

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут штучного інтелекту та робототехніки
Кафедра штучного інтелекту та аналізу даних

Мартинюк Сергій Сергійович
студент групи AI-231

Курсова робота з дисципліни
«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»

Затверджено на засіданні
кафедри інформаційних систем
протокол №1 від 28 серп
ня 2024 р. Одеса – 2025

ЗМІСТ

Вступ.....	4
1. Аналіз предметної області.....	5
2. Проєктування програмного забезпечення.....	7
2.1 . Проєктування структури даних	7
2.2. Опис архітектури застосунку	8
2.3. Опис RESTAPI.....	9
3. Реалізація програмного продукту.....	11
4. Тестування та налагодження.....	44
5. Висновок.....	53
6. Список використаних джерел.....	54
7. Додатки	54

АНОТАЦІЯ

Мартинюк С.С. Система управління бібліотекою: курсова робота з дисципліни «Об'єктно-орієнтоване програмування» за спеціальністю «122 Комп'ютерні науки» /Мартинюк Сергій Сергійович; керівник Микола Анатолійович Годовиченко. – Одеса : Нац. ун-т «Одес. політехніка», 2025. – 54 с.

Курсова робота містить основну текстову частину на 54 сторінках, список використаних джерел з 5 найменувань на 1 сторінці, додатки на 1 сторінці.

Розглянуто завдання для додатку.

Створено проєкт для роботи з бібліотекою.

Запропоновані запити для більш детального тестування проєкту.

Ключові слова: Spring Boot, база даних, REST API, Render, Java.

Вступ

Мета курсової роботи:

Систематизувати, поглибити та практично закріпити знання з дисципліни «Об'єктно-орієнтоване програмування», а також засвоїти навички самостійної розробки серверної частини прикладного програмного забезпечення засобами Java та фреймворку Spring Boot.

Актуальність теми:

Сучасні інформаційні технології відіграють ключову роль в автоматизації та оптимізації процесів управління бібліотечними системами. Потреба у цифрових рішеннях цього напрямку стабільно зростає, що забезпечує незмінну актуальність розробленого програмного продукту.

Завдання, виконані в межах проєкту: □

- Створення RESTful веб-сервісу з повною підтримкою CRUD-операцій.
- □ Зберігання та обробка даних у реляційній базі даних. □
- Коректна взаємодія компонентів застосунку завдяки чіткому поділу на шари (Controller → Service → Repository). □
- Розробка архітектури на основі Spring Framework із використанням Spring Data JPA, Lombok, MapStruct та інших сучасних інструментів. □
- Реалізація моделі предметної області з такими сутностями: Author, Book, Genre, Reader, Loan, User.

Створення захищеного REST API, що забезпечує: □

- додавання, перегляд, оновлення та видалення інформації; □
- автентифікацію та авторизацію користувачів на основі Spring Security та JWT

1. Аналіз предметної області

Кіноклуб – це культурна спільнота, що об'єднує людей, зацікавлених у спільному перегляді фільмів, обговоренні кінострічок та висловленні власних вражень у вигляді рецензій. У сучасному світі, де цифрові технології є невід'ємною частиною повсякденного життя, виникає необхідність автоматизації ключових процесів у діяльності кіноклубу. Це дозволяє покращити організацію показів фільмів, зберігати дані про учасників, перегляди, рецензії, а також підвищити ефективність управління клубом загалом.

Дана предметна область охоплює діяльність кіноклубу, в якому основними об'єктами є фільми, учасники, перегляди, рецензії та відвідування. Основні взаємозв'язки між об'єктами: один фільм може мати багато переглядів і рецензій; учасник може бути присутнім на багатьох переглядах і залишати рецензії на різні фільми.

Проблеми, що можуть виникати в предметній області:

- Введення некоректних або неповних даних ускладнює аналітику та облік.
- При високому навантаженні на систему (наприклад, під час реєстрації багатьох переглядів) можлива втрата продуктивності.
- Без автоматизації важко швидко знаходити потрібну інформацію (найкращі фільми, перегляди певного учасника тощо).

Розроблена система дозволяє автоматизувати такі процеси:

- Додавання, редагування та видалення фільмів, з можливістю фільтрації за жанром, режисером, рейтингом, тривалістю, роком.
- Реєстрація учасників та оновлення їх персональних даних.
- Організація переглядів з фіксацією дати, місця проведення та фільму.

- Відстеження відвідуваності кожного перегляду.
- Можливість учасникам залишати рецензії з оцінками та коментарями.
- Отримання статистики щодо активності учасника, середніх оцінок фільмів, найрейтинговіших стрічок, тощо.

Таким чином, автоматизована система управління кіноклубом значно полегшує роботу як організаторів, так і учасників, забезпечує зручний інтерфейс взаємодії, знижує ризики помилок та підвищує ефективність зберігання і обробки інформації.

2. Проєктування програмного забезпечення

2.1 . Проєктування структури даних

Під час розроблення проєкту, було створено такі сутності та їхні атрибути:

Сутності:

Movie:

- id: Long
- title: String
- director: String
- year: Integer
- genre: String

Member:

- id: Long
- name: String
- email: String

Screening:

- id: Long
- movie: Movie
- date: LocalDate
- location: String

Attendance:

- id: Long
- screening: Screening
- member: Member

Review:

- id: Long
- member: Member
- movie: Movie
- rating: Integer
- comment: String

Між цими сутностями представлені наступні зв'язки:

- Movie може бути показаний під час багатьох Screening.
- Member може бути присутнім на багатьох Screening через зв'язуючу сутність Attendance.
- Attendance пов'язує Member та Screening, фіксуючи факт участі у перегляді.
- Member може залишити багато Review.
- Review пов'язує Member та Movie, фіксуючи оцінку й коментар.
- Screening належить до одного Movie.
- Кожен User (у майбутньому, якщо реалізовуватимеш автентифікацію) може бути прив'язаний до Member або мати роль адміністратора для керування системою.

2.2. Опис архітектури застосунку

Також у проєкті реалізована класична архітектура, що складається з трьох рівнів — Controller – Service – Repository, кожен з яких виконує свою важливу функцію:

- **Controller** — відповідає за обробку HTTP-запитів від користувача. Він приймає запити (GET, POST, PUT, DELETE), викликає відповідні методи сервісного рівня та формує HTTP-відповіді.
- **Service** — реалізує бізнес-логіку системи. Саме тут відбувається обробка даних, перевірки, взаємодія між сутностями та координація викликів репозиторіїв. Цей рівень дозволяє централізовано керувати логікою програми.
- **Repository** — безпосередньо взаємодіє з базою даних за допомогою Spring Data JPA. Він забезпечує доступ до збережених даних та реалізує стандартні та кастомні CRUD-операції для сутностей, таких як Movie, Member, Screening, Attendance, Review.

Цей підхід дозволяє досягти високої модульності, гнучкості та зручності супроводу проєкту.

2.3. опис REST API

Також для кожної сутності було розроблено контролери з REST-запитами для коректної роботи проєкту filmclub:

MovieController

- Створити новий фільм — @PostMapping
- Отримати всі фільми — @GetMapping
- Отримати фільм за ID — @GetMapping("/{id}")
- Оновити фільм за ID — @PutMapping("/{id}")
- Видалити фільм за ID — @DeleteMapping("/{id}")
- Отримати фільми за movieId — @GetMapping("/movie/{movieId}")
- Отримати фільми за genreId — @GetMapping("/genre/{genreId}")

MemberController

- Створити нового учасника — @PostMapping
- Отримати всіх учасників — @GetMapping
- Отримати учасника за ID — @GetMapping("/{id}")

- Оновити учасника за ID — @PutMapping("/{id}")
- Видалити учасника за ID — @DeleteMapping("/{id}")

ScreeningController

- Створити новий сеанс — @PostMapping("/schedule")
- Повернути всі сеанси — @GetMapping
- Отримати сеанс за ID — @GetMapping("/{id}")
- Оновити сеанс за ID — @PutMapping("/{id}")
- Видалити сеанс за ID — @DeleteMapping("/{id}")
- Отримати сеанси за movieId — @GetMapping("/movie/{movieId}")
- Отримати сеанси за memberId — @GetMapping("/member/{memberId}")

AttendanceController

- Зареєструвати відвідування (нова присутність) — @PostMapping("/register")
- Отримати всі записи про відвідування — @GetMapping
- Отримати відвідування за ID — @GetMapping("/{id}")
- Оновити статус відвідування за ID — @PutMapping("/{id}")
- Видалити запис про відвідування за ID — @DeleteMapping("/{id}")
- Отримати відвідування за movieId — @GetMapping("/movie/{movieId}")
- Отримати відвідування за memberId — @GetMapping("/member/{memberId}")

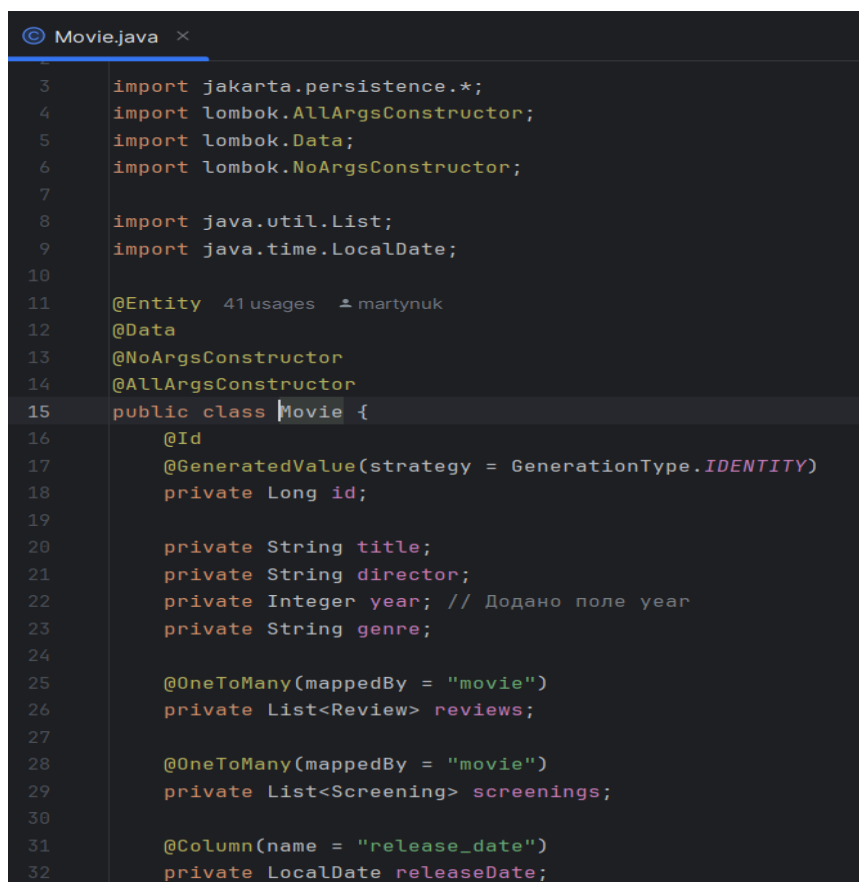
RewievController

- Створити новий жанр — @PostMapping
- Отримати всі жанри — @GetMapping
- Отримати жанр за ID — @GetMapping("/{id}")
- Оновити жанр за ID — @PutMapping("/{id}")
- Видалити жанр за ID — @DeleteMapping("/{id}")

3. Реалізація програмного продукту

У даному проєкті реалізовані такі моделі:

Movie — представляє фільм з ідентифікатором, назвою, режисером, роком випуску та жанром (рис. 3.1).



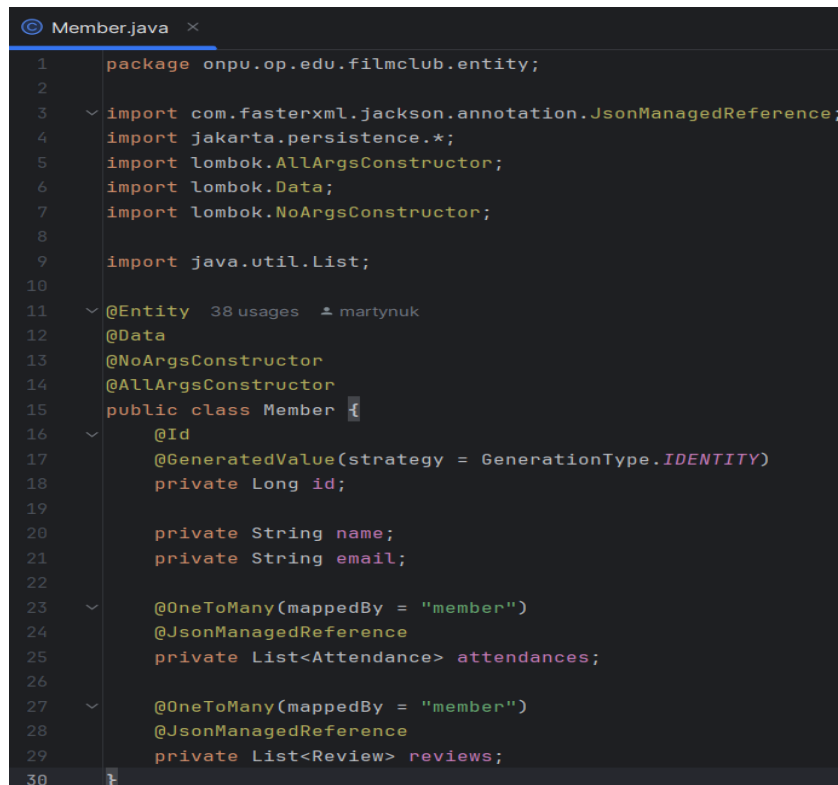
```

3      import jakarta.persistence.*;
4      import lombok.AllArgsConstructor;
5      import lombok.Data;
6      import lombok.NoArgsConstructor;
7
8      import java.util.List;
9      import java.time.LocalDate;
10
11     @Entity 41 usages 1 martynuk
12     @Data
13     @NoArgsConstructor
14     @AllArgsConstructor
15     public class Movie {
16         @Id
17         @GeneratedValue(strategy = GenerationType.IDENTITY)
18         private Long id;
19
20         private String title;
21         private String director;
22         private Integer year; // Додано поле year
23         private String genre;
24
25         @OneToMany(mappedBy = "movie")
26         private List<Review> reviews;
27
28         @OneToMany(mappedBy = "movie")
29         private List<Screening> screenings;
30
31         @Column(name = "release_date")
32         private LocalDate releaseDate;

```

Рисунок 3.1

Member — представляє учасника клубу з ідентифікатором, ім'ям та електронною поштою (рис. 3.2).



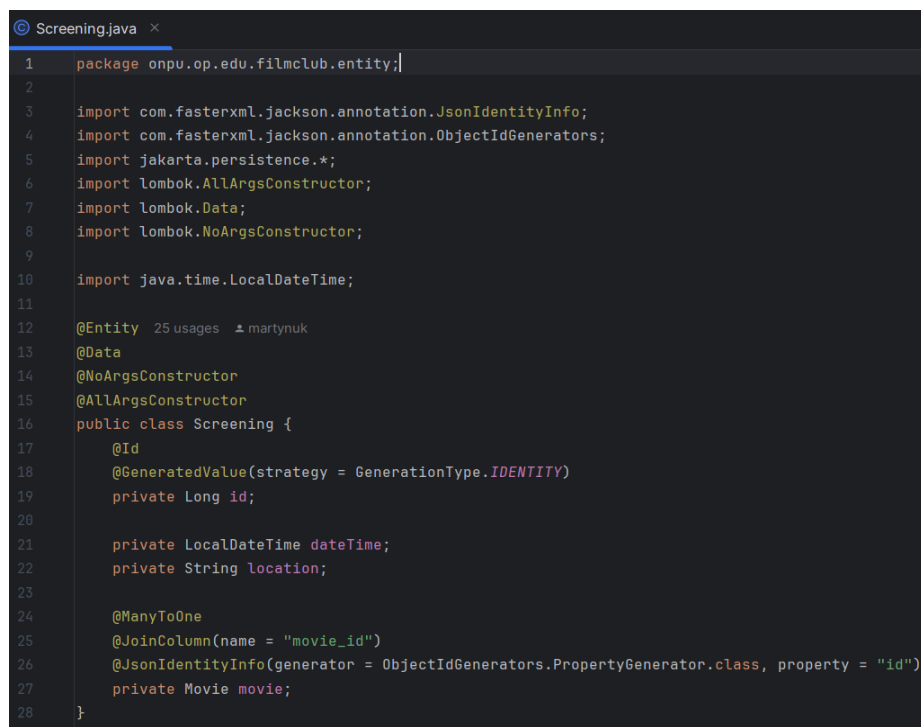
```

1 package onpu.op.edu.filmclub.entity;
2
3 import com.fasterxml.jackson.annotation.JsonManagedReference;
4 import jakarta.persistence.*;
5 import lombok.AllArgsConstructor;
6 import lombok.Data;
7 import lombok.NoArgsConstructor;
8
9 import java.util.List;
10
11 @Entity 38 usages  ▲ martynuk
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 public class Member {
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19
20     private String name;
21     private String email;
22
23     @OneToMany(mappedBy = "member")
24     @JsonManagedReference
25     private List<Attendance> attendances;
26
27     @OneToMany(mappedBy = "member")
28     @JsonManagedReference
29     private List<Review> reviews;
30 }

```

Рисунок 3.2

Screening — представляє показ фільму з ідентифікатором, посиланням на фільм, датою та місцем проведення (рис. 3.3).



```

1 package onpu.op.edu.filmclub.entity;
2
3 import com.fasterxml.jackson.annotation.JsonIdentityInfo;
4 import com.fasterxml.jackson.annotation.ObjectIdGenerators;
5 import jakarta.persistence.*;
6 import lombok.AllArgsConstructor;
7 import lombok.Data;
8 import lombok.NoArgsConstructor;
9
10 import java.time.LocalDateTime;
11
12 @Entity 25 usages  ▲ martynuk
13 @Data
14 @NoArgsConstructor
15 @AllArgsConstructor
16 public class Screening {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20
21     private LocalDateTime dateTime;
22     private String location;
23
24     @ManyToOne
25     @JoinColumn(name = "movie_id")
26     @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
27     private Movie movie;
28 }

```

Рисунок 3.3

Attendance — представляє присутність учасника на певному перегляді, містить зв'язки з Screening і Member (рис. 3.4).

```

Attendance.java x
1  package onpu.op.edu.filmclub.entity;
2
3  import com.fasterxml.jackson.annotation.JsonBackReference;
4  import com.fasterxml.jackson.annotation.JsonIdentityInfo;
5  import com.fasterxml.jackson.annotation.ObjectIdGenerators;
6  import jakarta.persistence.*;
7  import lombok.AllArgsConstructor;
8  import lombok.Data;
9  import lombok.NoArgsConstructor;
10
11  @Entity 22 usages martynuk
12  @Data
13  @NoArgsConstructor
14  @AllArgsConstructor
15  public class Attendance {
16      @Id
17      @GeneratedValue(strategy = GenerationType.IDENTITY)
18      private Long id;
19
20      private boolean attended;
21
22      @ManyToOne
23      @JoinColumn(name = "member_id")
24      @JsonBackReference
25      private Member member;
26
27      @ManyToOne
28      @JoinColumn(name = "screening_id")
29      @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
30      private Screening screening;
31  }

```

Рисунок 3.4

Review — рецензія, яку залишає учасник до фільму. Містить зв'язки з Member і Movie, оцінку та коментар (рис. 3.5).

```

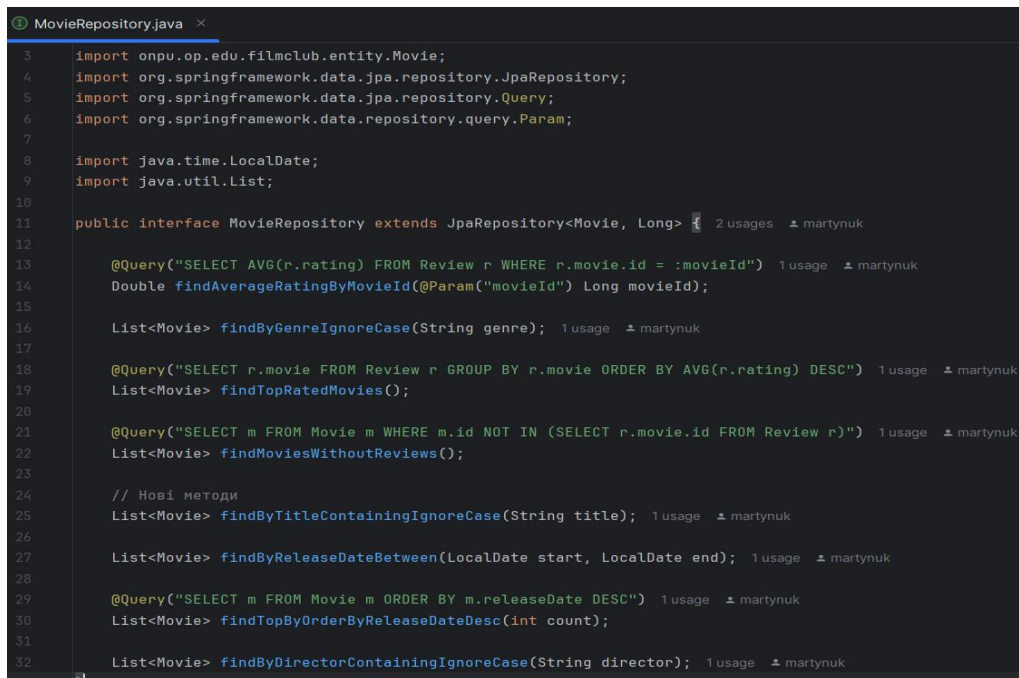
Review.java x
1  package onpu.op.edu.filmclub.entity;
2
3  import jakarta.persistence.*;
4  import lombok.AllArgsConstructor;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
7
8  @Entity 18 usages martynuk
9  @Data
10  @NoArgsConstructor
11  @AllArgsConstructor
12  public class Review {
13      @Id
14      @GeneratedValue(strategy = GenerationType.IDENTITY)
15      private Long id;
16
17      private Integer rating;
18      private String comment;
19
20      @ManyToOne
21      @JoinColumn(name = "member_id")
22      private Member member;
23
24      @ManyToOne
25      @JoinColumn(name = "movie_id")
26      private Movie movie;
27  }

```

Рисунок 3.5

Кожна з цих моделей має реалізовані репозиторії через Spring Data JPA для взаємодії з базою даних:

- **MovieRepository** — реалізує методи для збереження, оновлення, видалення та пошуку фільмів (рис. 3.6)



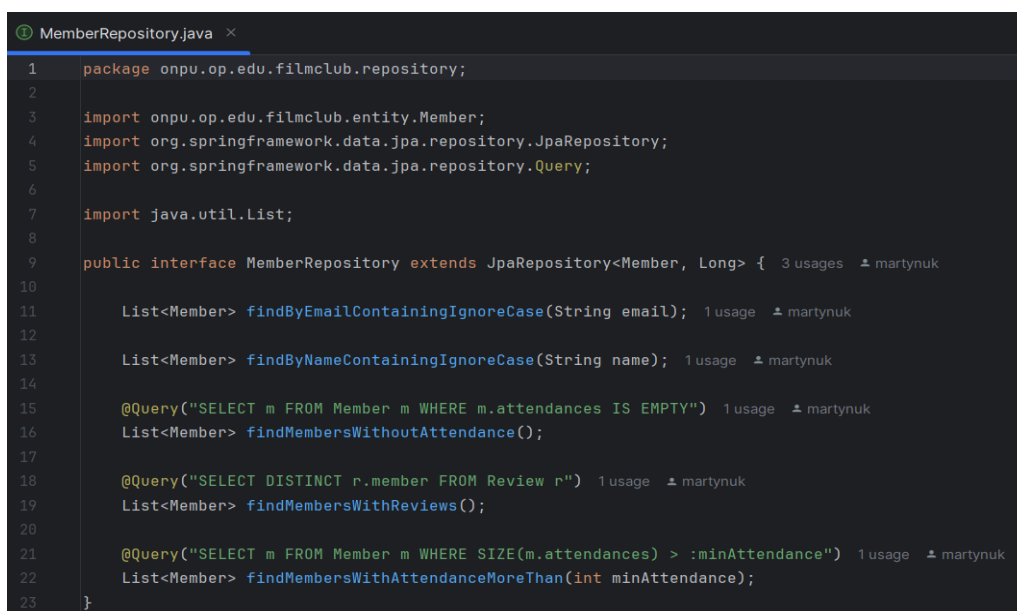
```

1  import onpu.op.edu.filmclub.entity.Movie;
2  import org.springframework.data.jpa.repository.JpaRepository;
3  import org.springframework.data.jpa.repository.Query;
4  import org.springframework.data.repository.query.Param;
5
6  import java.time.LocalDate;
7  import java.util.List;
8
9  public interface MovieRepository extends JpaRepository<Movie, Long> {
10
11     @Query("SELECT AVG(r.rating) FROM Review r WHERE r.movie.id = :movieId")
12     Double findAverageRatingByMovieId(@Param("movieId") Long movieId);
13
14     List<Movie> findByGenreIgnoreCase(String genre);
15
16     @Query("SELECT r.movie FROM Review r GROUP BY r.movie ORDER BY AVG(r.rating) DESC")
17     List<Movie> findTopRatedMovies();
18
19     @Query("SELECT m FROM Movie m WHERE m.id NOT IN (SELECT r.movie.id FROM Review r)")
20     List<Movie> findMoviesWithoutReviews();
21
22     // Нові методи
23     List<Movie> findByTitleContainingIgnoreCase(String title);
24
25     List<Movie> findByReleaseDateBetween(LocalDate start, LocalDate end);
26
27     @Query("SELECT m FROM Movie m ORDER BY m.releaseDate DESC")
28     List<Movie> findTopOrderByReleaseDateDesc(int count);
29
30     List<Movie> findByDirectorContainingIgnoreCase(String director);
31
32 }

```

Рисунок 3.6

- **MemberRepository** — забезпечує доступ до даних про учасників (рис. 3.7)



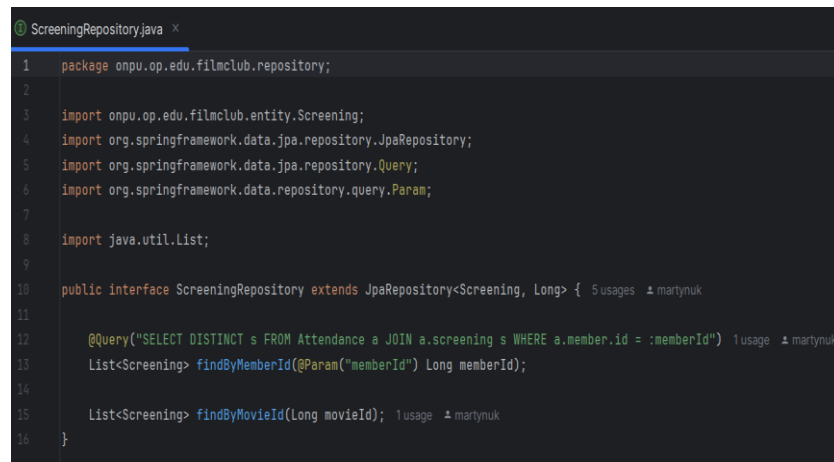
```

1  package onpu.op.edu.filmclub.repository;
2
3  import onpu.op.edu.filmclub.entity.Member;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.data.jpa.repository.Query;
6
7  import java.util.List;
8
9  public interface MemberRepository extends JpaRepository<Member, Long> {
10
11     List<Member> findByEmailContainingIgnoreCase(String email);
12
13     List<Member> findByNameContainingIgnoreCase(String name);
14
15     @Query("SELECT m FROM Member m WHERE m.attendances IS EMPTY")
16     List<Member> findMembersWithoutAttendance();
17
18     @Query("SELECT DISTINCT r.member FROM Review r")
19     List<Member> findMembersWithReviews();
20
21     @Query("SELECT m FROM Member m WHERE SIZE(m.attendances) > :minAttendance")
22     List<Member> findMembersWithAttendanceMoreThan(int minAttendance);
23 }

```

Рисунок 3.7

- ScreeningRepository — надає методи роботи з показами фільмів (рис. 3.8)



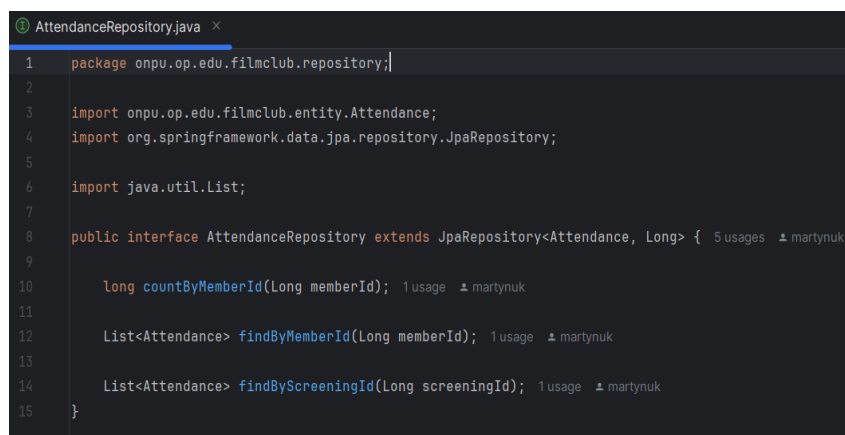
```

1 package onpu.op.edu.filmclub.repository;
2
3 import onpu.op.edu.filmclub.entity.Screening;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7
8 import java.util.List;
9
10 public interface ScreeningRepository extends JpaRepository<Screening, Long> { 5 usages  ⚡ martynuk
11
12     @Query("SELECT DISTINCT s FROM Attendance a JOIN a.screening s WHERE a.member.id = :memberId") 1 usage  ⚡ martynuk
13     List<Screening> findByMemberId(@Param("memberId") Long memberId);
14
15     List<Screening> findByMovieId(Long movieId); 1 usage  ⚡ martynuk
16 }

```

Рисунок 3.8

- AttendanceRepository — дозволяє працювати з інформацією про відвідування (рис. 3.9)



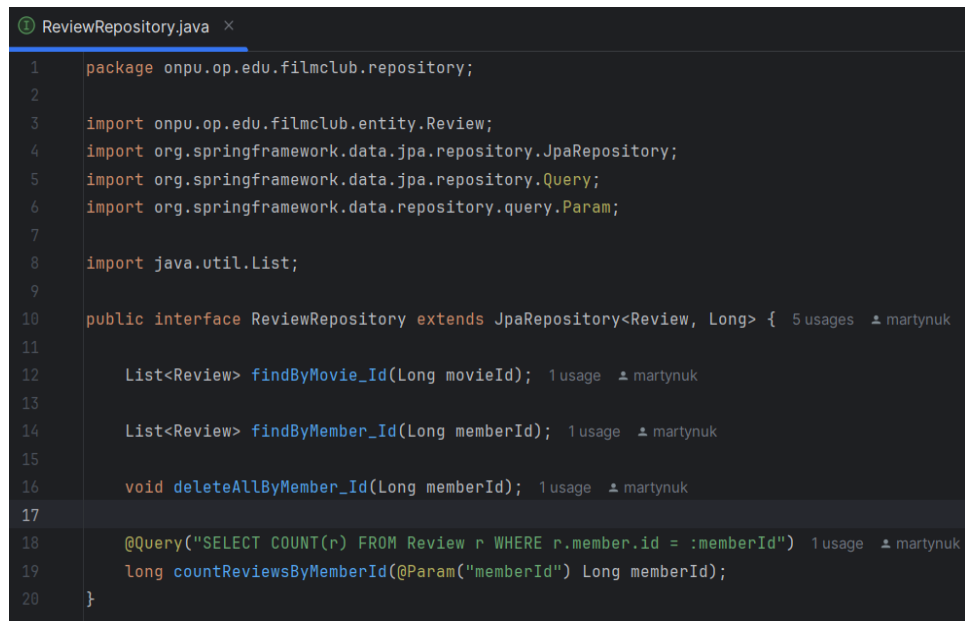
```

1 package onpu.op.edu.filmclub.repository;
2
3 import onpu.op.edu.filmclub.entity.Attendance;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.List;
7
8 public interface AttendanceRepository extends JpaRepository<Attendance, Long> { 5 usages  ⚡ martynuk
9
10     long countByMemberId(Long memberId); 1 usage  ⚡ martynuk
11
12     List<Attendance> findByMemberId(Long memberId); 1 usage  ⚡ martynuk
13
14     List<Attendance> findByScreeningId(Long screeningId); 1 usage  ⚡ martynuk
15 }

```

Рисунок 3.9

- ReviewRepository — забезпечує доступ до рецензій (рис. 3.10)



```

1 package onpu.op.edu.filmclub.repository;
2
3 import onpu.op.edu.filmclub.entity.Review;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.query.Param;
7
8 import java.util.List;
9
10 public interface ReviewRepository extends JpaRepository<Review, Long> { 5 usages ▲ martynuk
11
12     List<Review> findByMovie_Id(Long movieId); 1 usage ▲ martynuk
13
14     List<Review> findByMember_Id(Long memberId); 1 usage ▲ martynuk
15
16     void deleteAllByMember_Id(Long memberId); 1 usage ▲ martynuk
17
18     @Query("SELECT COUNT(r) FROM Review r WHERE r.member.id = :memberId") 1 usage ▲ martynuk
19     long countReviewsByMemberId(@Param("memberId") Long memberId);
20 }

```

Рисунок 3.10

Усі контролери розміщені в пакеті controller. Вони відповідають за обробку HTTP-запитів і делегують роботу відповідним сервісам.

- **MovieController** — обробляє запити, пов'язані з фільмами (CRUD, фільтрація за жанром, середній рейтинг, фільми без рецензій)

```
package onpu.op.edu.filmclub.controller;
```

```
import onpu.op.edu.filmclub.entity.Movie;
```

```
import onpu.op.edu.filmclub.service.MovieService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import java.time.LocalDate;
```

```
import java.util.List;
```

```
@RestController
```



```
@RequestMapping("/api/movies")
```

```
public class MovieController {
```

```
    @Autowired
```

```
    private MovieService movieService;
```

```
    @PostMapping
```

```
    public Movie createMovie(@RequestBody Movie movie) {
```

```
        return movieService.saveMovie(movie);
```

```
    }
```

```
    @GetMapping
```

```
    public List<Movie> getAllMovies() {
```

```
        return movieService.getAllMovies();
```

```
    }
```

```
    @GetMapping("/{id}")
```

```
    public Movie getMovieById(@PathVariable Long id) {
```

```
        return movieService.getMovieById(id);
```

```
    }
```

```
    @PutMapping("/{id}")
```

```
    public Movie updateMovie(@PathVariable Long id, @RequestBody Movie  
        movie) {
```

```
        movie.setId(id);
```

```
        return movieService.saveMovie(movie);
    }
```

```
@DeleteMapping("/{id}")
public void deleteMovie(@PathVariable Long id) {
    movieService.deleteMovie(id);
}
```

```
@GetMapping("/{id}/average-rating")
public Double getAverageRating(@PathVariable Long id) {
    return movieService.getAverageRating(id);
}
```

```
@GetMapping("/genre/{genre}")
public List<Movie> getMoviesByGenre(@PathVariable String genre) {
    return movieService.getMoviesByGenre(genre);
}
```

```
@GetMapping("/top-rated")
public List<Movie> getTopRatedMovies() {
    return movieService.getTopRatedMovies();
}
```

```
@GetMapping("/unreviewed")
```

```
public List<Movie> getUnreviewedMovies() {  
    return movieService.getUnreviewedMovies();  
}
```

```
@GetMapping("/search")
```

```
public List<Movie> searchMovies(@RequestParam String title) {  
    return movieService.searchMoviesByTitle(title);  
}
```

```
@GetMapping("/date-range")
```

```
public List<Movie> getMoviesInDateRange(@RequestParam String start,  
    @RequestParam String end) {  
    return movieService.getMoviesBetweenDates(LocalDate.parse(start),  
        LocalDate.parse(end));  
}
```

```
@GetMapping("/genres")
```

```
public List<String> getAllGenres() {  
    return movieService.getAllGenres();  
}
```

```
@GetMapping("/count")
```

```
public long countMovies() {  
    return movieService.countAllMovies();  
}
```

```

@GetMapping("/latest")

public List<Movie> getLatestMovies(@RequestParam(defaultValue = "5") int
count) {

    return movieService.getLatestMovies(count);

}

@GetMapping("/director/{director}")

public List<Movie> getMoviesByDirector(@PathVariable String director) {

    return movieService.getMoviesByDirector(director);

}
}

```

- MemberController — дозволяє створити, оновити, видалити учасника, отримати статистику по ньому.

```

package onpu.op.edu.filmclub.controller;

import onpu.op.edu.filmclub.dto.MemberDTO;
import onpu.op.edu.filmclub.dto.AttendanceDTO;
import onpu.op.edu.filmclub.entity.Attendance;
import onpu.op.edu.filmclub.entity.Member;
import onpu.op.edu.filmclub.service.MemberService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

```

```
import java.util.List;

import java.util.Map; // Додано імпорт для Map

import java.util.stream.Collectors;

@RestController

@RequestMapping("/api")

public class MemberController {

    private final MemberService memberService;

    @Autowired

    public MemberController(MemberService memberService) {

        this.memberService = memberService;

    }

    @PostMapping("/members")

    public Member createMember(@RequestBody Member member) {

        return memberService.saveMember(member);

    }

    @GetMapping("/members")

    public List<MemberDTO> getAllMembers() {

        List<Member> members = memberService.getAllMembers();

        return members.stream()
```

```

        .map(this::convertToDTO)
        .collect(Collectors.toList());
    }

```

```

@GetMapping("/members/{id}")
public Member getMemberById(@PathVariable Long id) {
    return memberService.getMemberById(id).orElse(null);
}

```

```

@PutMapping("/members/{id}")
public Member updateMember(@PathVariable Long id, @RequestBody
    Member member) {
    member.setId(id);
    return memberService.saveMember(member);
}

```

```

@DeleteMapping("/members/{id}")
public void deleteMember(@PathVariable Long id) {
    memberService.deleteMember(id);
}

```

```

@GetMapping("/members/{id}/screenings")
public List<AttendanceDTO> getMemberScreenings(@PathVariable Long id) {
    Member member = memberService.getMemberById(id)

```

```

        .orElseThrow(() -> new RuntimeException("Member not found with id: "
+ id));

    return member.getAttendances().stream()

        .map(this::convertToAttendanceDTO)

        .collect(Collectors.toList());
}

```

```

@GetMapping("/members/{id}/stats")

public Map<String, Long> getMemberStats(@PathVariable Long id) {

    return memberService.getMemberStats(id);

}

```

```

@GetMapping("/members/search/email")

public List<Member> searchMembersByEmail(@RequestParam String email) {

    return memberService.searchMembersByEmail(email);

}

```

```

@GetMapping("/members/search/name")

public List<Member> searchMembersByName(@RequestParam String name) {

    return memberService.searchMembersByName(name);

}

```

```

@GetMapping("/members/without-attendance")

public List<Member> getMembersWithoutAttendance() {

    return memberService.getMembersWithoutAttendance();

}

```

```
}
```

```
@GetMapping("/members/with-reviews")
```

```
public List<Member> getMembersWithReviews() {
    return memberService.getMembersWithReviews();
}
```

```
@GetMapping("/members/active")
```

```
public List<Member> getActiveMembers(@RequestParam int minAttendance) {
    return memberService.getActiveMembers(minAttendance);
}
```

```
private MemberDTO convertToDTO(Member member) {
    return new MemberDTO(
        member.getId(),
        member.getName(),
        member.getEmail(),
        member.getAttendances().stream()
            .map(this::convertToAttendanceDTO)
            .collect(Collectors.toList())
    );
}
```

```
private AttendanceDTO convertToAttendanceDTO(Attendance attendance) {
```



```

        return new AttendanceDTO(
            attendance.getId(),
            attendance.isAttended(),
            attendance.getScreening() != null ? attendance.getScreening().getId() :
            null
        );
    }
}

```

- `ScreeningController` — реалізує логіку створення, отримання переглядів, пошуку за фільмом або учасником.

```
package onpu.op.edu.filmclub.controller;
```

```
import onpu.op.edu.filmclub.entity.Screening;
```

```
import onpu.op.edu.filmclub.service.ScreeningService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
```

```
@RestController
```

```
@RequestMapping("/api/screenings")
```

```
public class ScreeningController {
```

```
    @Autowired
```

```
private ScreeningService screeningService;
```

```
@PostMapping
```

```
public Screening createScreening(@RequestBody Screening screening) {  
    return screeningService.saveScreening(screening);  
}
```

```
@GetMapping
```

```
public List<Screening> getAllScreenings() {  
    return screeningService.getAllScreenings();  
}
```

```
@GetMapping("/{id}")
```

```
public Screening getScreeningById(@PathVariable Long id) {  
    return screeningService.getScreeningById(id);  
}
```

```
@GetMapping("/movie/{movieId}")
```

```
public List<Screening> getScreeningsByMovieId(@PathVariable Long  
movieId) {  
    return screeningService.getScreeningsByMovieId(movieId);  
}
```

```
@DeleteMapping("/{id}")
```

```
public void deleteScreening(@PathVariable Long id) {
```

```

        screeningService.deleteScreening(id);
    }

    @PutMapping("/{id}")
    public Screening updateScreening(@PathVariable Long id, @RequestBody
        Screening screening) {
        screening.setId(id);
        return screeningService.saveScreening(screening);
    }
}

```

- AttendanceController — додає/видаляє участь у перегляді, а також виводить список присутніх.

```

package onpu.op.edu.filmclub.controller;

import onpu.op.edu.filmclub.entity.Attendance;
import onpu.op.edu.filmclub.service.AttendanceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List; // Додано імпорт для List

@RestController
@RequestMapping("/api/attendances")
public class AttendanceController {

```

@Autowired

private AttendanceService attendanceService;

@PostMapping

```
public Attendance createAttendance(@RequestBody Attendance attendance) {  
    return attendanceService.saveAttendance(attendance);  
}
```

@DeleteMapping("/{id}")

```
public void deleteAttendance(@PathVariable Long id) {  
    attendanceService.deleteAttendance(id);  
}
```

@GetMapping("/member/{memberId}")

```
public List<Attendance> getAttendancesByMember(@PathVariable Long  
memberId) {  
    return attendanceService.getAttendancesByMember(memberId);  
}
```

@GetMapping("/screening/{screeningId}")

```
public List<Attendance> getAttendancesByScreening(@PathVariable Long  
screeningId) {  
    return attendanceService.getAttendancesByScreening(screeningId);  
}
```

```
}
```

- `ReviewController` — дозволяє додати, оновити, видалити рецензію, переглянути оцінки фільму та учасника.

```
package onpu.op.edu.filmclub.controller;
```

```
import onpu.op.edu.filmclub.entity.Review;
```

```
import onpu.op.edu.filmclub.service.ReviewService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
```

```
@RestController
```

```
@RequestMapping("/api/reviews")
```

```
public class ReviewController {
```

```
    @Autowired
```

```
    private ReviewService reviewService;
```

```
    @PostMapping
```

```
    public Review createReview(@RequestBody Review review) {
```

```
        return reviewService.saveReview(review);
```

```
    }
```

```
    @GetMapping("/movie/{movieId}")
```

```
    public List<Review> getReviewsByMovie(@PathVariable Long movieId) {
```

```
        return reviewService.getReviewsByMovie(movieId);
```

```
}
```

```
@GetMapping("/member/{memberId}")
```

```
public List<Review> getReviewsByMember(@PathVariable Long memberId) {
    return reviewService.getReviewsByMember(memberId);
}
```

```
@PutMapping("/{id}")
```

```
public Review updateReview(@PathVariable Long id, @RequestBody Review
review) {
    review.setId(id);
    return reviewService.saveReview(review);
}
```

```
@DeleteMapping("/{id}")
```

```
public void deleteReview(@PathVariable Long id) {
    reviewService.deleteReview(id);
}
```

```
@DeleteMapping("/member/{memberId}")
```

```
public void deleteReviewsByMember(@PathVariable Long memberId) {
    reviewService.deleteReviewsByMemberId(memberId);
}
}
```

Усі сервіси зберігаються в пакеті `service`. Вони містять бізнес-логіку:

- `MovieService` — обробка даних про фільми: створення, оновлення, видалення, фільтрація.

```
package onpu.op.edu.filmclub.service;

import onpu.op.edu.filmclub.entity.Movie;
import onpu.op.edu.filmclub.repository.MovieRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class MovieService {

    @Autowired
    private MovieRepository movieRepository;

    public Movie saveMovie(Movie movie) {
        return movieRepository.save(movie);
    }

    public List<Movie> getAllMovies() {
        return movieRepository.findAll();
    }
}
```

```
public Movie getMovieById(Long id) {  
    return movieRepository.findById(id).orElse(null);  
}  
  
public void deleteMovie(Long id) {  
    movieRepository.deleteById(id);  
}  
  
public Double getAverageRating(Long movieId) {  
    return movieRepository.findAverageRatingByMovieId(movieId);  
}  
  
public List<Movie> getMoviesByGenre(String genre) {  
    return movieRepository.findByGenreIgnoreCase(genre);  
}  
  
public List<Movie> getTopRatedMovies() {  
    return movieRepository.findTopRatedMovies();  
}  
  
public List<Movie> getUnreviewedMovies() {  
    return movieRepository.findMoviesWithoutReviews();  
}
```


// Нові методи

```
public List<Movie> searchMoviesByTitle(String title) {  
    return movieRepository.findByTitleContainingIgnoreCase(title);  
}
```

```
public List<Movie> getMoviesBetweenDates(LocalDate start, LocalDate end) {  
    return movieRepository.findByReleaseDateBetween(start, end);  
}
```

```
public List<String> getAllGenres() {  
    return movieRepository.findAll().stream()  
        .map(Movie::getGenre)  
        .distinct()  
        .collect(Collectors.toList());  
}
```

```
public long countAllMovies() {  
    return movieRepository.count();  
}
```

```
public List<Movie> getLatestMovies(int count) {  
    return movieRepository.findTopOrderByReleaseDateDesc(count);  
}
```

```

public List<Movie> getMoviesByDirector(String director) {
    return movieRepository.findByDirectorContainingIgnoreCase(director);
}
}

```

- **MemberService** — реалізує логіку CRUD учасників, а також обчислення персональної статистики.

```
package onpu.op.edu.filmclub.service;
```

```

import onpu.op.edu.filmclub.entity.Member;
import onpu.op.edu.filmclub.entity.Screening;
import onpu.op.edu.filmclub.repository.MemberRepository;
import onpu.op.edu.filmclub.repository.ScreeningRepository;
import onpu.op.edu.filmclub.repository.ReviewRepository;
import onpu.op.edu.filmclub.repository.AttendanceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;

```

```
@Service
```

```
public class MemberService {

    private final MemberRepository memberRepository;
    private final ScreeningRepository screeningRepository;
    private final AttendanceRepository attendanceRepository;
    private final ReviewRepository reviewRepository;

    @Autowired
    public MemberService(MemberRepository memberRepository,
                          ScreeningRepository screeningRepository,
                          AttendanceRepository attendanceRepository,
                          ReviewRepository reviewRepository) {
        this.memberRepository = memberRepository;
        this.screeningRepository = screeningRepository;
        this.attendanceRepository = attendanceRepository;
        this.reviewRepository = reviewRepository;
    }

    public Member saveMember(Member member) {
        return memberRepository.save(member);
    }

    public List<Member> getAllMembers() {
        return memberRepository.findAll();
    }
}
```

```
}
```

```
public Optional<Member> getMemberById(Long id) {
    return memberRepository.findById(id);
}
```

```
public void deleteMember(Long id) {
    memberRepository.deleteById(id);
}
```

```
public List<Screening> getScreeningsForMember(Long memberId) {
    return screeningRepository.findByMemberId(memberId);
}
```

```
public Map<String, Long> getMemberStats(Long memberId) {
    long reviews = reviewRepository.countReviewsByMemberId(memberId);
    long visits = attendanceRepository.countByMemberId(memberId);
    Map<String, Long> stats = new HashMap<>();
    stats.put("reviewCount", reviews);
    stats.put("attendanceCount", visits);
    return stats;
}
```

```
public List<Member> searchMembersByEmail(String email) {
```

```

        return memberRepository.findByEmailContainingIgnoreCase(email);
    }

```

```

public List<Member> searchMembersByName(String name) {
    return memberRepository.findByNameContainingIgnoreCase(name);
}

```

```

public List<Member> getMembersWithoutAttendance() {
    return memberRepository.findMembersWithoutAttendance();
}

```

```

public List<Member> getMembersWithReviews() {
    return memberRepository.findMembersWithReviews();
}

```

```

public List<Member> getActiveMembers(int minAttendance) {
    return
        memberRepository.findMembersWithAttendanceMoreThan(minAttendance);
}
}

```

- ScreeningService — створює нові перегляди, надає список усіх або за фільмом/учасником.

```

package onpu.op.edu.filmclub.service;

```

```
import onpu.op.edu.filmclub.entity.Screening;

import onpu.op.edu.filmclub.repository.ScreeningRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;


import java.util.List;


@Service

public class ScreeningService {


    @Autowired

    private ScreeningRepository screeningRepository;


    public Screening saveScreening(Screening screening) {

        return screeningRepository.save(screening);

    }


    public List<Screening> getAllScreenings() {

        return screeningRepository.findAll();

    }


    public Screening getScreeningById(Long id) {

        return screeningRepository.findById(id).orElse(null);

    }

}
```

```

public void deleteScreening(Long id) {
    screeningRepository.deleteById(id);
}

public List<Screening> getScreeningsByMovieId(Long movieId) {
    return screeningRepository.findByMovieId(movieId);
}
}


```

- `AttendanceService` — дозволяє зареєструвати або скасувати участь, перевірити список учасників перегляду.

```

package onpu.op.edu.filmclub.service;

import onpu.op.edu.filmclub.entity.Attendance;
import onpu.op.edu.filmclub.repository.AttendanceRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class AttendanceService {

    @Autowired

```

```
private AttendanceRepository attendanceRepository;
```

```
public Attendance saveAttendance(Attendance attendance) {
    return attendanceRepository.save(attendance);
}
```

```
public void deleteAttendance(Long id) {
    attendanceRepository.deleteById(id);
}
```

```
public List<Attendance> getAttendancesByMember(Long memberId) {
    return attendanceRepository.findByMemberId(memberId);
}
```

```
public List<Attendance> getAttendancesByScreening(Long screeningId) {
    return attendanceRepository.findByScreeningId(screeningId);
}
}
```

- ReviewService — працює з рецензіями: створення, редагування, оцінка, пошук.

```
package onpu.op.edu.filmclub.service;
```

```
import onpu.op.edu.filmclub.entity.Review;
```

```
import onpu.op.edu.filmclub.repository.ReviewRepository;
```



```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ReviewService {

    @Autowired

    private ReviewRepository reviewRepository;

    public Review saveReview(Review review) {
        return reviewRepository.save(review);
    }

    public List<Review> getReviewsByMovie(Long movieId) {
        return reviewRepository.findByMovie_Id(movieId);
    }

    public List<Review> getReviewsByMember(Long memberId) {
        return reviewRepository.findByMember_Id(memberId);
    }

    public void deleteReview(Long id) {
```

```

        reviewRepository.deleteById(id);
    }

    public void deleteReviewsByMemberId(Long memberId) {
        reviewRepository.deleteAllByMember_Id(memberId);
    }
}

```

Кожен сервіс взаємодіє зі своїм Repository, який реалізовано на базі Spring Data JPA.

Безпека та User:

У пакеті config реалізовано клас SecurityConfig, який:

- Визначає шляхи, що вимагають авторизації (/api/**)
- Дозволяє публічний доступ до /auth/**
- Використовує JWT-фільтр для захисту запитів
- Інтегрується з Spring Security

```
package onpu.op.edu.filmclub.config;
```

```

import onpu.op.edu.filmclub.entity.AppUser;
import onpu.op.edu.filmclub.repository.AppUserRepository;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;

```

```
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.config.annotation.authentication.configuration.Authentic
ationConfiguration;
```

```
import java.util.ArrayList;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig {
```

```
    @Bean
```

```
    public BCryptPasswordEncoder passwordEncoder() {
```

```
        return new BCryptPasswordEncoder();
```

```
    }
```

```
    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
    {
```

```
        http
```

```
            .authorizeHttpRequests((requests) -> requests
```

```
                .requestMatchers("/api/register", "/api/login", "/api/movies/**",
"/api/members/**").permitAll()
```

```
                .anyRequest().authenticated()
```

```
            )
```

```
            .csrf(csrf -> csrf.disable())
```

```

        .formLogin(form -> form.disable());
    return http.build();
}

```

@Bean

```

public UserDetailsService userDetailsService(AppUserRepository
appUserRepository) {
    return username -> {
        AppUser appUser = appUserRepository.findByUsername(username);
        if (appUser == null) {
            throw new
org.springframework.security.core.userdetails.UsernameNotFoundException("User
not found");
        }
        return new org.springframework.security.core.userdetails.User(
            appUser.getUsername(), appUser.getPassword(), new ArrayList<>());
    };
}

```

@Bean

```

public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
    return authenticationConfiguration.getAuthenticationManager();
}
}

```

AppUser:

```

package onpu.op.edu.filmclub.entity;

```

```

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Entity
@Data
public class AppUser {

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

    @Column(unique = true, nullable = false)
    @NotBlank(message = "Username is required")
    private String username;

    @Column(nullable = false)
    @NotBlank(message = "Password is required")
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Role role;
}

```

Role do User:

```
package onpu.op.edu.filmclub.entity;
```

```
public enum Role {
```

```
    USER,
```

```
    ADMIN
```

```
}
```

4. Тестування та налагодження

Мета тестування системи полягала у перевірці правильності функціоналу, виявленні логічних і технічних помилок, а також у перевірці взаємодії між окремими компонентами проєкту. Особливу увагу було приділено валідації вхідних та вихідних даних, що обробляються в системі. Основна частина тестування була зосереджена на REST API-запитах, які реалізують доступ до функцій системи управління кіноклубом.

Для перевірки коректності роботи API використовувався інструмент Postman, що дозволяє виконувати HTTP-запити, працювати з авторизацією та аналізувати відповіді сервера. Завдяки JWT-аутентифікації, що була реалізована у проєкті, доступ до захищених ендпоінтів можливий лише після входу в систему. Після успішної авторизації система видає токен, який додається до заголовка кожного наступного запиту.

Спершу підключимо та перевіримо юзера (рис. 4.1).

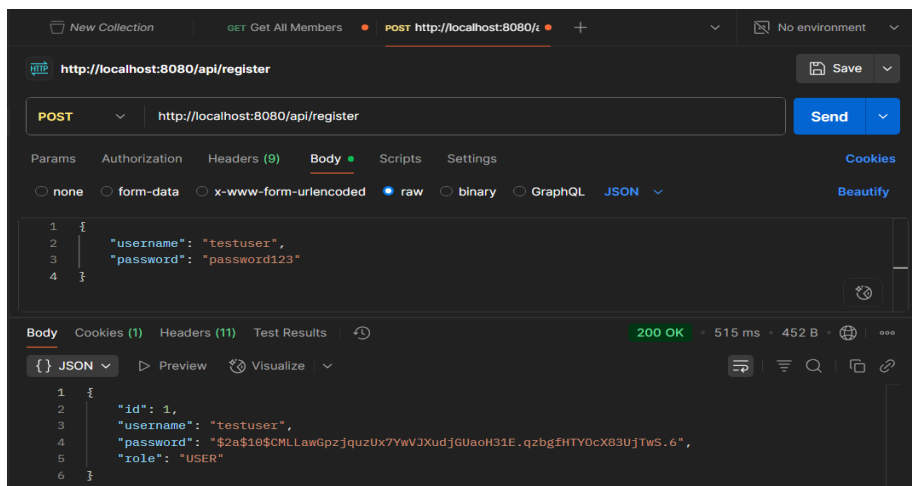


Рисунок 4.1

Отримати всі фільми (рис. 4.2).

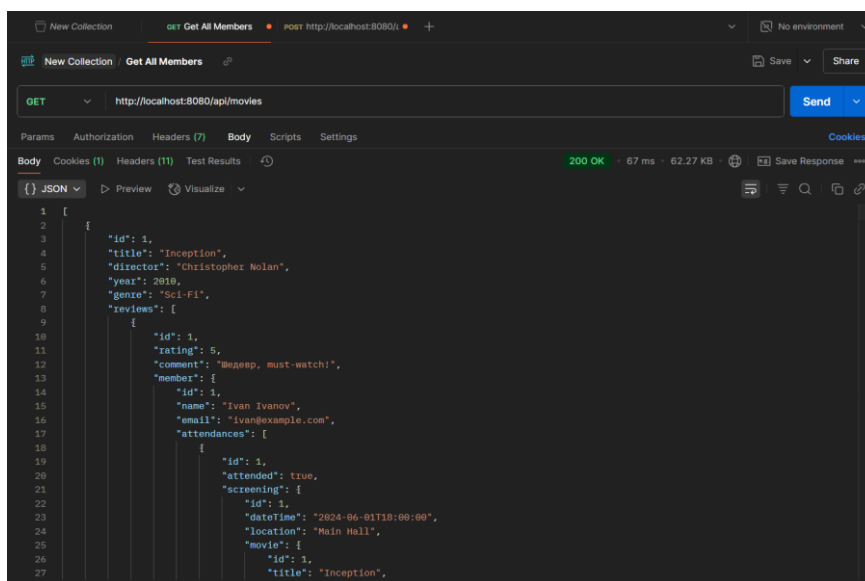


Рисунок 4.2

Наступний запит дозволить отримати фільм за ID (рис. 4.3).

/api/movies/{id}

Результат запиту:

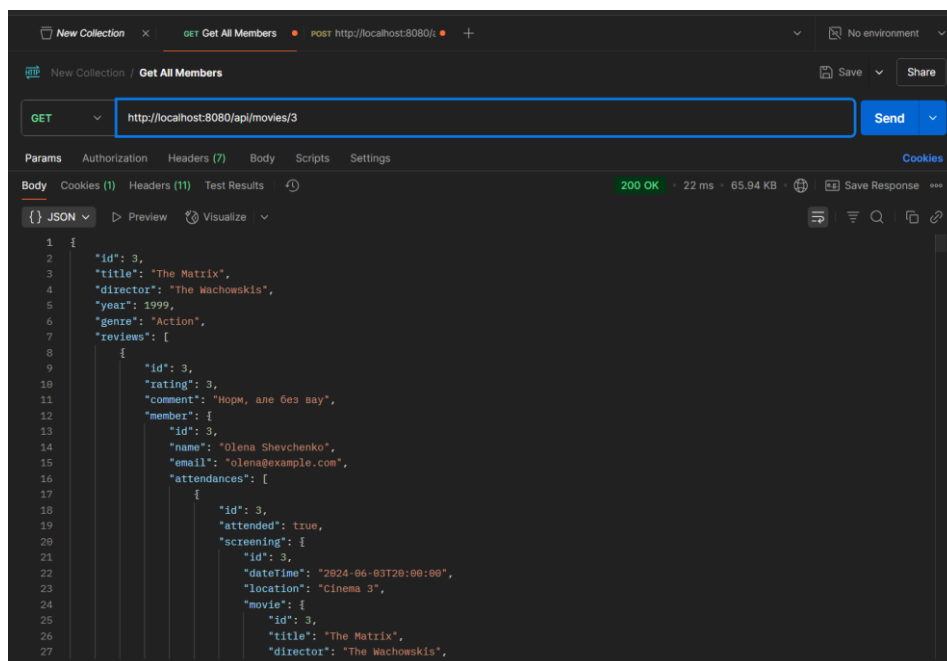


Рисунок 4.3

Отримати середній рейтинг фільму (рис. 4.4).

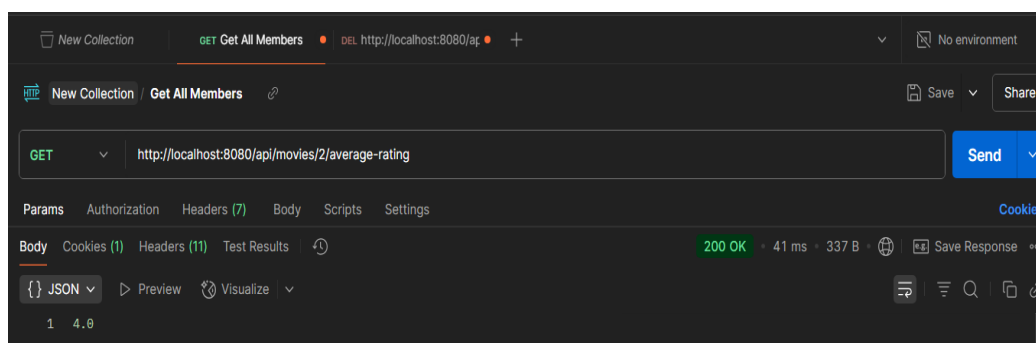


Рисунок 4.4

Фільми за жанром (рис. 4.5):

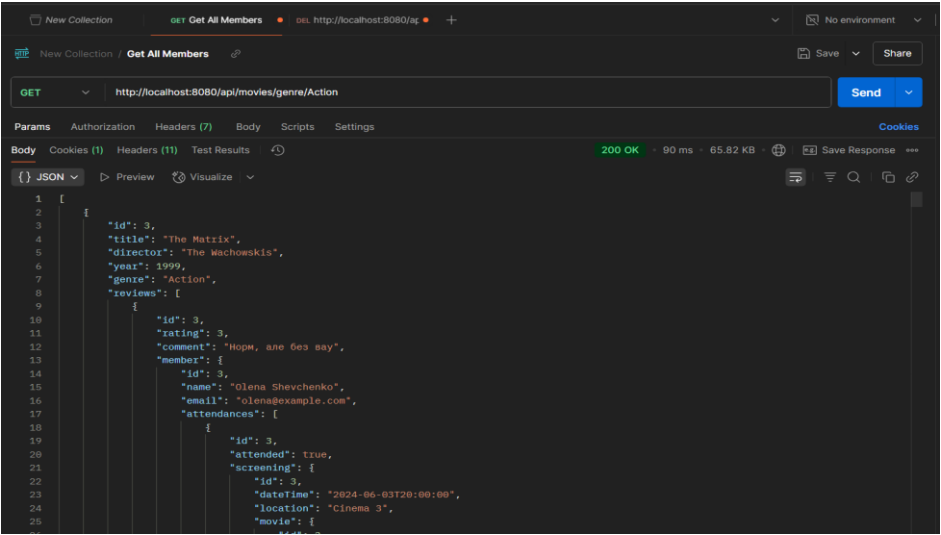


Рисунок 4.5

Топові фільми (рис. 4.6).

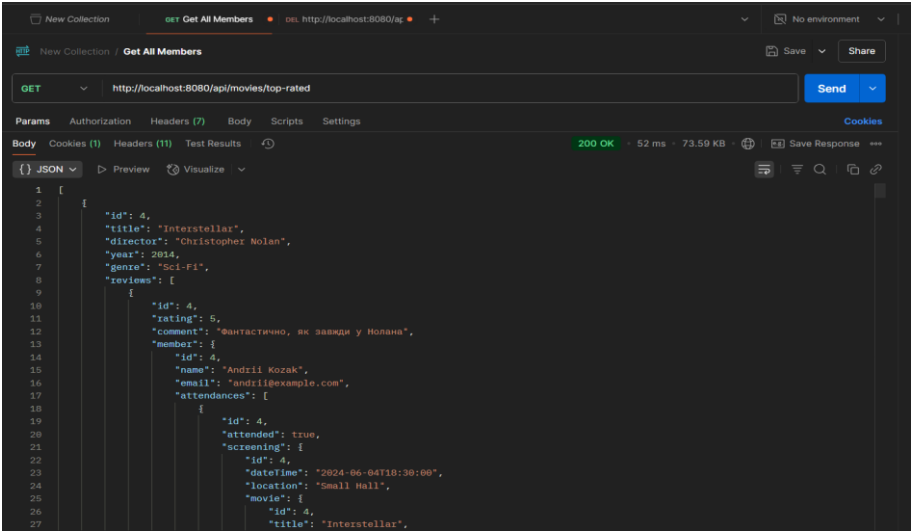


Рисунок 4.6

Фільми без рецензій (рис.4.7):

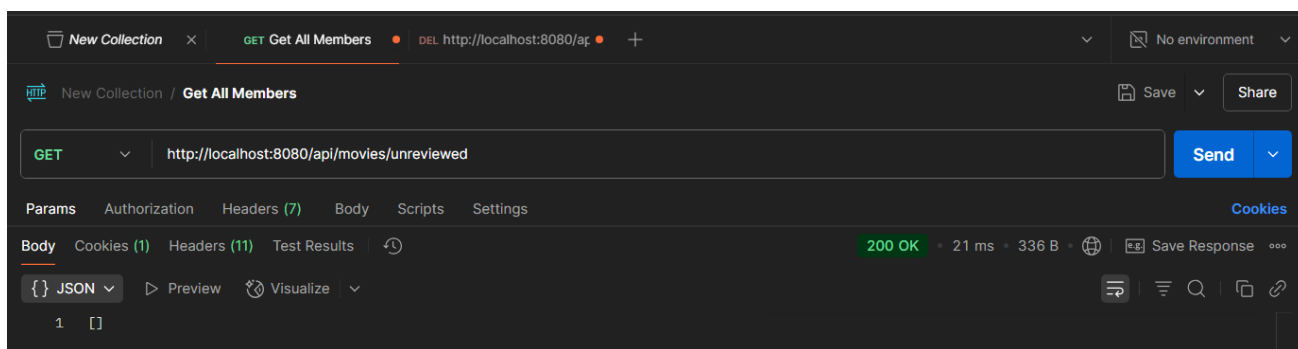


Рисунок 4.7

Виводить пусту тому що у всіх є рецензія.

Пошук за назвою (рис.4.8).

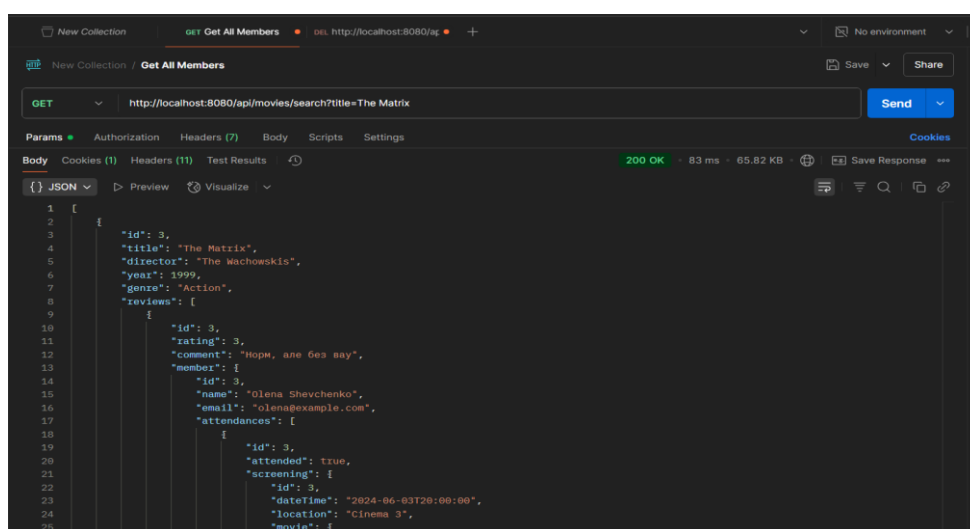


Рисунок 4.8

Отримати всіх учасників (рис.4.9).

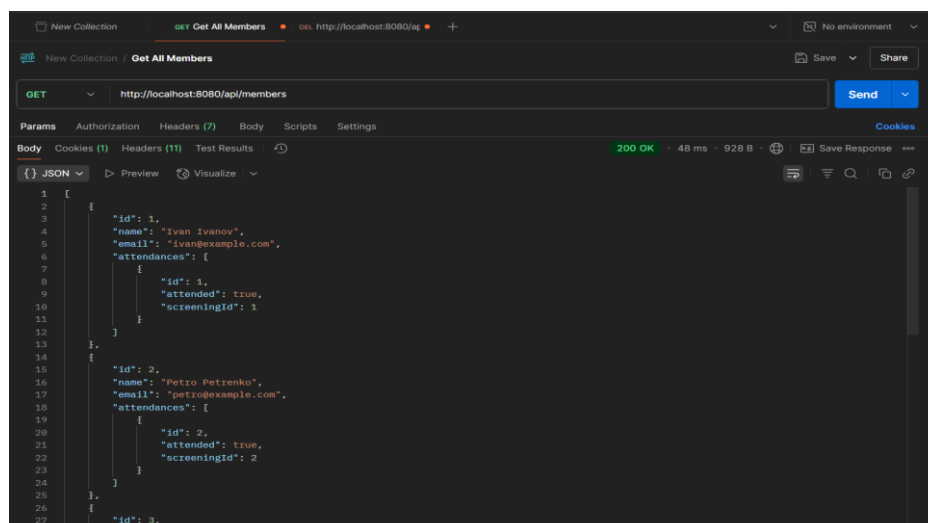


Рисунок 4.9

Пошук за email (рис. 4.10)

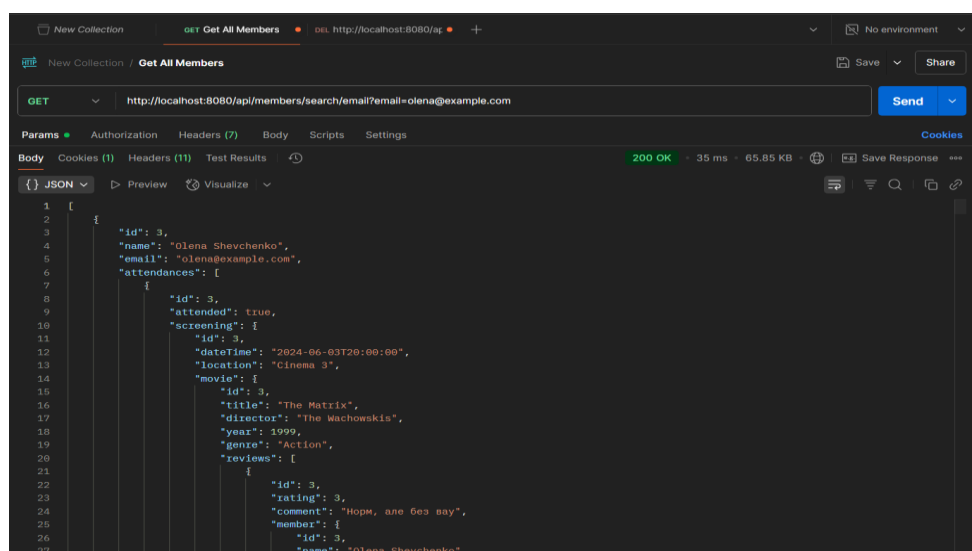


Рисунок 4.10

Висновок

Під час виконання курсової роботи було реалізовано повноцінний вебзастосунок на базі Spring Boot, що дозволяє автоматизувати основні процеси роботи кіноклубу. Основною метою проєкту було створення RESTful API для управління сутностями, такими як фільми, учасники, перегляди, рецензії та участь у заходах, а також забезпечення їх взаємодії через HTTP-запити.

У ході роботи були виконані наступні ключові етапи:

- створено реляційну базу даних із сутностями Movie, Member, Screening, Attendance, Review та відповідними зв'язками між ними;
- реалізовано понад 25 REST-запитів, які охоплюють CRUD-операції, фільтрацію, отримання статистики, рейтингів та зв'язків між сутностями;
- впроваджено механізм автентифікації та авторизації з використанням JWT-токенів, а також інтеграцію з Google через OAuth2;
- протестовано всі основні функції системи за допомогою Postman.

У результаті розробки було створено функціональний серверний застосунок, що демонструє принципи об'єктно-орієнтованого програмування, архітектуру “контролер-сервіс-репозиторій”, використання DTO, а також дотримання принципів безпечної взаємодії з клієнтом.

Підсумовуючи, дана курсова робота дозволила закріпити знання у сфері Java-розробки, Spring Boot, REST API, роботи з базами даних PostgreSQL, а також

розвинути практичні навички у сфері backend-програмування. Проєкт є прикладом сучасного підходу до цифровізації та автоматизації процесів у галузі культури та дозвілля.

Список використаних джерел

- Spring Boot Reference Documentation. URL: <https://docs.spring.io/spring-boot/docs/current/reference/html/> (звернення: 12.06.2025).
- Java SE Documentation. URL: <https://docs.oracle.com/en/java/javase/17/> (звернення: 13.06.2025).
- PostgreSQL: The world's most advanced open source database. URL: <https://www.postgresql.org/docs/> (звернення: 13.06.2025).
- REST API Tutorial. URL: <https://restfulapi.net/> (звернення: 14.06.2025).
- Docker Documentation. URL: <https://docs.docker.com/> (звернення: 14.06.2025).

Додаток

Посилання на Github де розміщений повний код проекту :
<https://github.com/SergeyDob/filmclub.git>