дисципліна:
«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

Укладачі:
М.А. Годовиченко

Одеса – 2025

ЗМІСТ

ВСТУП 3

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 5

2.1 Основні бізнес-об’єкти ..... **Error! Bookmark not defined.**

2.2 Бізнес-сценарії використання 9

2.3 Обґрунтування структури моделі 10

2.4 Особливості предметної області 10

ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ Error! Bookmark not defined.

3.1 Структура даних (модель сутностей) ..... **Error! Bookmark not defined.**2

3.2 Архітектура застосунку ..... **Error! Bookmark not defined.**4

3.3 Організація пакетів ..... **Error! Bookmark not defined.**7

3.4 REST API 178

3.5 Універсальність архітектури ..... **Error! Bookmark not defined.**20

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ..... **Error! Bookmark not defined.**22

4.1 Моделі (Model-класи) ..... **Error! Bookmark not defined.**22

4.2 DTO (Data Transfer Objects) ..... **Error! Bookmark not defined.**3

4.3 Репозиторії ..... **Error! Bookmark not defined.**4

4.4 Сервіси ..... **Error! Bookmark not defined.**5

4.5 Контролери ..... **Error! Bookmark not defined.**6

4.6 Реалізація реєстрації та автентифікації користувача .... **Error! Bookmark not defined.**8

ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ ..... **Error! Bookmark not defined.**9

ЗАГАЛЬНІ ВИСНОВКИ ..... **Error! Bookmark not defined.**41

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... **Error! Bookmark not defined.**3

ВСТУП

У сучасному інформаційному суспільстві бібліотеки залишаються ключовими центрами доступу до знань та культури, поєднуючи традиційні паперові фонди з сучасними цифровими колекціями. Зі збільшенням обсягів літератури та різноманіттям форматів (електронні книги, аудіокниги, медіатека) виникає потреба у єдиній платформі для обліку матеріалів та взаємодії з читачами. Автоматизована система управління бібліотекою не лише відстежує наявність книжок і історію видач, але й дозволяє аналізувати популярність видань, оптимізувати логістику поповнення фонду, створювати персоналізовані рекомендації та забезпечувати безперервний доступ до ресурсів навіть у режимі карантинних обмежень.

Актуальність розробки веб-системи управління бібліотекою зумовлена такими чинниками:

– Збільшенням кількості фондів та читачів у бібліотеках різного масштабу;

– Необхідністю прозорого обліку видачі та повернення книг у режимі реального часу;

– Потребою в єдиному інтерфейсі для бібліотекарів та користувачів з різними правами доступу;

– Використанням сучасних веб-технологій для віддаленого доступу з будь-якого пристрою.

Мета курсової роботи – розробити веб-систему управління бібліотекою, яка реалізує функціонал:

– зберігання та редагування даних про книги, авторів, жанри та читачів;

– видачі та обліку повернення книг;

– реєстрації та автентифікації користувачів з ролями "LIBRARIAN" та "READER";

– захищеного доступу через JWT та OAuth2;

– тестування й документування API.

Використані технології та інструменти: У роботі застосовано Java 17 та Spring Boot для реалізації серверної логіки, Spring Data JPA і Hibernate для керування даними в PostgreSQL, аутентифікацію і авторизацію через Spring Security із JWT та OAuth2, а також OpenAPI/Swagger UI для інтуїтивної документації REST API. Для тестування сервісів використовувалися Postman і JUnit із підтримкою Spring Security Test, а розгортання – на Render.com.

Основні завдання курсової роботи: Розробити інтуїтивний REST API для обліку книг, авторів, жанрів і читачів із повним CRUD-функціоналом; забезпечити видачу та повернення книг з обліком історії позик; впровадити безпечний доступ із ролями LIBRARIAN і READER за допомогою JWT та можливістю OAuth2-логіну; документувати інтерфейси в Swagger UI та покрити ключові сценарії тестами.

Таким чином, система надає бібліотекарям і читачам зручний інструмент для оперативного обліку, а розробнику – ґрунтовні навички створення сучасних безпечних веб-додатків на Java.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Предметною областю дослідження в даній курсовій роботі є процеси обліку бібліотечного фонду, читачів та фактів видачі книг у сучасній бібліотеці. Основна мета системи — автоматизація бібліотечної діяльності, що включає реєстрацію книг, авторів, читачів, ведення історії видачі та повернення книг, а також класифікацію книжок за жанрами.

Бібліотека традиційно оперує значним обсягом інформації, пов’язаної з літературними ресурсами та їх використанням читачами. Ручний облік значно ускладнює пошук книг, контроль повернення, формування звітів тощо. Тому розробка автоматизованої системи дозволяє покращити обслуговування читачів, оптимізувати ресурси та забезпечити прозорість обліку.

Розроблена система повинна дозволяти:

− реєструвати книги з зазначенням назви, ISBN, року видання, автора та жанру;

− реєструвати авторів, читачів і ведення їхніх даних;

− видавати книги читачам і фіксувати дату видачі та повернення;

− формувати звіти по історії видачі книг та діяльності читачів;

− реалізовувати контроль доступу до функцій системи (читач / бібліотекар);

− організовувати книги за жанрами для спрощення навігації та пошуку.

2.1 Основні бізнес-об’єкти

1) Книга (Book)

Центральний об'єкт системи, що представляє літературне видання. Містить такі атрибути:

− назва (title);

− ISBN (isbn);

− рік видання (publishedYear);

− автор (author);

− жанр (genre).

Одна книга може бути багаторазово видана різним читачам у різні моменти часу.

Зв’язки:

− багато книг може мати один автор (ManyToOne);

− одна книга може належати до одного жанру (ManyToOne);

− одна книга може мати багато записів про видачу (OneToMany з Loan).

2) Автор (Author)

Представляє творця одного або декількох літературних творів. Має:

− ім’я (name);

− дату народження (birthDate).

Автор може бути пов’язаний з багатьма книгами (OneToMany).

3) Читач (Reader)

Особа, яка користується послугами бібліотеки. Атрибути:

− ім’я та прізвище (firstName, lastName);

− номер читацького квитка (libraryCardNumber);

− дата реєстрації (registeredAt).

Читач може одночасно мати кілька книг на руках, а також історію попередніх видач.

Зв’язки:

− один читач може мати багато видач (OneToMany з Loan).

4) Видача книги (Loan)

Описує факт видачі певної книги певному читачеві:

− книга (book);

− читач (reader);

− дата видачі (loanDate);

− дата повернення (returnDate, може бути null, якщо книга ще не повернена).

Цей об’єкт є логом транзакцій між книгами та читачами й використовується для формування статистики, нагадувань і контролю.

5) Жанр (Genre)

Категорія, що дозволяє групувати книги за тематикою:

− назва жанру (name).

Кожна книга має один жанр, однак один жанр може охоплювати багато книг (OneToMany).

2.2 Бізнес-сценарії використання

1) Додавання нової книги

Бібліотекар заходить у веб-інтерфейс, відкриває форму додавання книги, вводить назву, ISBN, рік видання, вибирає автора та жанр зі списку. Після збереження книга з’являється у загальному каталозі.

2) Реєстрація читача

Бібліотекар створює новий обліковий запис читача, вказуючи ім’я, прізвище, номер читацького квитка та дату реєстрації. Після цього читач може брати книги в користування.

3) Видача книги

Бібліотекар обирає читача та книгу, яка є доступною для видачі. Система фіксує дату видачі та додає запис до історії видач. Якщо книга вже на руках у когось, вона тимчасово недоступна для інших читачів.

4) Повернення книги

Коли читач повертає книгу, бібліотекар знаходить відповідну видачу й встановлює дату повернення. Система оновлює статус книги як «доступна».

5) Перегляд історії видач

Бібліотекар або читач переглядає історію видач конкретної книги чи читача. Система відображає всі відповідні записи з датами видачі та повернення.

2.3 Обґрунтування структури моделі

Чітке структурування сутностей дозволяє:

− уникнути дублювання даних (наприклад, один автор може бути пов'язаний з багатьма книгами, без повторення імені);

− забезпечити нормалізацію БД, що спрощує підтримку та масштабування;

− дозволяє гнучко розширювати функціонал (наприклад, додати підтримку кількох жанрів на одну книгу, або рейтинг читача);

− забезпечити ефективне зв’язування сутностей: Reader − Loan − Book − Author / Genre.

Зв’язки OneToMany і ManyToOne реалізують логічну цілісність — кожна книга має автора й жанр, кожна видача належить до одного читача і однієї книги. Це дає змогу реалізувати гнучку фільтрацію, пошук і аналітику.

2.4 Особливості предметної області

Система управління бібліотекою має ряд специфічних особливостей, які варто враховувати при проєктуванні:

− дата видачі/повернення є ключовим параметром для обліку доступності книг та контролю своєчасного повернення;

− одна книга може бути багаторазово видана різним читачам — важливо зберігати повну історію видач;

− читач може одночасно мати кілька книг — тому потрібна перевірка на обмеження (наприклад, не більше 5 книг одночасно);

− жанр (важливий атрибут для класифікації літератури та зручного пошуку);

− автор може мати десятки або сотні книг, і зв'язок з ним має бути ефективним як у збереженні, так і у запитах;

− безпека доступу − важливо розділити ролі (бібліотекар / читач) для запобігання несанкціонованим діям (наприклад, видалення книг).

ПРОЕКТУВАННЯ ПРОГРАММНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі описано архітектуру та модель даних системи управління бібліотекою. Система реалізована з дотриманням принципів трирівневої архітектури:

− рівень доступу до даних (repository layer) − реалізований за допомогою Spring Data JPA;

− рівень бізнес-логіки (service layer) − обробляє запити та виконує перевірки/умови;

− рівень контролерів (controller layer) − надає REST API для взаємодії з клієнтською частиною або Swagger.

Для побудови системи використано Java (Spring Boot), PostgreSQL (на Render), а також засоби безпеки (Spring Security, JWT, OAuth2).

3.1 Структура даних (модель сутностей)

Модель даних системи охоплює основні об’єкти бібліотечної діяльності. Загалом реалізовано 6 ключових сутностей:

1) Book

Представляє літературну одиницю, що може бути видана читачам. Має зв’язки з автором, жанром та історією видач.

Поля:

− id: Long − унікальний ідентифікатор книги;

− title: String − назва книги;

− isbn: String − міжнародний стандартний номер книги;

− publishedYear: Integer − рік видання;

− author: Author − зв'язок з автором (ManyToOne);

− genre: Genre − зв'язок із жанром (ManyToOne).

2) Author

Відображає автора книги. Один автор може мати багато книг.

Поля:

− id: Long − ідентифікатор автора;

− name: String − ім’я автора;

− birthDate: LocalDate − дата народження;

− books: List<Book> − зв’язок OneToMany з Book.

3) Reader

Учасник системи, який може брати книги в користування. Має унікальний читацький квиток.

Поля:

− id: Long − ідентифікатор читача;

− firstName: String − ім’я;

− lastName: String − прізвище;

− libraryCardNumber: String − номер читацького квитка (унікальний);

− registeredAt: LocalDate − дата реєстрації;

− loans: List<Loan> − зв’язок з історією видач (OneToMany).

4) Loan

Відображає факт видачі книги певному читачеві.

Поля:

− id: Long − ідентифікатор видачі;

− book: Book − книга, яку було видано (ManyToOne);

− reader: Reader − читач, який отримав книгу (ManyToOne);

− loanDate: LocalDate − дата видачі;

− returnDate: LocalDate − дата повернення (може бути null).

Книга може бути видана багато разів різним читачам у різний час. Видача фіксується окремо.

5) Genre

Класифікаційна категорія книги.

Поля:

− id: Long − ідентифікатор жанру;

− name: String − назва жанру (напр. наукова фантастика, історія, драма);

− books: List<Book> − книги, що належать до цього жанру (OneToMany).

6) AppUser (для автентифікації)

Реалізує збереження облікових записів користувачів системи з ролями.

Поля:

− id: Long − ідентифікатор;

− email: String − логін користувача;

− password: String − захешований пароль;

− role: Role − роль користувача (READER або LIBRARIAN).

− імплементує UserDetails для інтеграції з Spring Security.

Зв’язки між сутностями:

Усі зв’язки між сутностями реалізовані через JPA (Hibernate) анотації:

@ManyToOne − у книзі до автора і жанру, у видачі до книги і читача;

@OneToMany(mappedBy = "...") − у автора до книг, у читача до видач;

@OneToOne можливо використовується для OAuth2/деталей профілю (в додаткових розширеннях).

3.2 Архітектура застосунку

Система управління бібліотекою реалізована відповідно до багаторівневої архітектури типу MVC (Model−View−Controller), адаптованої для створення

RESTful веб-застосунків. В основі − фреймворк Spring Boot, база даних PostgreSQL (на Render), система безпеки на основі Spring Security з підтримкою JWT та OAuth2.

Архітектура поділена на шість основних пакетів:

1) model

Містить сутності (Entity-класи), які відповідають таблицям у базі даних. Кожна сутність має відповідні поля, геттери/сеттери, а також зв’язки через JPA (@ManyToOne, @JoinColumn).

Усі запити приймають і повертають безпосередньо ці сутності — DTO-класи не використовуються.

Приклади:

Book, Author, Reader, Loan, Genre, AppUser.

2) repository

Інтерфейси для доступу до бази даних. Розширюють CrudRepository або JpaRepository і надають CRUD-операції (save, findById, deleteById, findAll), а також спеціалізовані запити, наприклад:

findByAuthorId(Long id),

findByGenreId(Long id),

existsByBookIdAndReturnDateIsNull().

3) service

Реалізує бізнес-логіку. Сервіси відповідають за:

− перевірку існування об’єктів перед діями;

− взаємодію з репозиторіями;

− обробку складних запитів (наприклад, видача книги, перевірка повернення, логіка реєстрації користувача);

− валідацію вхідних параметрів;

− логіку авторизації та хешування паролів (AppUserService).

Приклади:

BookService, LoanService, AppUserService, ReaderService.

4) controller

REST-контролери, які приймають HTTP-запити (@PostMapping, @GetMapping, @PutMapping, @DeleteMapping), викликають відповідні сервіси та формують відповіді через ResponseEntity.

Документація API реалізована через анотації OpenAPI (@Operation), а сам Swagger UI налаштовано через OpenApiConfig.

Контролери для всіх основних сутностей:

BookController, AuthorController, LoanController, ReaderController, GenreController, AuthController.

5) config

Містить конфігураційні класи застосунку:

SecurityConfig − основне налаштування Spring Security: правила доступу до ендпоінтів, логіка авторизації за ролями (READER, LIBRARIAN), підключення JWT-фільтра, налаштування OAuth2;

OpenApiConfig − налаштування Swagger для авторизації через Bearer JWT.

6) security

Містить компоненти для роботи з JWT:

− JwtProvider − генерація, підпис і перевірка валідності токенів;

− JwtAuthFilter − фільтр, який перехоплює запити, вилучає JWT з заголовка та встановлює аутентифікацію в контексті;

− CustomAppUserDetailsService − завантаження користувача з БД для Spring Security (реалізація UserDetailsService).

Завдяки такій структурі система легко розширюється, тестується через Swagger, а також підтримує безпечний доступ з використанням ролей, токенів і сторонніх сервісів авторизації (Google OAuth2).

3.3 Організація пакетів

У проєкті дотримано чіткої та логічної структури пакетів, яка забезпечує зрозумілу модульність:

− controller − містить класи-контролери (BookController, LoanController тощо), які обробляють REST-запити, викликають сервіси та повертають відповіді у форматі JSON;

− service − реалізує логіку обробки даних, взаємодію з репозиторіями, перевірки та трансформації. Наприклад: BookService, LoanService, AppUserService;

− repository − інтерфейси, що розширюють CrudRepository або JpaRepository, для прямого доступу до БД;

− model − містить JPA-сутності, які відображають таблиці в базі даних (Book, Author, Reader, Loan, Genre, AppUser). Передача даних між клієнтом і сервером відбувається напряму через ці сутності;

− config − зберігає конфігурацію безпеки (SecurityConfig), JWT-фільтрацію (SecurityFilterChain), а також конфігурацію Swagger (OpenApiConfig);

− security − містить компоненти безпеки: JwtProvider для генерації та валідації токенів, JwtAuthFilter для перевірки заголовків авторизації, CustomAppUserDetailsService для завантаження користувача.

Така структура забезпечує:

− високу читабельність коду;

− простоту навігації;

− чітке відокремлення відповідальностей;

− масштабованість і підтримку проєкту в майбутньому.

3.4 REST API

Усі основні дії в системі управління бібліотекою реалізовано через REST API, що дозволяє легко інтегрувати клієнтські застосунки (веб, мобільні тощо) з серверною частиною.

Запити організовано згідно з REST-принципами:

− GET − отримання ресурсів;

− POST − створення нових об’єктів;

− PUT − оновлення існуючих;

− DELETE − видалення об'єктів.

Приклади основних ендпоінтів для Reader (читач):

− POST /reader зареєструвати нового читача;

− GET /reader/all − отримати всіх читачів;

− GET /reader/{id} − отримати читача за ID;

− PUT /reader/{id} − оновити дані читача;

− DELETE /reader/{id} − видалити читача.

Loan (видачі книг)

− POST /loan − видати книгу читачу;

− PUT /loan/return/{id} − повернути книгу;

− GET /loan/all − отримати всі видачі;

− GET /loan/reader/{readerId} − історія видач певного читача;

− GET /loan/book/{bookId} − історія видач певної книги.

Book (книги)

− POST /book − додати нову книгу;

− GET /book/all − отримати всі книги;

− GET /book/{id} − отримати книгу за ID;

− PUT /book/{id} − оновити книгу;

− DELETE /book/{id} − видалити книгу;

− GET /book/author/{authorId} − книги певного автора;

− GET /book/genre/{genreId} − книги певного жанру.

Author (автори)

− POST /author − додати нового автора;

− GET /author/all − отримати всіх авторів;

− GET /author/{id} − отримати автора за ID;

− PUT /author/{id} − оновити дані автора;

− DELETE /author/{id} − видалити автора.

Genre (жанри)

− POST /genre − створити жанр;

− POST /genre/assign − призначити жанр для книги;

− GET /genre/all − список всіх жанрів;

− GET /genre/{genreId}/books − отримати книги за жанром;

− DELETE /genre/{id} − видалити жанр.

Auth (автентифікація)

− POST /auth/register − реєстрація нового користувача з логіном, паролем та роллю (READER / LIBRARIAN);

− POST /auth/login − логін з поверненням JWT-токена.

Особливості реалізації:

Усі запити повертають JSON-об'єкти (наприклад, книга, автор, список видач тощо). Swagger UI інтегровано через OpenAPI, що дозволяє тестувати всі запити через браузер.

Доступ до ендпоінтів обмежено ролями:

− LIBRARIAN − повний доступ до CRUD-операцій;

− READER − перегляд, повернення книг, історія своїх видач;

3.5 Універсальність архітектури

Архітектура застосунку побудована з урахуванням масштабованості, модульності та зручності розширення. Завдяки правильному розділенню логіки на окремі шари (controller, service, repository, model, config, security), система є універсальною і готовою до подальшого розвитку.

Потенціал архітектури:

− Легке додавання нових сутностей: наприклад, можна реалізувати додаткові сутності − «Рецензія», «Видавництво», «Рейтинг книги» − без порушення загальної структури;

− Розширення під багатокористувацьке середовище: реалізовано повноцінну автентифікацію користувачів з різними ролями (READER, LIBRARIAN), система вже підтримує реєстрацію, логін, JWT та OAuth2;

− Інтеграція з UI-фронтендом: усі контролери надають REST API, тому легко підключити сучасний фронтенд − React, Angular, Vue.js або мобільний застосунок (через JSON);

− Підтримка мікросервісної архітектури: завдяки чіткому розділенню логіки, можливий поділ системи на окремі сервіси − наприклад, сервіс обліку читачів, обробки видач, або сервіс авторизації;

− Зручна реалізація безпеки: Spring Security повністю інтегровано, з підтримкою ролей, JWT-фільтрацією, авторизацією через Google OAuth2 та захистом ендпоінтів.

Ця універсальність дозволяє адаптувати проєкт під реальні сценарії використання, масштабувати його для великої бібліотеки або навчального закладу, а також доповнювати без порушення існуючої архітектури.

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі описано практичну реалізацію програмного забезпечення згідно зі спроєктованою архітектурою. Було реалізовано повний набір CRUD-операцій для всіх ключових сутностей бібліотеки: книги, автори, читачі, жанри, видачі. Також додано аналітичні функції, які дозволяють переглядати історію видач книги або конкретного читача.

Застосунок реалізовано мовою програмування Java із використанням Spring Boot, що забезпечує:

− інверсію керування (IoC);

− розділення відповідальностей (контролери, сервіси, репозиторії);

− автоматичну ін'єкцію залежностей через анотації @Autowired або @RequiredArgsConstructor.

4.1 Моделі (Model-класи)

Сутності створено з використанням:

− JPA-анотацій (@Entity, @Id, @ManyToOne, @OneToMany) − для визначення структури таблиць та їхніх зв’язків у PostgreSQL;

− Lombok (@Data, @NoArgsConstructor, @AllArgsConstructor) − для скорочення коду: автоматична генерація геттерів, сетерів, конструкторів.

Усі сутності мають поле id типу Long, яке генерується автоматично через @GeneratedValue.

Приклад: Book.java

```
@Entity
@Data
@NoArgsConstructor
```

```
@AllArgsConstructor
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String isbn;
    private Integer publishedYear;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;

    @ManyToOne
    @JoinColumn(name = "genre_id")
    private Genre genre;
}
```

У всіх моделях використовуються анотації:

− @Entity − для позначення сутності;

− @Id, @GeneratedValue − для первинного ключа;

− @ManyToOne, @OneToMany − для зв’язків;

− @JoinColumn − для вказівки на зовнішній ключ.

Це дозволяє побудувати коректну реляційну схему у базі даних PostgreSQL та ефективно взаємодіяти з нею через Spring Data JPA.

4.2 Репозиторії

Для кожної сутності створено окремий інтерфейс репозиторію, який розширює JpaRepository. Це дозволяє отримати доступ до всіх CRUD-операцій (створення, оновлення, видалення, пошук) без необхідності їх ручного написання. Крім того, Spring Data JPA автоматично генерує реалізацію на основі імені методу.

Також, реалізовано додаткові методи з використанням:

− іменованих запитів (findBy...);

− JPQL-запитів через @Query;

− фільтрацій за параметрами (findByAuthorId, findByGenreId, тощо).

Приклади репозиторіїв:

```
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByAuthorId(Long authorId);
    List<Book> findByGenreId(Long genreId);
}

public interface LoanRepository extends JpaRepository<Loan, Long> {
    List<Loan> findByReaderId(Long readerId);
    List<Loan> findByBookId(Long bookId);
    boolean existsByBookIdAndReturnDateIsNull(Long bookId);
}
```

Репозиторії служать основою для запитів, які викликаються у відповідних сервісах. Це дозволяє розмежувати доступ до бази даних і логіку обробки даних.

4.3 Сервіси

Сервісний рівень відповідає за обробку всієї логіки, що виконується після отримання запиту та перед передачею відповіді. Саме тут відбувається

перевірка даних, взаємодія з репозиторіями, логіка доступу, валідація і підготовка об’єктів до збереження.

Сервіси побудовані з використанням анотацій @Service і @RequiredArgsConstructor, що дозволяє автоматично інжектити залежності через конструктор.

Приклад: LoanService

```
public Loan issueBook(Long bookId, Long readerId) {
    if (loanRepository.existsByBookIdAndReturnDateIsNull(bookId)) {
        throw new IllegalStateException("Book is already loaned out.");
    }
    Book book = bookRepository.findById(bookId).orElseThrow();
    Reader reader = readerRepository.findById(readerId).orElseThrow();

    Loan loan = new Loan();
    loan.setBook(book);
    loan.setReader(reader);
    loan.setLoanDate(LocalDate.now());

    return loanRepository.save(loan);
}
```

Сервіси — центральне місце для реалізації ділової логіки, завдяки чому контролери залишаються чистими та зосередженими лише на маршрутах.

4.4 Контролери

Контролери − це точка входу у додаток для користувача або клієнта (браузера, мобільного застосунку тощо). Вони приймають HTTP-запити, викликають відповідні сервіси та повертають результат у форматі JSON.

Усі контролери реалізовано за допомогою @RestController та @RequestMapping. Також використовується анотація @PreAuthorize, яка забезпечує доступ лише для певних ролей.

Приклад: ReaderController

```
@PostMapping
@PreAuthorize("hasRole('LIBRARIAN')")
public Reader registerReader(@RequestBody Reader reader) {
    return readerService.register(reader);
}
```

Для кожної сутності створено окремий контролер:

− BookController;

− AuthorController;

− ReaderController;

− LoanController;

− GenreController;

− AuthController.

Це дозволяє чітко структурувати API за ресурсами.

4.5 Реєстрація та автентифікація користувача

Реалізація автентифікації є ключовою частиною другої половини проєкту. Система підтримує як реєстрацію через логін/пароль (з JWT-токеном), так і авторизацію через Google (OAuth2).

4.5.1 Реєстрація

Запит POST /auth/register дозволяє створити нового користувача. Введений пароль хешується через BCryptPasswordEncoder, а результат зберігається у таблиці app_user.

4.5.2 Авторизація

Після логіну (POST /auth/login) система формує JWT-токен:

```
{
    "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9...",
    "type": "Bearer"
}
```

Токен передається в заголовку Authorization при наступних запитах. Він перевіряється у JwtAuthFilter, ідентифікує користувача та забезпечує доступ до захищених маршрутів.

4.5.3 Авторизація через OAuth2

Система інтегрується з Google через OAuth2. Після успішного входу:

− отримується email користувача;

− створюється новий запис у БД, якщо його не існувало;

− видається JWT-токен так само, як при звичайному логіні.

Цей функціонал реалізовано в OAuth2SuccessHandler, який формує JSON-відповідь із токеном.

4.5.4 Конфігурація безпеки

У класі SecurityConfig прописано:

− які маршрути відкриті (/auth/**, /swagger-ui/**);

− які захищені (/reader/**, /loan/** — лише для LIBRARIAN);

− підключення фільтрів JwtAuthFilter;

− OAuth2 логіку (/oauth2/authorization/google);

− глобальні обробники помилок (JwtAuthEntryPoint, AccessDeniedHandler).

4.5.5 Глобальна обробка помилок

Для всіх винятків використовується клас GlobalExceptionHandler, який дозволяє централізовано:

− обробити помилки пошуку (EntityNotFoundException);

− відловити порушення логіки (IllegalStateException);

− повернути єдиний формат помилки у відповіді.

Формат помилки у відповіді:

```
{
    "success": false,
    "message": "Reader not found",
    "error": "Not Found"
}
```

Це забезпечує кращий UX і спрощує відладку на стороні клієнта.

4.6 Конфігурація, база даних та хостинг

Усі параметри конфігурації системи винесено до файлу application.yml:

− підключення до PostgreSQL на Render;

− секрет JWT та час життя токена;

− параметри Google OAuth2 (clientId, clientSecret, callback-uri);

− автоматичне створення таблиць (ddl-auto=update);

− лог SQL-запитів (show-sql=true).

Застосунок успішно розгорнуто на Render і протестовано через Swagger UI.

ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Реєстрація нового користувача з повними правами: (рис. 5.1)

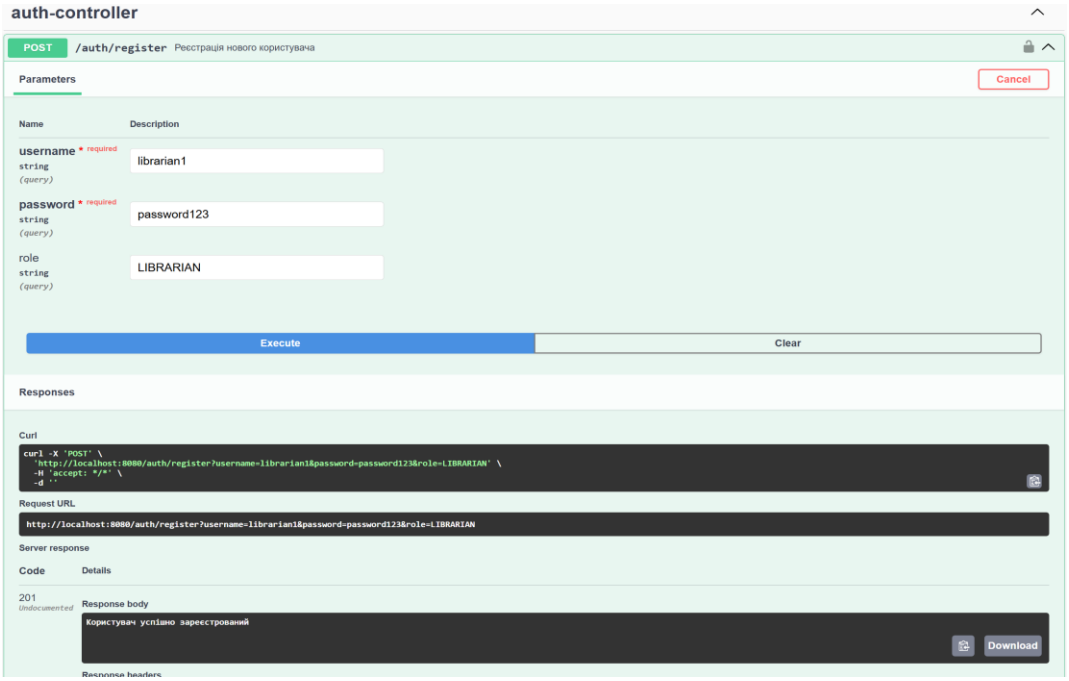

Рисунок 5.1 − Результат виконання запиту POST /auth/register

Ми створюємо обліковий запис із роллю LIBRARIAN, щоб потім за його допомогою отримувати JWT-токен і виконувати адміністративні операції (додавати/редагувати книги, авторів тощо). Код відповіді 201 і повідомлення «Користувач успішно зареєстрований» підтверджують успішну реєстрацію.

Аутентифікація та отримання JWT-токену: (рис. 5.2)

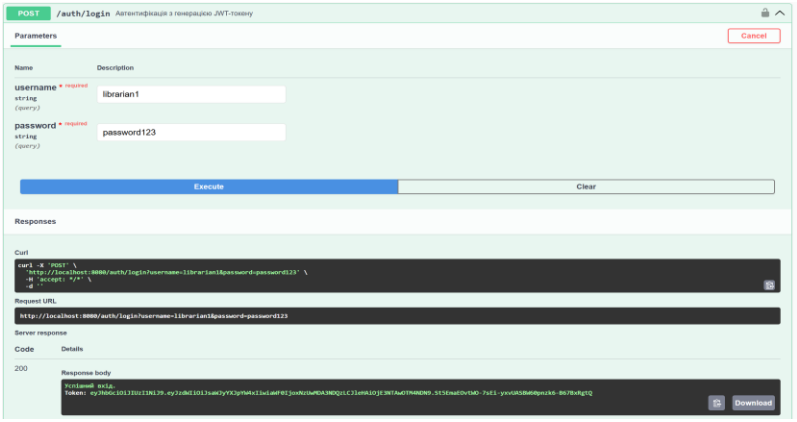

Рисунок 5.2 - Результат виконання запиту POST /auth/login

За допомогою тільки-що створеного облікового запису отримуємо JWT-токен. Він знадобиться для авторизації захищених ендпоінтів.

Авторизація у Swagger (Authorize): (рис. 5.3)

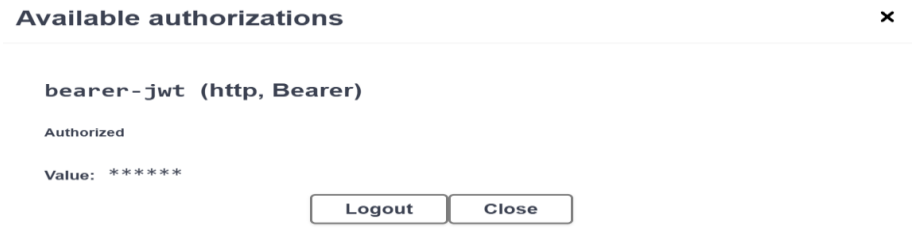

Рисунок 5.3– Авторизація у Swagger

Після цього всі запити до захищених ресурсів (наприклад, POST /author) автоматично додаватимуть заголовок Authorization і проходитимуть перевірку ролей.

Додавання нового автора: (рис 5.4)

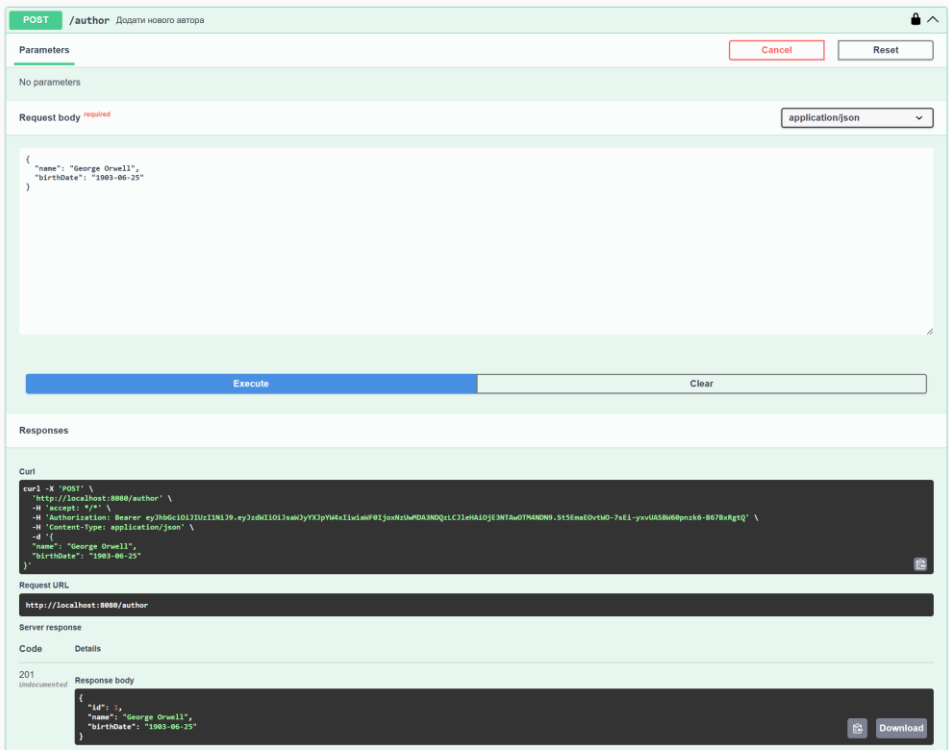

Рисунок 5.4 – Успішне додавання нового автора

Перевіряємо, що роль LIBRARIAN дозволяє створювати авторів. Відповідь 201 і повернений об’єкт із полем id свідчать про успішне додавання.

Оновлення автора: (рис. 5.5)

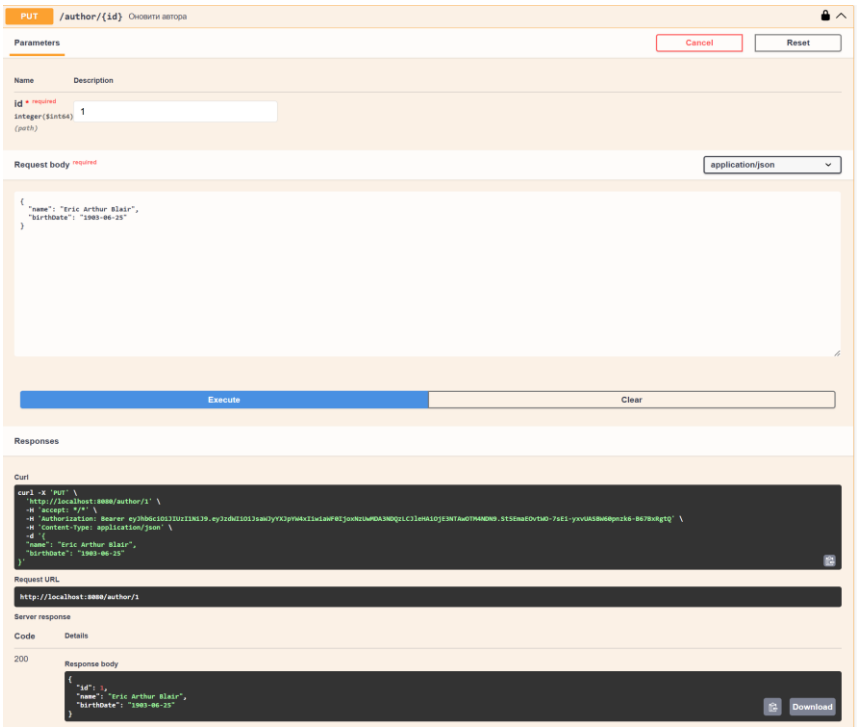

Рисунок 5.5 – Успішне оновлення автора

Роль LIBRARIAN дозволяє змінювати дані автора. Відповідь 200 OK із оновленим об’єктом підтверджує успіх.

Виведення автора за id: (рис. 5.6)

Додамо ще одного автора для того, щоб наступні перевірки були більш наглядними:

```
{
  "name": "Taras Shevchenko",
  "birthDate": "1814-03-09"
}
```

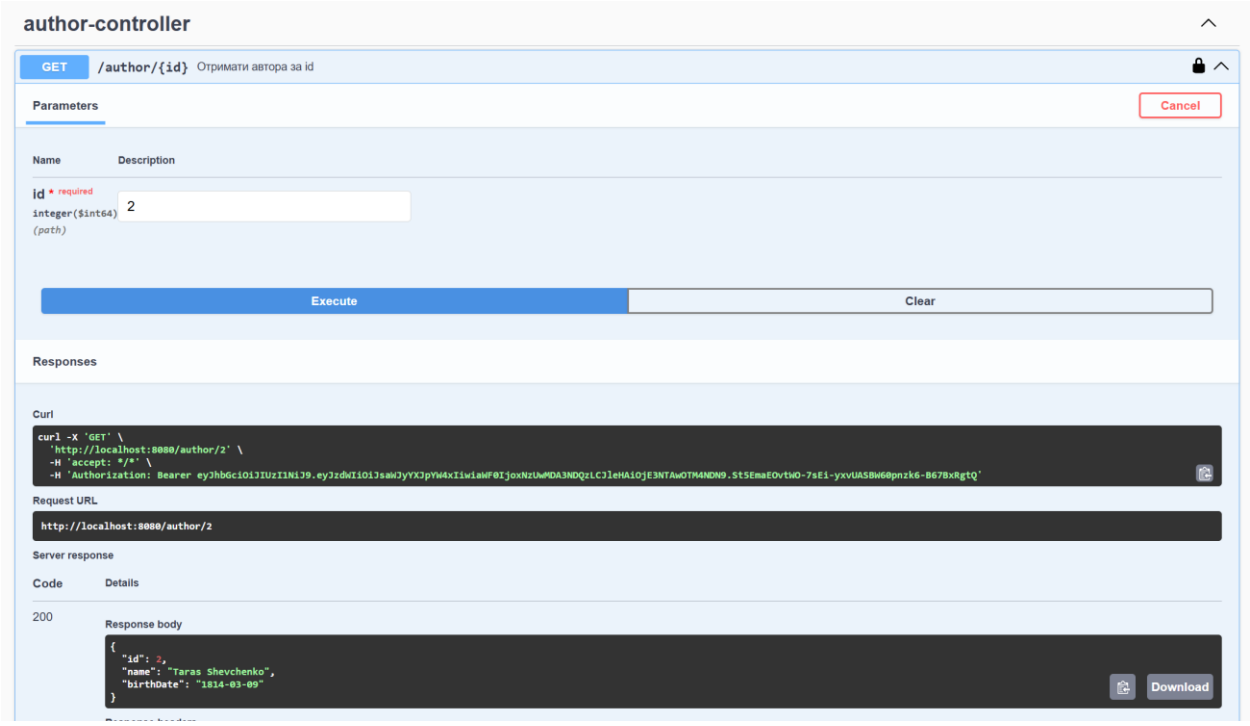

Рисунок 5.6 – Успішне виведення автора за власним id

Бачимо успішну роботу знаходження потрібно автора за його особистим id-номером.

Виведення всіх існуючих авторів: (рис. 5.7)

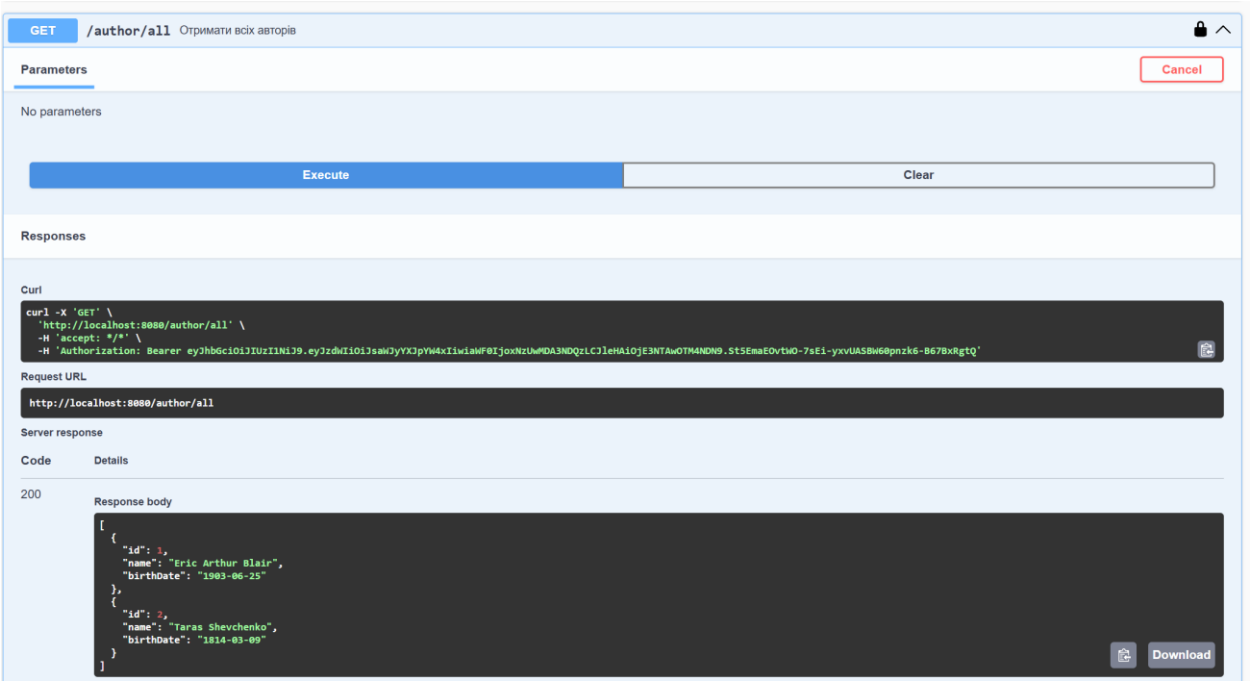

Рисунок 5.7 – Успішне виведення всіх авторів

Видалення автора за власним id: (рис 5.8)

Рисунок 5.8 – Успішне видалення автора за власним id

Додавання жанру: (рис. 5.9)

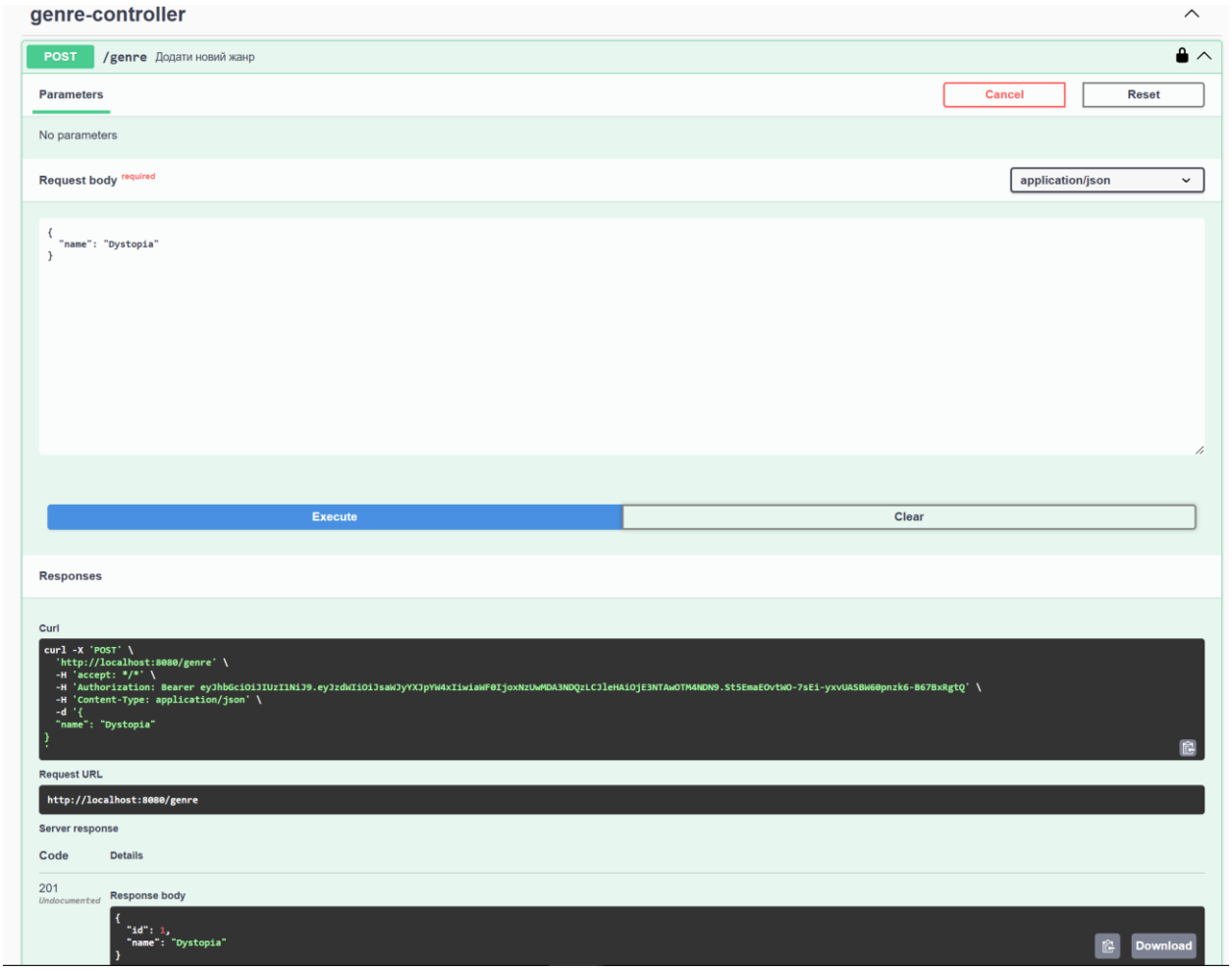

Рисунок 5.9 – Успішне додавання нового жанру

Для перевірки наступних запитів додамо ще один жанр:

```
{
  "name": "Comedy"
}
```

Виведення усіх жанрів: (рис. 5.10)

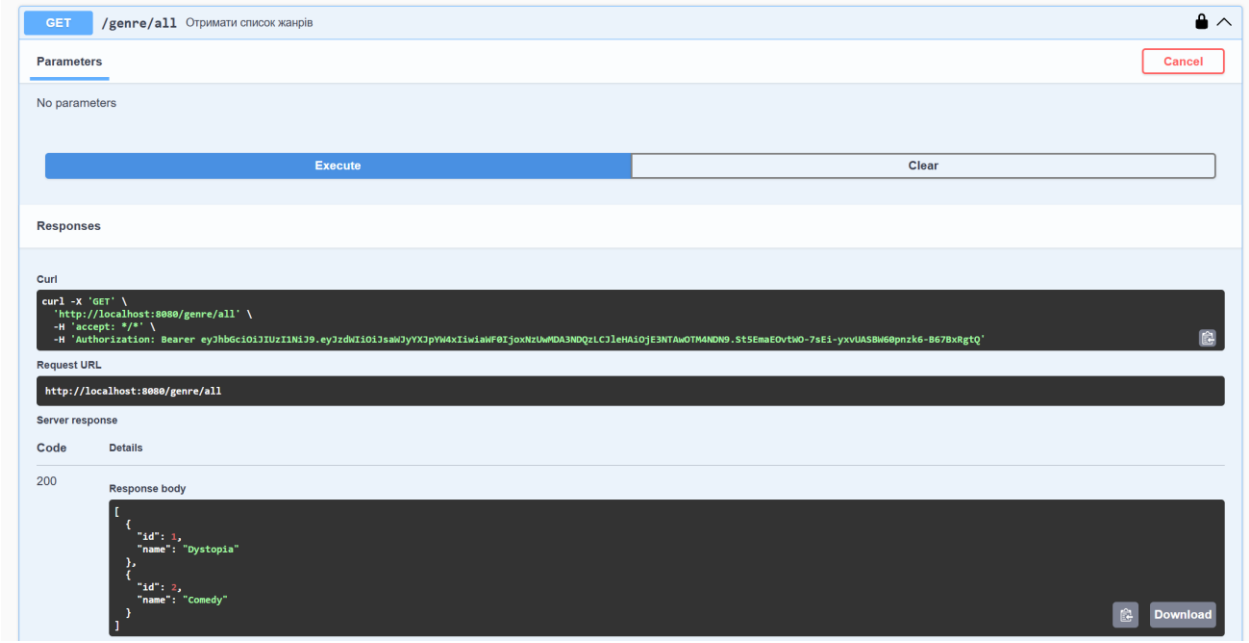

Рисунок 5.10 – Успішне виведення усіх жанрів

Пошук книги за жанром: (рис. 5.11)

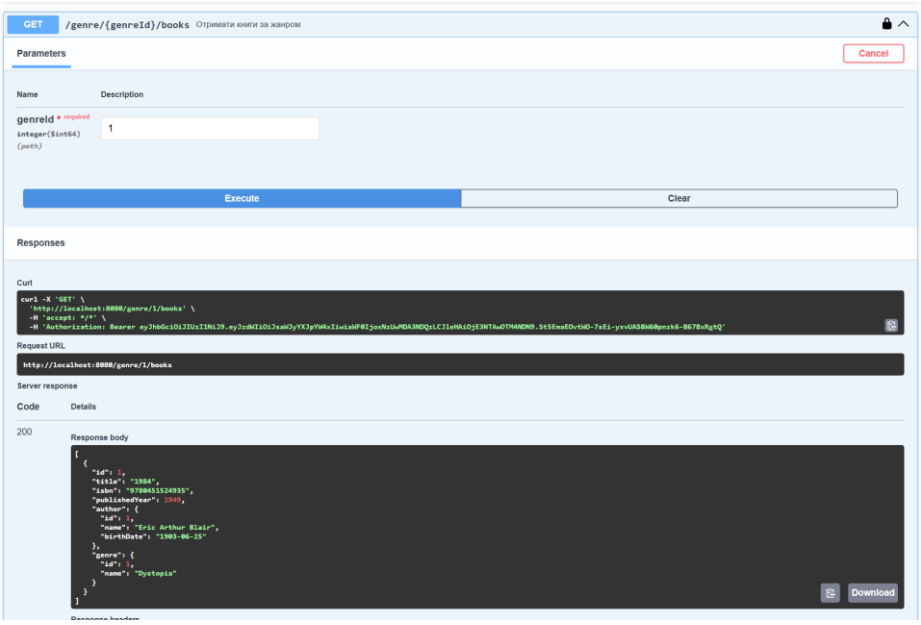

Рисунок 5.11 – Успішне находження книги за власним id-номером жанру

Успішне видалення жанру за власним id: (рис. 5.12)

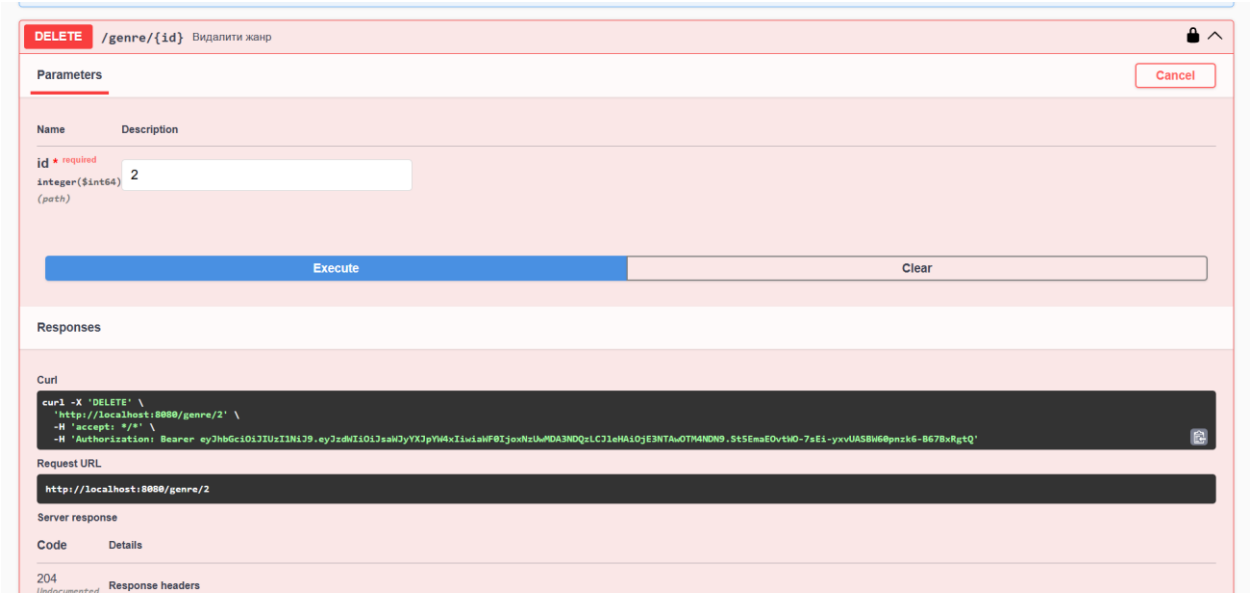

Рисунок 5.12 – Успішне видалення жанру за власним id-номером

Додавання нової книги: (рис. 5.13)

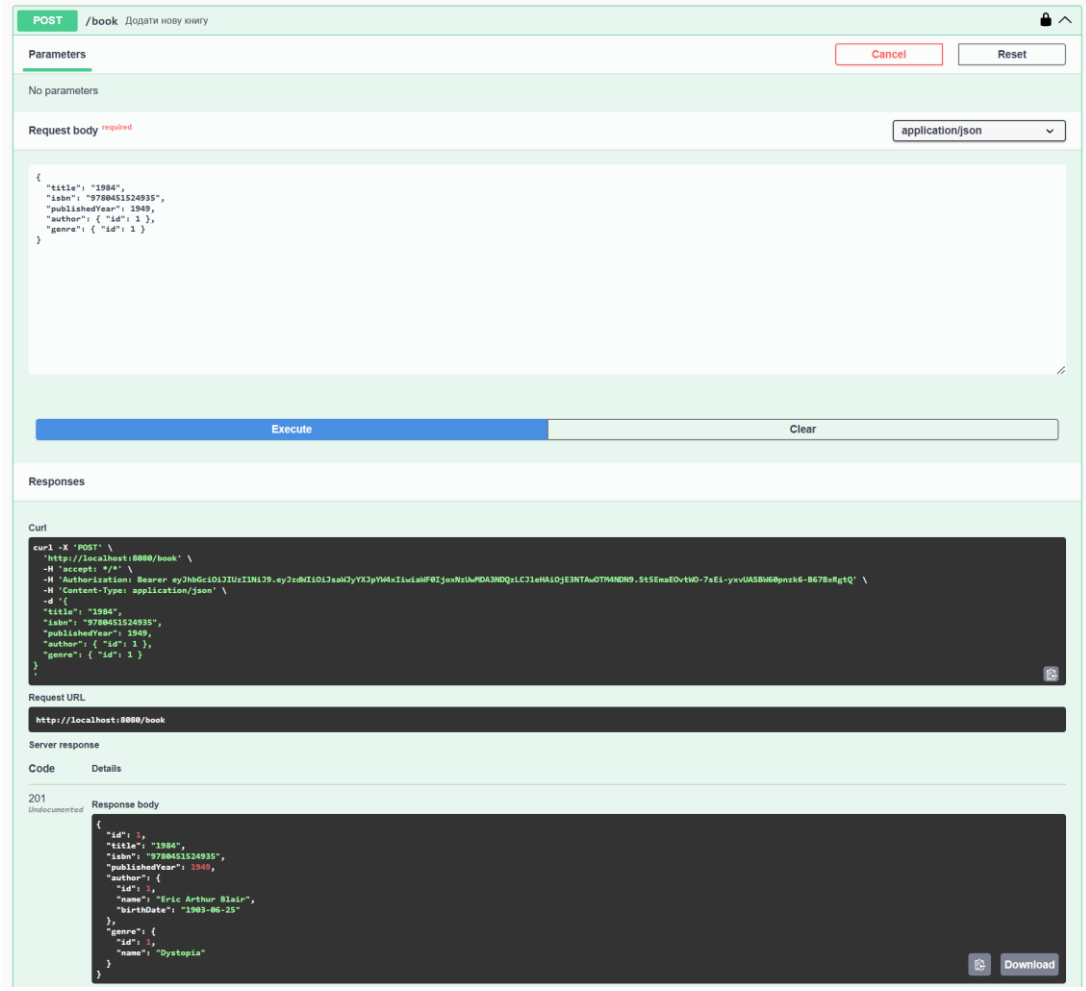

Рисунок 5.13 – Успішне додавання нової книги

Для перевірки наступних запитів додамо ще одину книгу:

```
{
  "title": "Nineteen Eighty-Four",
  "isbn": "9780451524935",
  "publishedYear": 1949,
  "author": { "id": 1 },
  "genre": { "id": 2 }
}
```

Виведення усіх книг: (рис. 5.14)

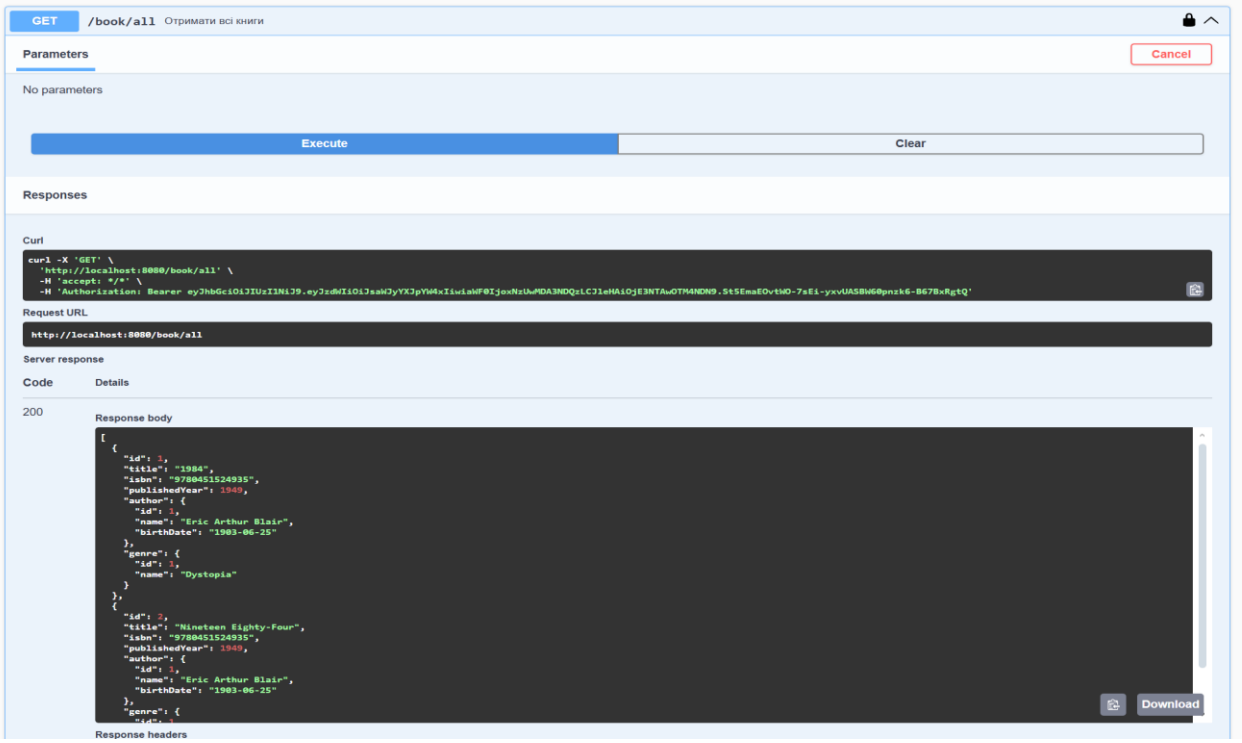

Рисунок 5.14 – Успішне усіх книг

Пошук книг, певного автора за власним id автора: (рис. 5.15)

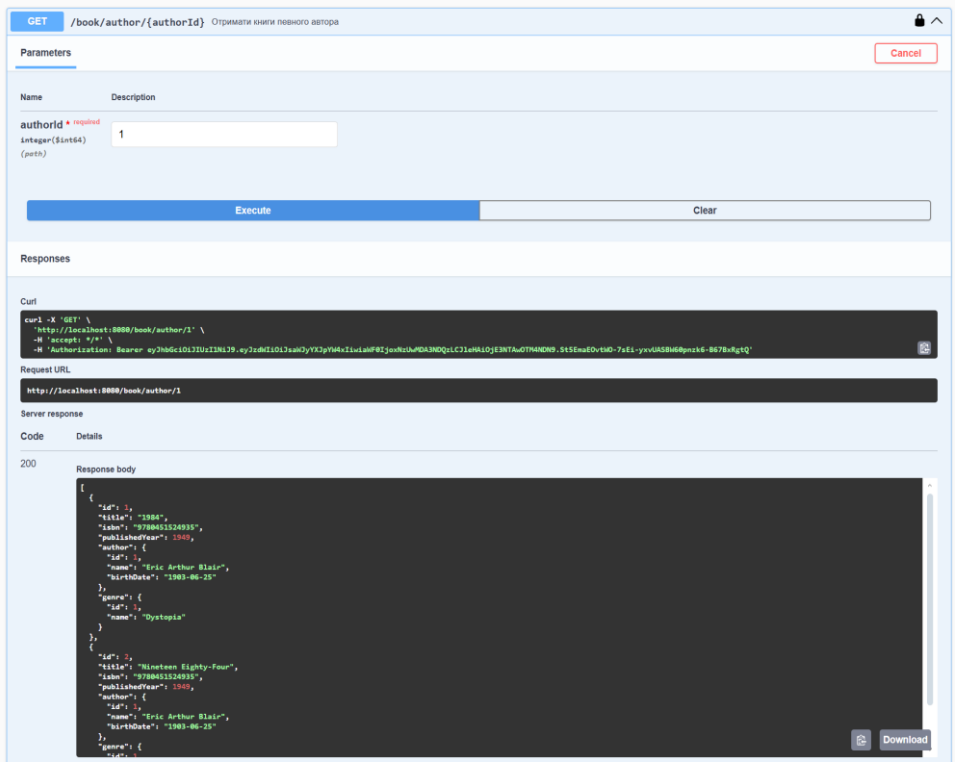

Рисунок 5.15 – Успішний пошук книг певного автора

Пошук книги за власним id: (рис. 5.16)

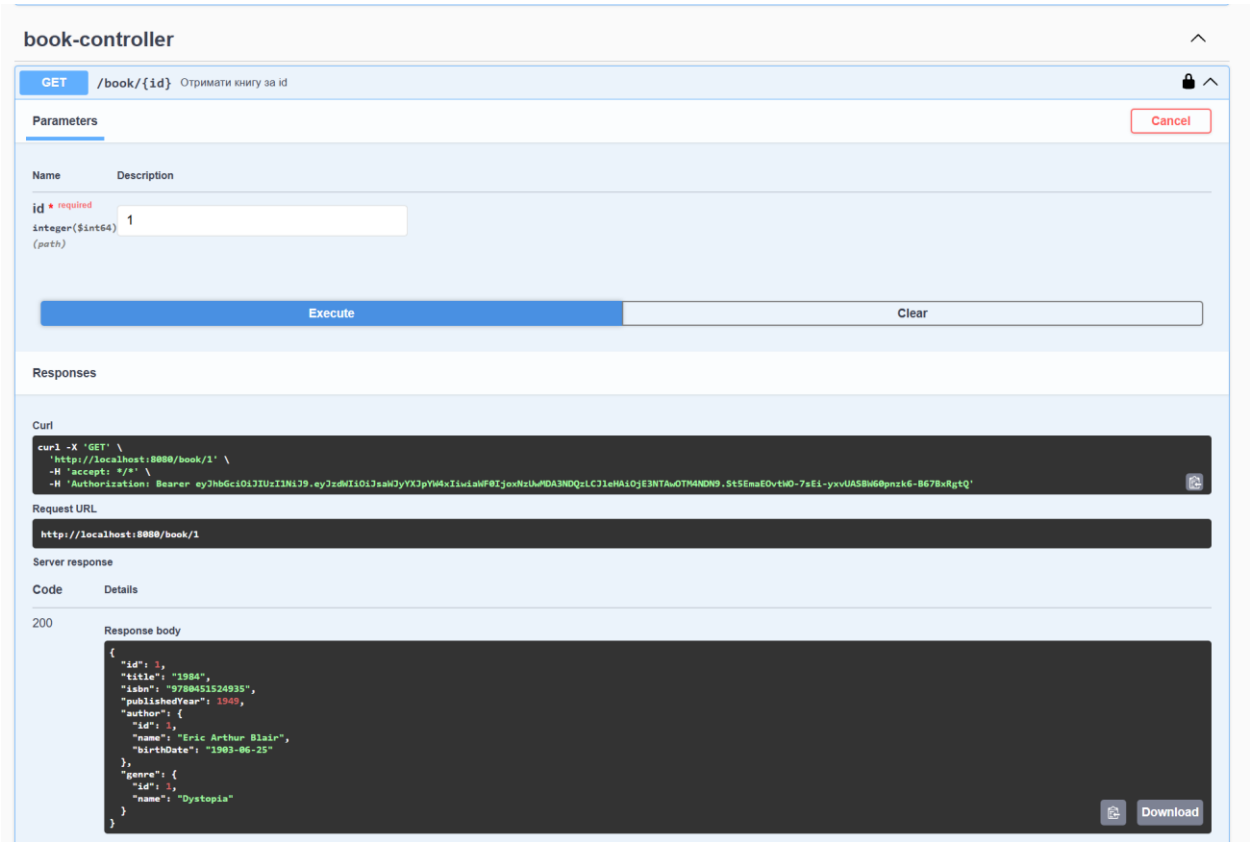

Рисунок 5.16 – Успішний книги за власним id

Таким же чином працють і настпуні запити, які точно таким же чином правильно виконують свої функції за наявністю певних прав доступу.

Перегляд усіх стровенних запитів: (рис. 5.17)

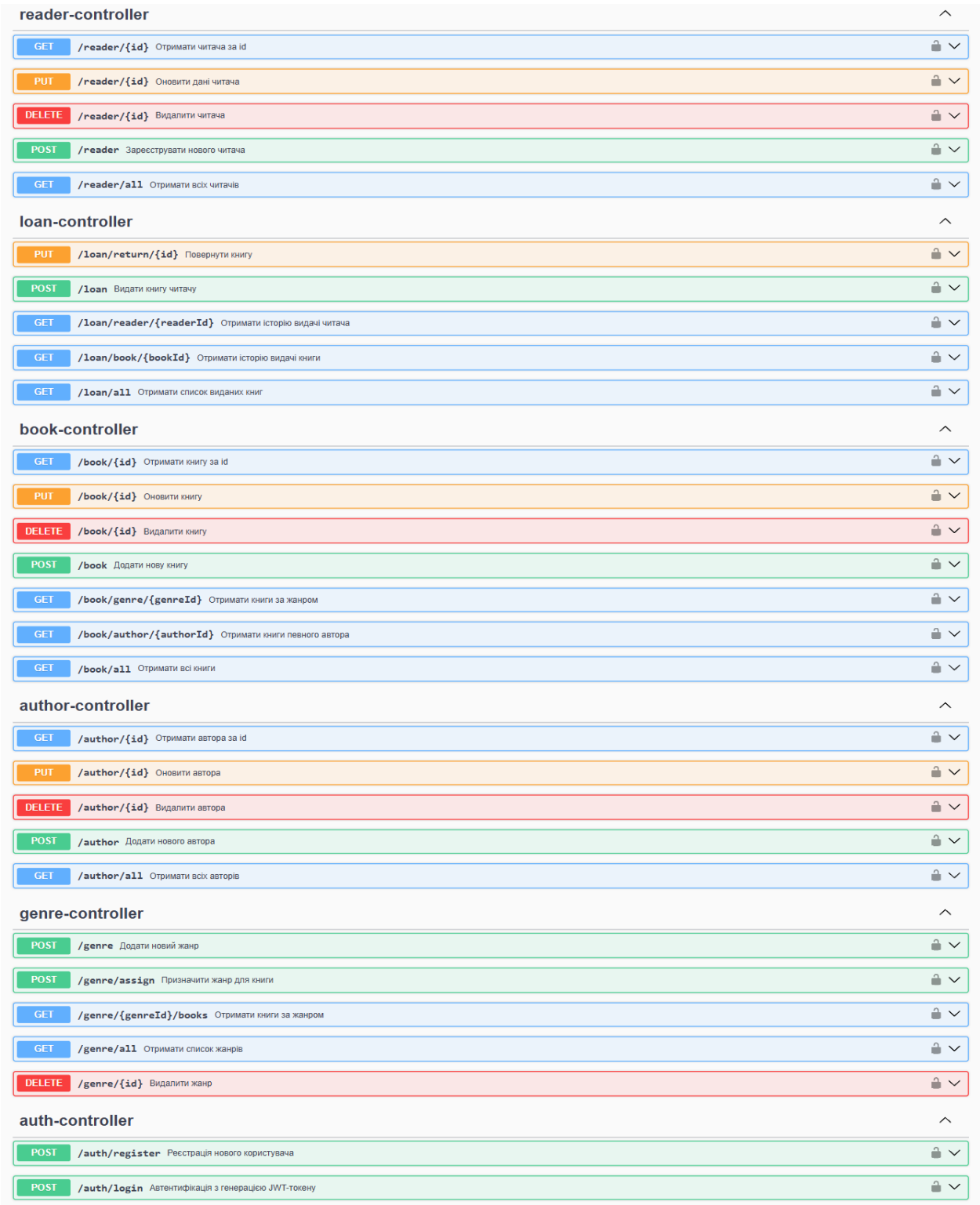

Рисунок 5.17 – Усі створенні запити в Swagger

Аутентифікація через oAuth2: (рис 5.18)

При використанні данного посилання http://localhost:8080/auth/login

Відкривається наступне вікно

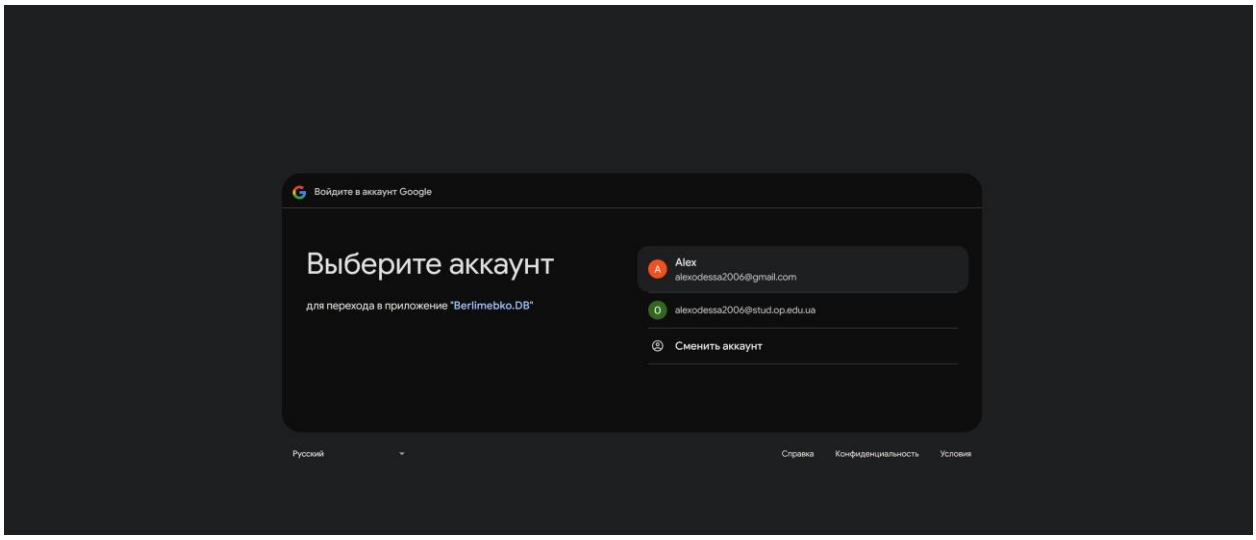

Рисунок 5.18 – Вибір облікового запису Google для входу через OAuth2

Після успішної аутентифікація через обліковий запит користувач потрапляє на головну сторінку Swagger де знаходяться усі запити БД.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті виконання курсової роботи реалізовано повнофункціональний веб-застосунок для автоматизованого обліку бібліотечних ресурсів. Система дозволяє:

- керувати каталогом книг (створення, читання, оновлення, видалення записів про книги та їх авторів і жанри);
- реєструвати та оновлювати дані читачів;
- видавати книги читачам і фіксувати дати позики та повернення;
- отримувати історію видачі за читачем або за конкретною книгою.

Архітектура застосунку побудована за принципом багаторівневого MVC:

1. Контролери обробляють HTTP-запити та повертають відповіді (JSON або plain text).
2. Сервіси містять бізнес-логіку (реєстрація користувачів, CRUD-операції, логіка позик).
3. Репозиторії забезпечують доступ до бази даних PostgreSQL через Spring Data JPA.
4. Сутності (AppUser, Author, Book, Genre, Reader, Loan) відображаються на відповідні таблиці БД, із налаштованими зв’язками OneToMany та ManyToOne.

У першій частині роботи реалізовано:

- CRUD-функціонал для сутностей Author, Book, Genre, Reader, Loan;
- Призначення жанру книзі та фільтрацію книг за автором і жанром;
- Валідацію вхідних даних і централізовану обробку помилок.

Друга частина присвячена безпеці:

- Реєстрація користувачів із шифруванням пароля (BCrypt) та перевіркою унікальності логіна;
- Аутентифікація за логіном/паролем із генерацією JWT-токена;

- OAuth2-вхід через Google (редирект, прихований у Swagger);
- Налаштування Spring Security, яке захищає всі маршрути (окрім /auth/register та /auth/login) та реалізує фільтрацію JWT-токена через власний JwtAuthFilter;
- Розмежування прав доступу за ролями READER і LIBRARIAN.

Усі ендпоінти протестовані у Swagger UI (авторизація через “Authorize”, перевірка CRUD-операцій) та в Postman (успішні та негативні кейси). Результати тестування підтвердили стабільність роботи, коректність бізнес-логіки і надійність захисту.

Таким чином, поставлену мету – створення безпечної, розширюваної веб-системи управління бібліотекою – досягнуто. Отриманий застосунок готовий до практичного використання та може слугувати базою для подальшого розвитку (наприклад, додаванням пошуку, рейтингів книг, нагадувань про повернення тощо). Розробка цього проєкту поглибила знання з Java, Spring Boot, Spring Data JPA, Spring Security, JWT, OAuth2, а також навички проєктування баз даних і тестування REST-API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» для здобувачів інституту комп’ютерних систем за спеціальністю 122 Комп'ютерні науки, освітня програма «Комп’ютерні науки» / Укладачі: М.А. Годовиченко. – Одеса: Національний університет «Одеська Політехніка», 2024. – 59 с.

2. Spring Boot Documentation Site. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 23.04.2025)

3. Official OAuth 2.0 Authorization Framework. URL: https://oauth.net/2/ (дата звернення: 11.06.2025)

4. JWT.io – Introduction to JSON Web Tokens. URL: https://jwt.io/introduction/ (дата звернення: 05.06.2025).

5. Lombok Project Documentation. URL: https://projectlombok.org/ (дата звернення: 18.06.2025)