BMT-308
WEB PROGRAMLAMA
SPRING PROJE RAPORU

Film Arşiv Uygulama Raporu

Hazırlayanlar:

Sena Nur Türkmen	-	22181616006
Ecemnur Sünnetcioğlu	-	22181617007
İrem Şaziye Koç	-	22181616020
Şura Özkan	-	22181616503
Ceyda Emre	-	201816707
Gülse Kayıkcı	-	22181616014

Dersin Öğretmeni : Sinan Toklu

BMT-308
WEB PROGRAMLAMA
SPRING PROJE RAPORU

İçindekiler

1. Giriş 4
2. Kullanılan Yöntem ve Teknolojiler 4
    2.1. Yazılım Geliştirme Yaklaşımı 4
    2.2. Kullanılan Teknolojiler 5
    2.3. Proje Yapısı 5
    2.4. Veritabanı Yapılandırması 7
    2.5. Proje Bağımlılıkları 7
3. Sistem Mimarisi ve Tasarımı 7
    3.1. Katmanlı Mimari Yapı 7
    3.2. MVC Mimarisi 7
    3.3. Sınıflar Arası İlişki Diyagramı 8
    3.4. Spring Security Mimarisi 8
    3.5. Görsel Saklama Yaklaşımı 8
4. Veritabanı Tasarımı 9
    4.1. Tablolar ve Alanlar 9
5. Kullanıcı Rolleri ve Yetkilendirme 10
    5.1. Kullanıcı Rolleri 10
    5.2. Kimlik Doğrulama 10
    5.3. Yetkilendirme 10
    5.4. Başarılı Giriş ve Yönlendirme 11
    5.5. Güvenli Oturum Yönetimi 11
    5.6. HTML'de Rol Bazlı Görünüm Kontrolü 11
6. Kodların Açıklamalı İncelemesi 12
    6.1. Entity Sınıfları 12
        6.1.1. User.java 13
        6.1.2. Film.java 14
        6.1.2. Comment.java 15
        6.1.2. Rating.java 15
    6.2. Repository 16
        6.2.1. UserRepository.java 16
        6.2.1. FilmRepository.java 16
        6.2.1. CommentRepository.java 17
        6.2.1. RatingRepository.java 17
    6.3. Service 17
        6.3.1. UserService.java 18
        6.3.2. FilmService.java 18
        6.3.3. CommentService.java 19
        6.3.4. RatingService.java 19
    6.4. Controller 20
        6.4.1. AppController.java 20

6.5. Yapılandırma (Config) Sınıfları 27
6.5.1. SecurityConfig.java 27
7. Gerçekleştirilen Fonksiyonlar ve Ekranlar 30
7.1. Kayıt Olma (Register) 30
7.2. Giriş Yapma (Login) 30
7.3. Film Listeleme (Ana Sayfa) 30
7.4. Film Detayı 31
7.5. Yorum Ekleme 31
7.6. Puan Verme 32
7.7. Film Ekleme (Admin Paneli) 32
7.8. Film Güncelleme (Admin Paneli) 34
7.9. Film Silme 35
7.10. Oturumdan Çıkış (Logout) 35
8. Sonuç ve Değerlendirme 35

BMT-308
WEB PROGRAMLAMA
SPRING PROJE RAPORU

1. Giriş

Bu proje, kullanıcıların sistem üzerinde kayıt olup giriş yapabildiği, filmleri görüntüleyip detaylarını inceleyebildiği, yorum ve puan verebildiği; aynı zamanda yönetici rolüne sahip kullanıcıların sistemde film ekleme, güncelleme ve silme gibi işlemleri gerçekleştirebildiği, role dayalı bir yetkilendirme altyapısına sahip film arşivleme sistemidir.

Uygulama, Java programlama dili ve Spring Boot çatısı kullanılarak geliştirilmiş olup, veritabanı olarak MySQL tercih edilmiştir. Sistem; kullanıcı arayüzü için Thymeleaf şablon motoru ve Bootstrap kütüphanesi ile desteklenmiş, kullanıcı deneyimini artırmak amacıyla dinamik ve responsive bir tasarımla yapılandırılmıştır. Güvenlik katmanında ise Spring Security kullanılarak kimlik doğrulama ve yetkilendirme işlemleri gerçekleştirilmiştir.

2. Kullanılan Yöntem ve Teknolojiler

2.1. Yazılım Geliştirme Yaklaşımı

Controller Katmanı: Kullanıcının taleplerini alır ve ilgili servis katmanına yönlendirir.

Service Katmanı: İş mantığının yürütüldüğü katmandır. Doğrudan veri erişimi içermez.

Repository Katmanı: Veritabanı ile etkileşimi sağlayan bileşenlerdir.

Config Katmanı: Güvenlik ve başlangıç konfigürasyonları burada tanımlanmıştır.

View Katmanı: Kullanıcıya sunulan HTML içerikleri burada yer alır.

2.2. Kullanılan Teknolojiler

Bileşen / Katman	Teknoloji / Araç
Backend Framework	Spring Boot
Web Katmanı	Spring MVC
ORM & Veritabanı	Spring Data JPA, Hibernate
Veritabanı	MySQL
View (Şablon Motoru)	Thymeleaf
UI Tasarımı	Bootstrap 5
Kimlik Doğrulama	Spring Security
Dosya Yükleme	MultipartFile (Spring)
Görsel Saklama	byte[] veri tipi
Şifreleme Algoritması	BCryptPasswordEncoder
Geliştirme Ortamı	IntelliJ IDEA / Spring Tools
Versiyon Kontrolü	Git, GitHub
Bağımlılık Yönetimi	Maven

2.3. Proje Yapısı

```
com.example.filmarchive
├── config
│   ├── DataInitializer.java
│   └── SecurityConfig.java
│
├── controller
│   └── AppController.java
│
├── entity
│   ├── User.java
│   ├── Film.java
│   ├── Comment.java
```

```
│   └── Rating.java
│
├── repository
│   ├── UserRepository.java
│   ├── FilmRepository.java
│   ├── CommentRepository.java
│   └── RatingRepository.java
│
├── service
│   ├── UserService.java
│   ├── FilmService.java
│   ├── CommentService.java
│   └── RatingService.java
│
├── DemoApplication.java
│
└── resources
    ├── static
    │   └── images/
    │
    ├── templates/
    │   ├── login.html
    │   ├── register.html
    │   ├── film_list.html
    │   ├── film_detail.html
    │   ├── add_film.html
    │   └── header.html
    │
    └── application.properties
```

2.4. Veritabanı Yapılandırması

Uygulama, `application.properties` dosyası üzerinden MySQL veritabanına bağlanmaktadır. Geliştirme ortamında `spring.jpa.hibernate.ddl-auto=create-drop` ayarı ile uygulama her çalıştırıldığında tablo yapıları sıfırdan oluşturulmaktadır. Bu, test senaryoları için uygun bir yaklaşımdır.

2.5. Proje Bağımlılıkları

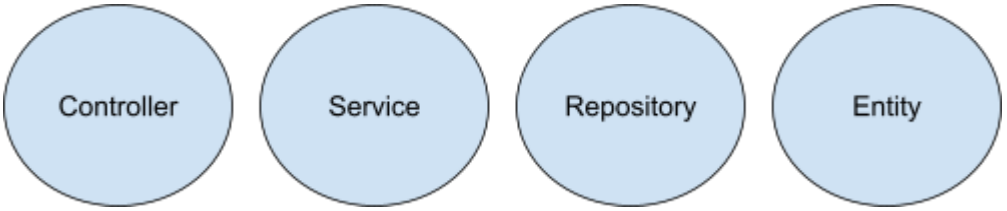
Uygulama aşağıdaki ana Maven bağımlılıklarını kullanmaktadır:

- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-security
- spring-boot-starter-thymeleaf
- mysql-connector-java
- spring-boot-devtools

3. Sistem Mimarisi ve Tasarımı

3.1. Katmanlı Mimari Yapı

Uygulama, aşağıdaki dört temel katmandan oluşmaktadır:

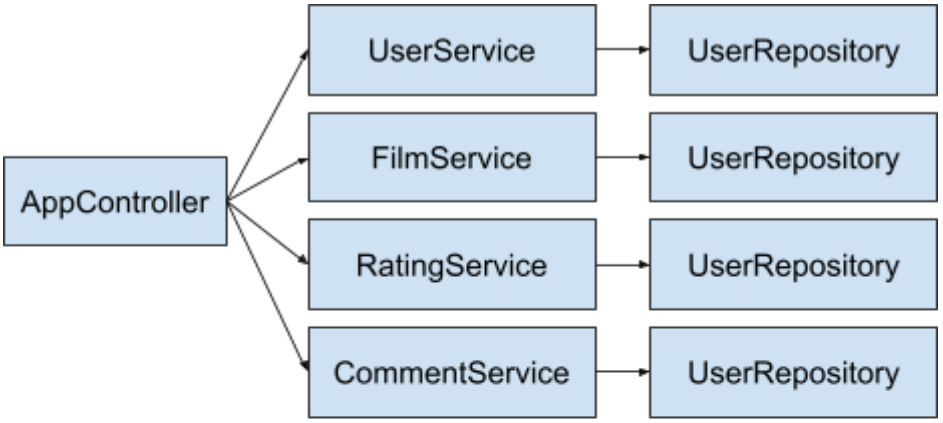


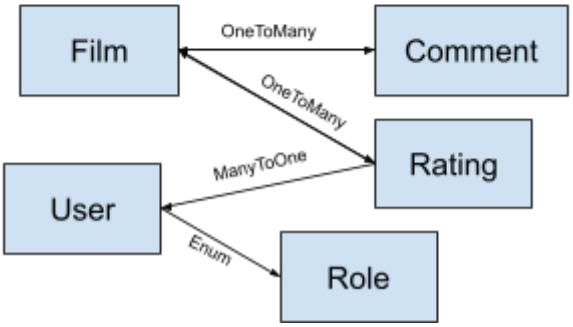
3.2. MVC Mimarisi

Spring MVC altyapısı kullanılarak geliştirilen bu uygulama, klasik Model-View-Controller yapısına sahiptir:

- **Model (Entity):** Uygulamadaki veri yapısını tanımlar (Film, User, Rating, Comment)
- **View (Thymeleaf):** Kullanıcıya sunulan HTML şablonlarıdır. Dinamik içerikler Thymeleaf ile Java'dan alınır.
- **Controller:** Uygulamanın yönlendirme katmanıdır. Kullanıcının isteklerini alır, iş mantığını çalıştırır ve görünümle birleştirir.

3.3. Sınıflar Arası İlişki Diyagramı





3.4. Spring Security Mimarisi

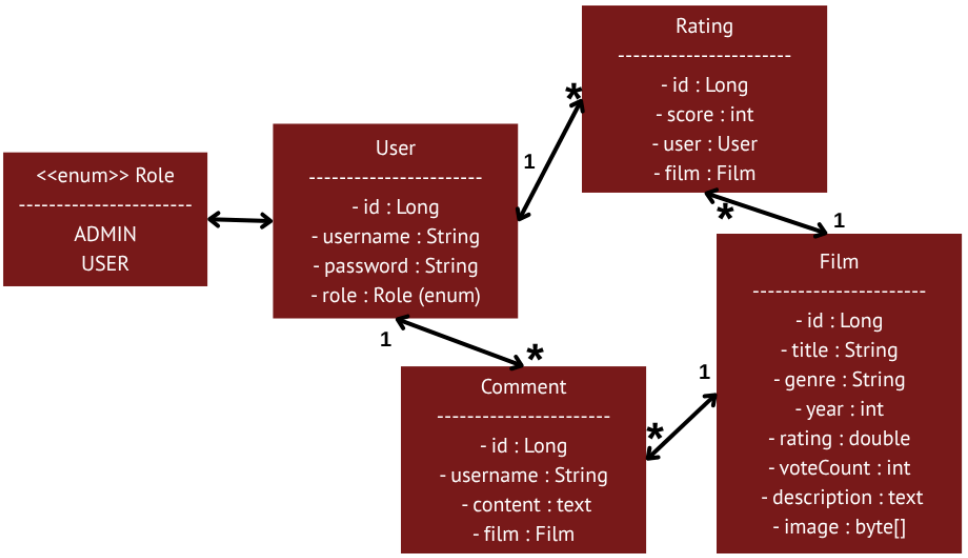
- SecurityConfig sınıfı, uygulamanın tüm güvenlik yapılandırmasını içerir.
- /admin/** gibi endpoint'ler sadece ADMIN rolüne sahip kullanıcılar tarafından erişilebilir şekilde sınırlandırılmıştır.
- Şifreler BCrypt algoritmasıyla güvenli biçimde hashlenerek veritabanına kaydedilir.
- UserDetailsService arabirimiyle özelleştirilmiş kullanıcı kimlik doğrulama işlemleri yapılır.

3.5. Görsel Saklama Yaklaşımı

- Kullanıcı tarafından yüklenen resimler MultipartFile aracılığıyla alınır.
- byte[] tipine dönüştürülerek veritabanına yazılır.
- Görüntü talep edildiğinde HTTP yanıtı içerisine Content-Type: image/jpeg header'ı eklenerek sunulur.

4. Veritabanı Tasarımı

4.1. Tablolar ve Alanlar




```sql
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL -- 'ADMIN' or 'USER'
);

CREATE TABLE films (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    genre VARCHAR(100) NOT NULL,
    year INT NOT NULL,
    rating DOUBLE NOT NULL DEFAULT 0.0,
    vote_count INT NOT NULL DEFAULT 100,
    image LONGBLOB NOT NULL,
    description TEXT
);

CREATE TABLE comments (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    film_id BIGINT NOT NULL,
    CONSTRAINT fk_comment_film FOREIGN KEY (film_id) REFERENCES films(id) ON DELETE CASCADE
);

CREATE TABLE ratings (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    film_id BIGINT NOT NULL,
    score INT NOT NULL CHECK (score BETWEEN 1 AND 5),
    CONSTRAINT fk_rating_user FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    CONSTRAINT fk_rating_film FOREIGN KEY (film_id) REFERENCES films(id) ON DELETE CASCADE,
    UNIQUE (user_id, film_id)
);
```

5. Kullanıcı Rolleri ve Yetkilendirme

5.1. Kullanıcı Rolleri

USER (Standart Kullanıcı)

- Sisteme kayıt olur ve giriş yapar
- Film listesini görüntüler
- Film detaylarını görüntüler
- Filmlere puan verir
- Filmlere yorum yapar

ADMIN (Yönetici)

- USER tüm yetkilerine sahiptir
- Yeni film ekler
- Mevcut filmi düzenler
- Filmi sistemden siler

5.2. Kimlik Doğrulama

Spring Security sayesinde kullanıcı, giriş yaptığı sırada kullanıcı adı ve şifresiyle kimlik doğrulamasına tabi tutulur. Giriş işlemi:

- /login formu üzerinden yapılır
- UserDetailsService arayüzü override edilerek veritabanından kullanıcı ve rol bilgileri yüklenir
- Şifreler BCryptPasswordEncoder ile hashlenmiş şekilde kontrol edilir

```java
@Bean
public UserDetailsService userDetailsService() {
    return username -> {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("Kullanıcı bulunamadı: " + username));

        return org.springframework.security.core.userdetails.User.builder()
                .username(user.getUsername())
                .password(user.getPassword())
                .roles(user.getRole().name()) // ENUM degiskeni olarak aldik
                .build();
    };
}
```

5.3. Yetkilendirme

Kullanıcı rolüne göre erişim yetkileri SecurityConfig sınıfında HttpSecurity ile tanımlanmıştır:

```
.authorizeHttpRequests(auth -> auth
        .requestMatchers("/register", "/login", "/css/**", "/images/**").permitAll()
        .requestMatchers("/admin/**").hasRole("ADMIN") // Admin erişimi kontrolü
        .anyRequest().authenticated()
)
```

5.4. Başarılı Giriş ve Yönlendirme

Spring Security tarafından kullanıcı giriş yaptığında, sistem otomatik olarak /films sayfasına yönlendirir

```
.formLogin(form -> form
        .loginPage("/login")
        .failureUrl( authenticationFailureUrl: "/login?error")
        .defaultSuccessUrl( defaultSuccessUrl: "/films",  alwaysUse: true)
        .permitAll()
)
```

5.5. Güvenli Oturum Yönetimi

Kullanıcı çıkış yaptığında oturum sonlandırılır ve çerez temizlenir:

```
.logout(logout -> logout
        .logoutUrl("/logout")
        .logoutSuccessUrl("/login?logout")
        .invalidateHttpSession(true)
        .deleteCookies( ...cookieNamesToClear: "JSESSIONID")
        .permitAll()
);
```

5.6. HTML'de Rol Bazlı Görünüm Kontrolü

Thymeleaf sayesinde kullanıcı rolüne göre buton, içerik gibi HTML öğeleri koşullu olarak gösterilebilir:

```
<!-- SADECE ADMIN GÖREBİLİR BURAYI -->
<div th:if="${#authorization.expression('hasRole(''ADMIN'')')}">
    <div class="d-flex justify-content-end gap-2 mb-3">
        <!-- Düzenleme  -->
        <a th:href="@{/admin/films/edit/{id}(id=${film.id})}" class="btn btn-warning">✏ Düzenle</a>

        <!-- Silme  -->
        <form th:action="@{/admin/films/delete/{id}(id=${film.id})}" method="post"
              onsubmit="return confirm('Bu filmi silmek istediğinizden emin misiniz?');">
            <input type="hidden" name="_method" value="DELETE"> <!-- DELETE -->
            <button type="submit" class="btn btn-danger">🗑 Sil</button>
        </form>
    </div>
</div>
```

6. Kodların Açıklamalı İncelemesi

6.1. Entity Sınıfları

Film Arşivi uygulamamızda kullanılan entity sınıfları, sistemdeki temel iş nesnelerini temsil eder ve veritabanındaki tablolar ile birebir ilişkilidir. Her entity sınıfı, Java sınıfları aracılığıyla JPA anotasyonları kullanılarak tanımlanmıştır.

User sınıfı, sisteme giriş yapan kullanıcıları temsil ederken, her kullanıcının bir rolü (ADMIN veya USER) bulunmaktadır.

Film sınıfı, sistemde yer alan tüm film verilerini (başlık, tür, yıl, açıklama, görsel, puan vb.) kapsar.

Comment sınıfı, kullanıcıların filmler hakkındaki görüşlerini içeren yorumları temsil eder ve her yorum belirli bir kullanıcıya ve filme aittir.

Rating sınıfı ise kullanıcıların filmlere verdikleri puanları saklar; bu sayede ortalama puan hesaplamaları yapılabilmektedir.

Sınıflar arasında doğru biçimde tanımlanmış @OneToMany, @ManyToOne gibi ilişki anotasyonları ile, veritabanında bütünlük korunmuş ve nesne yönelimli programlama ile ilişkisel veri modeli başarılı bir şekilde eşleştirilmiştir.

6.1.1. User.java

```java
@Entity
@Table(name = "users")
public class User {

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

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING) // Enum'ları string olarak sakladik
    @Column(nullable = false)
    private Role role; // Tek bir rol olacak

    //ENUM yaptık çünkü sadece 2 seçenekler var.
    public enum Role {
        ADMIN, USER
    }

    public User() {
    }

    public User(Long id, String username, String password, Role role) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.role = role;
    }
```

```java
    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public Role getRole() { return role; }

    public void setRole(Role role) { this.role = role; }
}
```

6.1.2. Film.java

```java
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String genre;
    private int year;
    @Column(nullable = false)
    private double rating;
    @Column(nullable = false)
    private int voteCount = 0;
    @OneToMany(mappedBy = "film", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Comment> comments;
    @OneToMany(mappedBy = "film", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Rating> ratings;

    public int getVoteCount() { return voteCount; }
    public void setVoteCount(int voteCount) { this.voteCount = voteCount; }
    public double getRating() { return rating; }
    public void setRating(double rating) { this.rating = rating; }

    @Lob
    private byte[] image;
    private String description;
    public Film() {
    }
    public Film(Long id, String title, String genre, int year, byte[] image, String description) {
        this.id = id;
        this.title = title;
        this.genre = genre;
        this.year = year;
        this.image = image;
        this.description = description;
    }
```

```java

    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }

    public void setTitle(String title) { this.title = title; }

    public String getGenre() { return genre; }

    public void setGenre(String genre) { this.genre = genre; }

    public int getYear() { return year; }

    public void setYear(int year) { this.year = year; }

    public byte[] getImage() { return image; }

    public void setImage(byte[] image) { this.image = image; }

    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }

    public byte[] loadImageAsBytes(String filename) {
        try {
            Path imagePath = Path.of( first: "src/main/resources/static/images/" + filename);
            return Files.readAllBytes(imagePath);
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

6.1.2. Comment.java

```java
package com.example.filmarchive.entity;

import jakarta.persistence.*;

@Entity
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;
    private String content;

    @ManyToOne
    @JoinColumn(name = "film_id", nullable = false)
    private Film film;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getContent() { return content; }

    public void setContent(String content) { this.content = content; }

    public Film getFilm() { return film; }

    public void setFilm(Film film) { this.film = film; }
}
```

6.1.2. Rating.java

```java
@Entity
@Table(name = "ratings")
public class Rating {

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

    @ManyToOne(optional = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "film_id", nullable = false)
    private Film film;

    private int score; // 1-5 arası puan

    public Rating() {
    }

    public Rating(User user, Film film, int score) {
        this.user = user;
        this.film = film;
        this.score = score;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public User getUser() { return user; }
    public void setUser(User user) { this.user = user; }
    public Film getFilm() { return film; }
    public void setFilm(Film film) { this.film = film; }
    public int getScore() { return score; }
    public void setScore(int score) { this.score = score; }
```

6.2. Repository

Film Arşivi uygulamasında veri erişim işlemleri, Spring Data JPA kullanılarak geliştirilmiş repository arayüzleri aracılığıyla gerçekleştirilmiştir. Uygulamada UserRepository, FilmRepository, CommentRepository ve RatingRepository olmak üzere dört temel repository sınıfı bulunmaktadır.

UserRepository, kullanıcıların kullanıcı adı üzerinden sorgulanmasını sağlayarak kimlik doğrulama sürecinin temelini oluşturur.

FilmRepository, filmlerin başlık, tür veya yapım yılı gibi alanlara göre filtrelenmesini sağlar ve kullanıcıya film arama deneyimi sunar.

CommentRepository, belirli bir filme ait tüm yorumların listelenmesini kolaylaştırır.

RatingRepository ise kullanıcıların daha önce oy verip vermediğini kontrol eder ve aynı zamanda film puan ortalamasını hesaplamada kullanılacak tüm oyları çekmeyi sağlar.

Bu arayüzler, Spring'in sağladığı hazır JpaRepository yapısını genişleterek yazılımcının manuel SQL sorgusu yazmasına gerek bırakmaz ve veri işlemlerini sade, güvenli ve etkili bir biçimde gerçekleştirir.

6.2.1. UserRepository.java

```java
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

6.2.1. FilmRepository.java

```java
public interface FilmRepository extends JpaRepository<Film, Long> {
    // arama metotlari

    List<Film> findByTitleContainingIgnoreCase(String title);

    List<Film> findByGenreContainingIgnoreCase(String genre);

    List<Film> findByYear(int year);

}
```

6.2.1. CommentRepository.java

```
public interface CommentRepository extends JpaRepository<Comment, Long> {
    List<Comment> findByFilmId(Long filmId);
}
```

6.2.1. RatingRepository.java

```
public interface RatingRepository extends JpaRepository<Rating, Long> {
    Optional<Rating> findByUserAndFilm(User user, Film film);
    List<Rating> findByFilm(Film film);
}
```

6.3. Service

Service katmanı, Film Arşivi uygulamasında Controller ile Repository katmanları arasında bir köprü görevi gören orta katmandır. Bu katman, veri erişimi ile ilgili karmaşık işlemlerin işlenmesini sağlar, veri tutarlılığını korur ve Controller sınıflarını doğrudan veritabanı bağımlılıklarından soyutlar. Uygulamada UserService, FilmService, CommentService ve RatingService olmak üzere dört ana service sınıfı bulunmaktadır.

UserService, kullanıcıların sisteme kaydedilmesi, şifrelerin güvenli biçimde hashlenmesi ve kullanıcı bilgilerine erişim gibi işlemleri yürütür.

FilmService, film ekleme, güncelleme, silme, filtreleme ve sıralama gibi film yönetimi ile ilgili tüm işlevleri üstlenir.

CommentService, kullanıcı yorumlarının veritabanına eklenmesi ve listelenmesini sağlar.

RatingService ise film puanlarının hesaplanması, güncellenmesi ve ortalama puanın yönetilmesini kontrol eder.

6.3.1. UserService.java

```java
@Service
public class UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
    public boolean isUsernameTaken(String username) {
        return userRepository.findByUsername(username).isPresent();
    }
    public User save(User user) {
        user.setPassword(passwordEncoder.encode(user.getPassword())); // sifreleme
        return userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }
}
```

6.3.2. FilmService.java

```java
@Service
public class FilmService {

    private final FilmRepository filmRepository;

    public FilmService(FilmRepository filmRepository) { this.filmRepository = filmRepository; }

    // Tüm filmleri getir
    public List<Film> findAll() { return filmRepository.findAll(); }

    // ID'ye göre film bul
    public Optional<Film> findById(Long id) { return filmRepository.findById(id); }

    // Film ekle/güncelle
    public Film save(Film film) { return filmRepository.save(film); }
    public List<Film> findByGenreContainingIgnoreCase(String genre) {
        return filmRepository.findByGenreContainingIgnoreCase(genre);
    }
    public List<String> findAllGenres() {
        return filmRepository.findAll().stream() Stream<Film>
                .map(Film::getGenre) Stream<String>
                .distinct()
                .toList();
    }

    public List<Film> findByTitleContaining(String title) {
        return filmRepository.findByTitleContainingIgnoreCase(title);
    }

    public void deleteById(Long id) { filmRepository.deleteById(id); // id ile sildik }
}
```

6.3.3. CommentService.java

```java
@Service
public class CommentService {

    private final CommentRepository commentRepository;

    public CommentService(CommentRepository commentRepository) {
        this.commentRepository = commentRepository;
    }

    // idye gore yorumlari getirdik
    public List<Comment> findByFilmId(Long filmId) {
        return commentRepository.findByFilmId(filmId);
    }

    // Yorum ekle
    public Comment save(Comment comment) {
        return commentRepository.save(comment);
    }
}
```

6.3.4. RatingService.java

```java
@Service
public class RatingService {

    private final RatingRepository ratingRepository;

    public RatingService(RatingRepository ratingRepository) {
        this.ratingRepository = ratingRepository;
    }

    public Rating save(Rating rating) {
        return ratingRepository.save(rating);
    }

    public Optional<Rating> findByUserAndFilm(User user, Film film) {
        return ratingRepository.findByUserAndFilm(user, film);
    }

    public List<Rating> findByFilm(Film film) {
        return ratingRepository.findByFilm(film);
    }

    public double calculateAverageRating(Film film) {
        List<Rating> ratings = ratingRepository.findByFilm(film);
        if (ratings.isEmpty()) {
            return 0.0;
        }
        double total = ratings.stream().mapToInt(Rating::getScore).sum();
        return total / ratings.size();
    }
}
```

6.4. Controller

Film Arşivi uygulamasında tüm HTTP endpoint'leri tek bir sınıf olan AppController içerisinde yönetilmiştir.

6.4.1. AppController.java

Genel Sorumluluklar:

- Giriş ve kayıt ekranlarını sunar
- Film listesini ve detaylarını kullanıcıya gösterir
- Arama, filtreleme, sıralama işlemlerini yönetir
- Yorum ve oy ekleme işlemlerini gerçekleştirir
- Admin işlemleri (film ekle, düzenle, sil) için endpoint'ler içerir
- Dinamik görsel sunumunu gerçekleştirir (`/films/{id}/image`)

Metotlar ve Görevleri:

Metot	HTTP	Açıklama
showLoginPage()	GET /login	Giriş sayfasını gösterir
showRegistrationForm()	GET /register	Kayıt sayfasını gösterir
registerUser()	POST /register	Yeni kullanıcıyı kayıt eder
listFilms(...)	GET /films	Tüm filmleri listeler, arama/sıralama/filtreleme desteği içerir
getFilmDetail(id)	GET /films/{id}	Film detay sayfası, yorumlar ve puan ortalaması ile birlikte
addComment(id)	POST /films/{id}/comments	Filme yorum ekler
rateFilm(id)	POST /films/{id}/rate	Filme oy verir ve puanı günceller
getFilmImage(id)	GET /films/{id}/image	Filmin görselini byte[] olarak döner
showAdminFilmForm()	GET /admin/films/add	Admin panelinden yeni film ekleme sayfası
addFilmAdmin()	POST /admin/films	Admin tarafından film eklenir/güncellenir
showEditFilmForm(id)	GET /admin/films/edit/{id}	Film düzenleme sayfasını açar
deleteFilm(id)	POST /admin/films/delete/{id}	Filmi veritabanından siler

Rol Bazlı Erişim:

Thymeleaf şablonlarında hasRole('ADMIN') kullanılarak butonlar ve işlemler role göre kontrol edilir.

/admin/** yolları yalnızca ADMIN kullanıcılar tarafından erişilebilir (SecurityConfig içinde ayarlanmıştır).

BMT-308
WEB PROGRAMLAMA
SPRING PROJE RAPORU

Kodları:

```
@Controller
public class AppController {

    private final UserService userService;

    private final FilmService filmService;
    private final CommentService commentService;
    private final RatingService ratingService;

    public AppController(UserService userService, FilmService filmService,
                         CommentService commentService, RatingService
ratingService) {
        this.userService = userService;
        this.filmService = filmService;
        this.commentService = commentService;
        this.ratingService = ratingService;
    }
    // LOGIN SAYFASI
    @GetMapping("/login")
    public String showLoginPage() {
        return "login";
    }
    // REGISTER SAYFASI
    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }
```

```
    @PostMapping("/register")
    public String registerUser(@ModelAttribute User user,
                               @RequestParam String confirmPassword,
                               Model model) {
        if (userService.isUsernameTaken(user.getUsername())) {
            model.addAttribute("error", "Bu kullanıcı adı zaten alınmış!");
            return "register";
        }

        if (!user.getPassword().equals(confirmPassword)) {
            model.addAttribute("passwordError", "Şifreler uyuşmuyor!"); // sifre
uyusmazsa hata gonderdil
            return "register";
        }
        user.setRole(User.Role.USER);
        userService.save(user);
        return "redirect:/login?success";
    }

    @GetMapping("/films")
    public String listFilms(@RequestParam(value = "genre", required = false)
String genre,
                            @RequestParam(value = "sort", required = false)
String sort,
                            @RequestParam(value = "query", required = false)
String query,
                            Model model) {
        List<Film> films;

        if (query != null && !query.isEmpty()) {
            films = filmService.findByTitleContaining(query);
        } else if (genre != null && !genre.isEmpty()) {
            films = filmService.findByGenreContainingIgnoreCase(genre);
```

```
        } else {
            films = filmService.findAll();
        }

        if (sort != null) {
            switch (sort) {
                case "yearAsc" ->
films.sort(Comparator.comparing(Film::getYear));
                case "yearDesc" ->
films.sort(Comparator.comparing(Film::getYear).reversed());
                case "ratingAsc" ->
films.sort(Comparator.comparing(Film::getRating));
                case "ratingDesc" ->
films.sort(Comparator.comparing(Film::getRating).reversed());
            }
        }

        List<String> genres = filmService.findAllGenres();

        model.addAttribute("films", films);
        model.addAttribute("genres", genres);
        model.addAttribute("selectedGenre", genre);

        return "film_list";
    }
    //  FİLM DETAYI + YORUMLAR + PUAN ORTALAMASI
    @GetMapping("/films/{id}")
    public String getFilmDetail(@PathVariable Long id, Model model) {
        Film film = filmService.findById(id).orElseThrow(() -> new
RuntimeException("Film bulunamadı"));
        List<Comment> comments = commentService.findByFilmId(id);
        double averageRating = ratingService.calculateAverageRating(film);

        model.addAttribute("film", film);
```

```
        model.addAttribute("comments", comments);
        model.addAttribute("averageRating", averageRating);

        return "film_detail";
    }
    // FİLME YORUM EKLE
    @PostMapping("/films/{id}/comments")
    public String addComment(@PathVariable Long id,
                             @RequestParam String content,
                             @AuthenticationPrincipal UserDetails principal) {
        Film film = filmService.findById(id).orElseThrow(() -> new
RuntimeException("Film bulunamadı"));
        User user = userService.findByUsername(principal.getUsername())
                .orElseThrow(() -> new RuntimeException("Kullanıcı
bulunamadı"));

        Comment comment = new Comment();
        comment.setUsername(user.getUsername());
        comment.setContent(content);
        comment.setFilm(film);
        commentService.save(comment);
        return "redirect:/films/" + id;
    }

    // FİLME PUAN VER
    @PostMapping("/films/{id}/rate")
    public String rateFilm(@PathVariable Long id,
                           @RequestParam int score,
                           @AuthenticationPrincipal UserDetails principal) {
        Film film = filmService.findById(id).orElseThrow(() -> new
RuntimeException("Film bulunamadı"));
        User user = userService.findByUsername(principal.getUsername())
```

```
                .orElseThrow(() -> new RuntimeException("Kullanıcı
bulunamadı"));

        Optional<Rating> existingRating = ratingService.findByUserAndFilm(user,
film);


        Rating rating = existingRating.orElseGet(() -> new Rating(user, film,
score));

        if (existingRating.isPresent()) {

            double totalRating = film.getRating() * film.getVoteCount() -
existingRating.get().getScore() + score;

            film.setRating(totalRating / film.getVoteCount());

        } else {

            double totalRating = film.getRating() * film.getVoteCount() + score;

            film.setVoteCount(film.getVoteCount() + 1);

            film.setRating(totalRating / film.getVoteCount());

        }

        rating.setScore(score);

        filmService.save(film);

        ratingService.save(rating);

        return "redirect:/films/" + id;

    }

    // GÖRSEL GETİR

    @GetMapping("/films/{id}/image")

    public ResponseEntity<byte[]> getFilmImage(@PathVariable Long id) {

        Film film = filmService.findById(id).orElseThrow(() -> new
RuntimeException("Film bulunamadı"));

        byte[] image = film.getImage();

        HttpHeaders headers = new HttpHeaders();

        headers.setContentType(MediaType.IMAGE_JPEG);

        return new ResponseEntity<>(image, headers, HttpStatus.OK);

    }

    // ADMIN: Film Ekleme Formu

    @GetMapping("/admin/films/add")

    public String showAdminFilmForm(Model model) {
```

```
        model.addAttribute("film", new Film());
        return "add_film";
    }
    @PostMapping("/admin/films")
    public String addFilmAdmin(@ModelAttribute Film film,
@RequestParam("imageFile") MultipartFile imageFile) throws IOException {
        if (film.getId() != null) { // guncellenen film mi
            Film existingFilm = filmService.findById(film.getId())
                    .orElseThrow(() -> new RuntimeException("Film bulunamadı"));

            if (!imageFile.isEmpty()) {
                film.setImage(imageFile.getBytes()); // yeni resim
            } else {
                film.setImage(existingFilm.getImage()); // eski resim
            }
        }
        else { // yeni bir film ekleniyorsa, resim kontrolünü ekle
            if (!imageFile.isEmpty()) {
                film.setImage(imageFile.getBytes()); // yeni resim eklendiyse
kaydet
            } else {
                throw new IllegalArgumentException("Film eklerken resim
zorunludur!");
            }
        }

        filmService.save(film);
        return "redirect:/films?updateSuccess"; // film başarıyla kaydedildiyse
yonlendir
    }

    @GetMapping("/admin/films/edit/{id}")
    public String showEditFilmForm(@PathVariable Long id, Model model) {
```

```
        Film film = filmService.findById(id).orElseThrow(() -> new
RuntimeException("Film bulunamadı"));

        model.addAttribute("film", film);

        return "add_film";

    }


    @PostMapping("/admin/films/delete/{id}")

    public String deleteFilm(@PathVariable Long id) {

        try {

            Film film = filmService.findById(id)

                    .orElseThrow(() -> new RuntimeException("Film bulunamadı"));


            filmService.deleteById(id);

            return "redirect:/films?deleteSuccess"; // silme başariliysa
yönlendirme

        } catch (RuntimeException e) {

            return "redirect:/films?error=FilmSilinemedi";

        }

    }

}
```

6.5. Yapılandırma (Config) Sınıfları

6.5.1. SecurityConfig.java

Kullanıcıların hangi sayfalara erişebileceğini belirleyen, kimlik doğrulama ve yetkilendirme işlemlerini yöneten sınıftır.

```
@Configuration

@EnableWebSecurity

public class SecurityConfig {


    private final UserRepository userRepository;


    // Constructor injection
```

```
    public SecurityConfig(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/register", "/login", "/css/**",
"/images/**").permitAll()
                        .requestMatchers("/admin/**").hasRole("ADMIN") // Admin
erişimi kontrolü
                        .anyRequest().authenticated()
                )
                .formLogin(form -> form
                        .loginPage("/login")
                        .failureUrl("/login?error")
                        .defaultSuccessUrl("/films", true)
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutUrl("/logout")
                        .logoutSuccessUrl("/login?logout")
                        .invalidateHttpSession(true)
                        .deleteCookies("JSESSIONID")
                        .permitAll()
                );
        return http.build();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

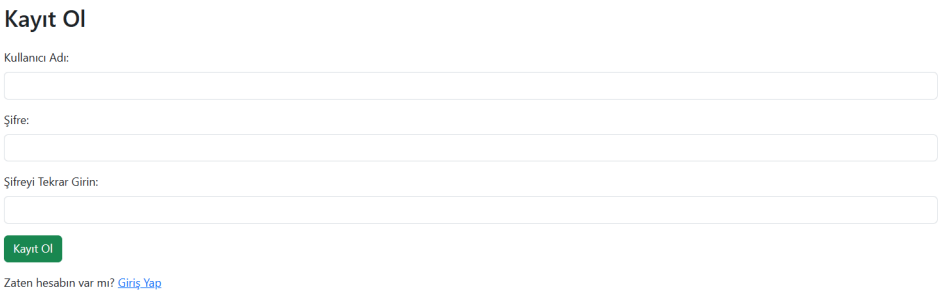
```
    @Bean
    public UserDetailsService userDetailsService() {
        return username -> {
            User user = userRepository.findByUsername(username)
                    .orElseThrow(() -> new UsernameNotFoundException("Kullanıcı
bulunamadı: " + username));


            return org.springframework.security.core.userdetails.User.builder()
                    .username(user.getUsername())
                    .password(user.getPassword())
                    .roles(user.getRole().name()) // ENUM degiskeni olarak aldik
                    .build();
        };
    }
    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

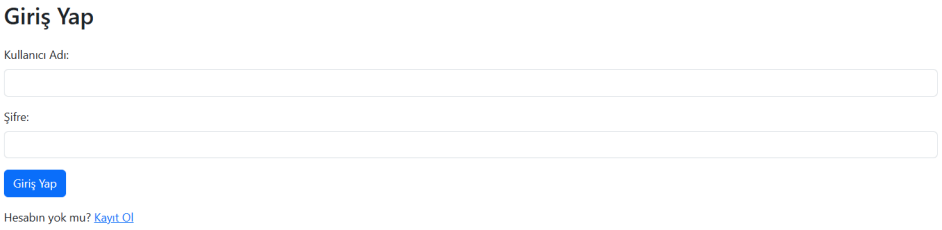
Fonksiyon Adı	Görevi / Açıklama
SecurityConfig(UserRepository)	UserRepository nesnesini sınıfa enjekte ederek kullanıcı verilerine erişimi sağlar.
filterChain(HttpSecurity http)	Uygulamadaki güvenlik kurallarını tanımlar: giriş/çıkış, yetki kontrolü, URL erişimi.
passwordEncoder()	Kullanıcı şifrelerini güvenli biçimde hash'lemek için BCryptPasswordEncoder döner.
userDetailsService()	Kullanıcının username bilgisine göre UserDetails nesnesi oluşturur (giriş kontrolü).

7. Gerçekleştirilen Fonksiyonlar ve Ekranlar

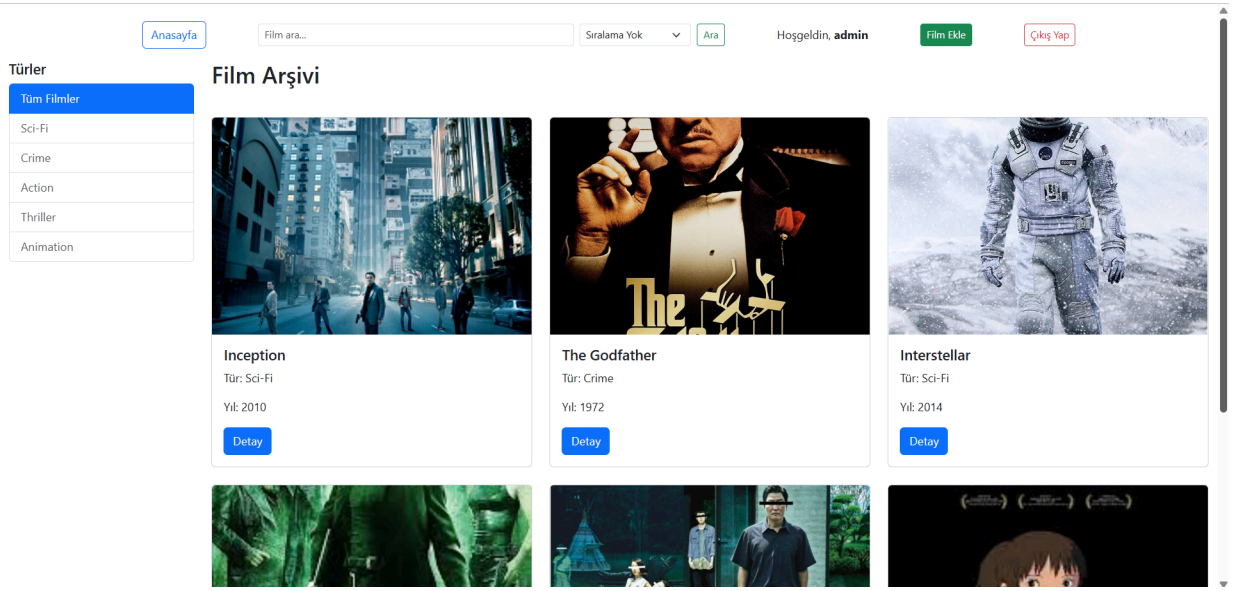
7.1. Kayıt Olma (Register)



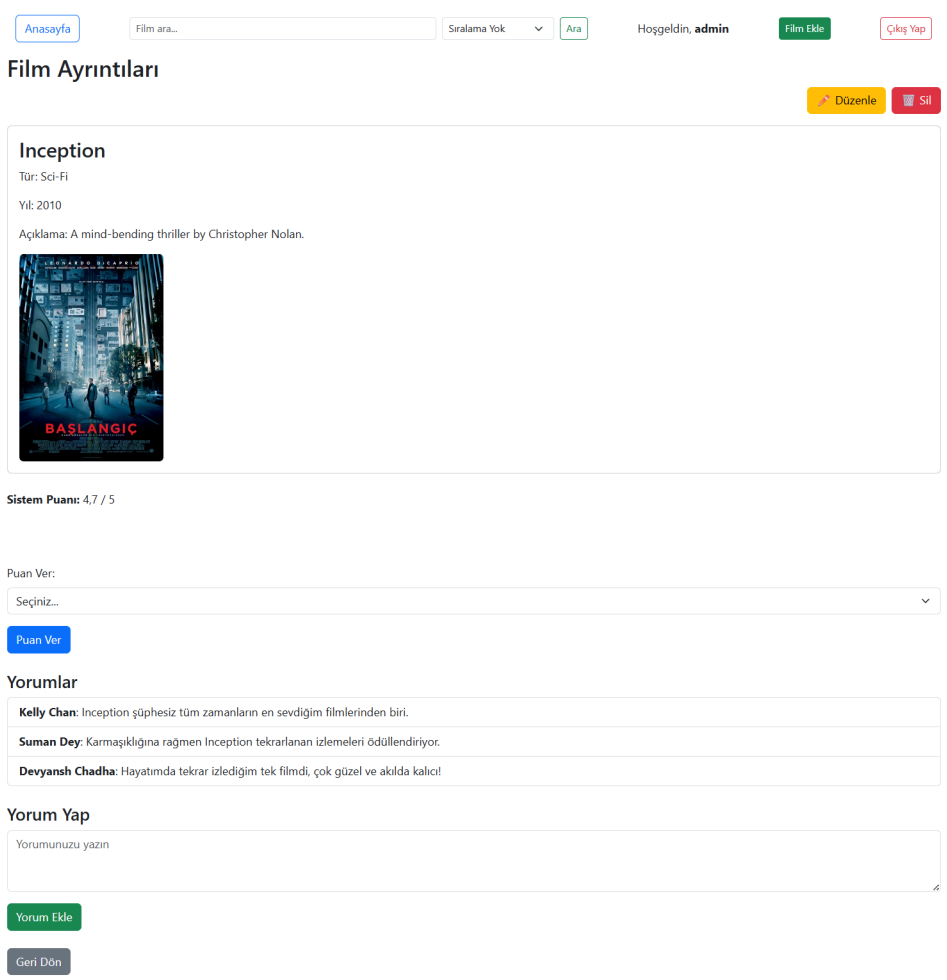
7.2. Giriş Yapma (Login)



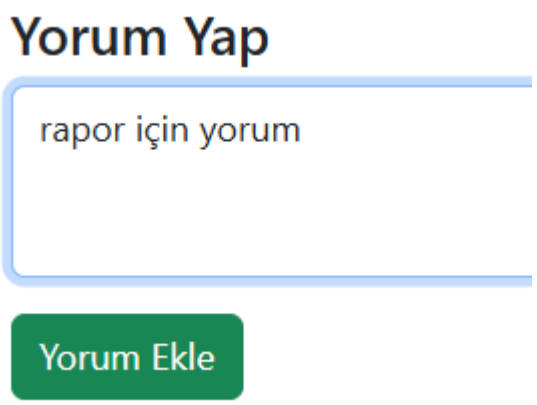
7.3. Film Listeleme (Ana Sayfa)

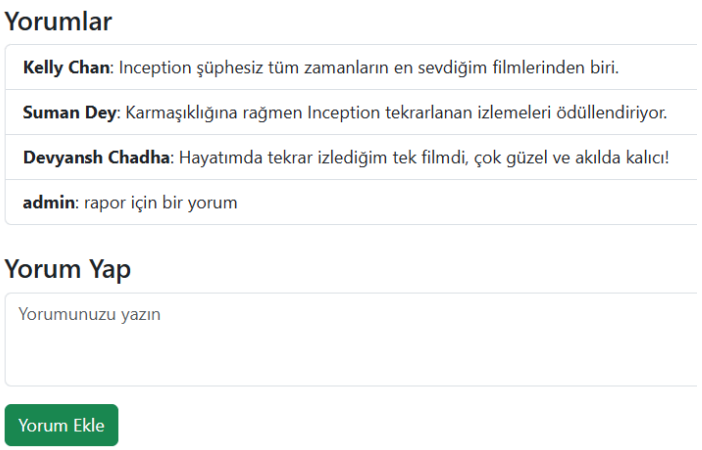


7.4. Film Detayı

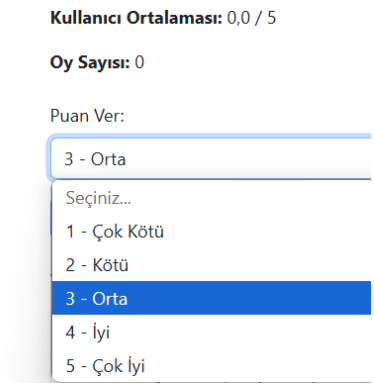


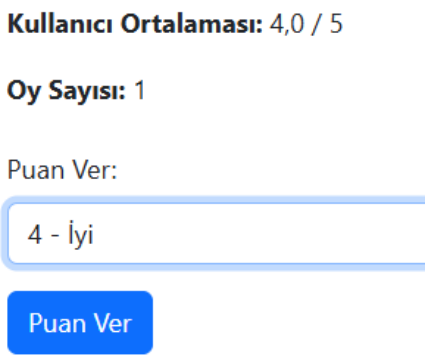
7.5. Yorum Ekleme



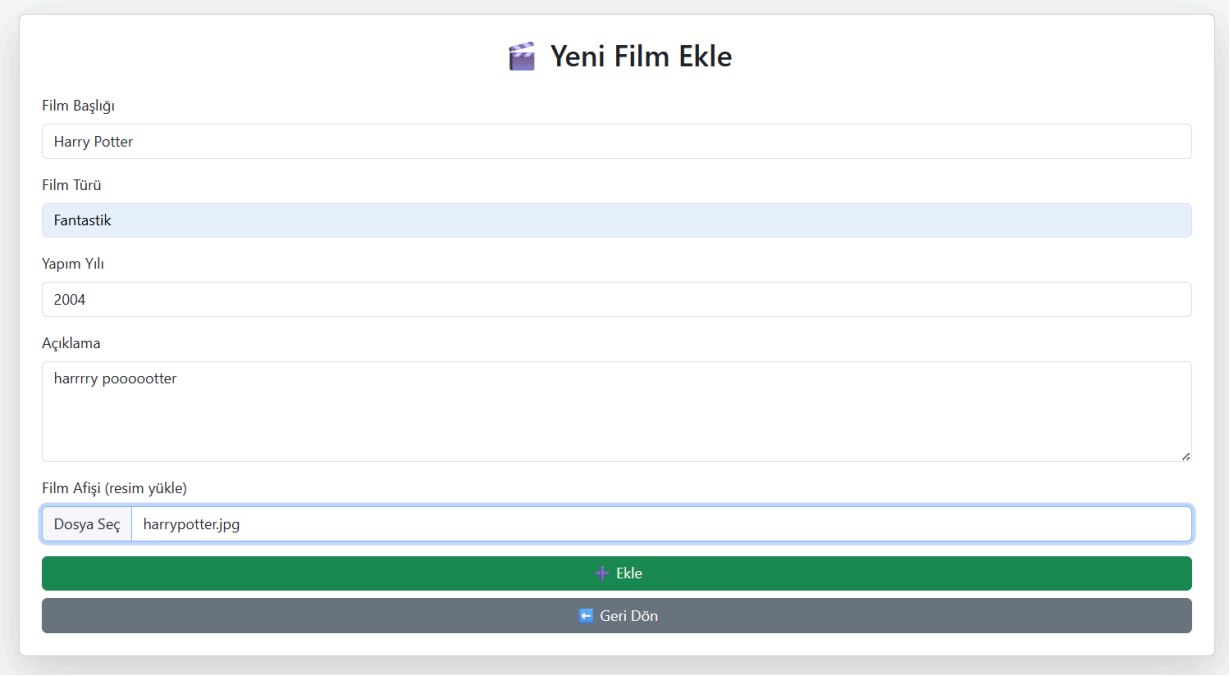


7.6. Puan Verme





7.7. Film Ekleme (Admin Paneli)



BMT-308
WEB PROGRAMLAMA
SPRING PROJE RAPORU

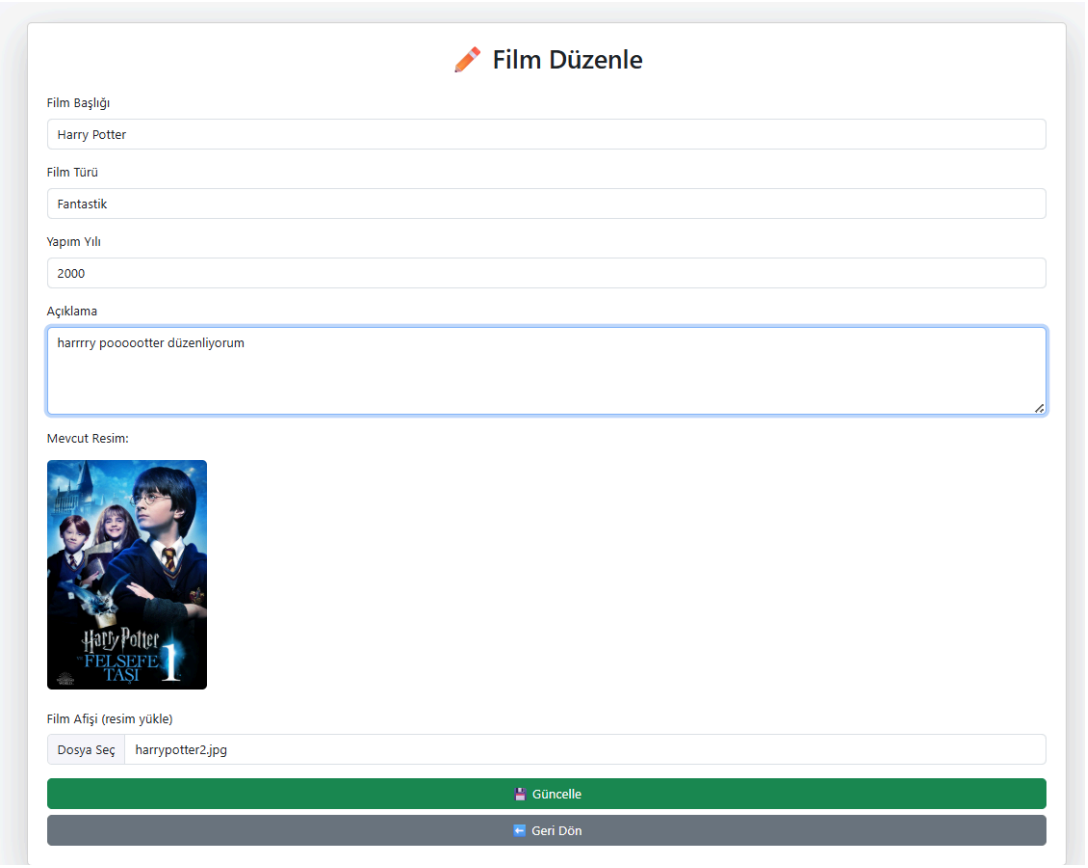
Harry Potter

Tür: Fantastik

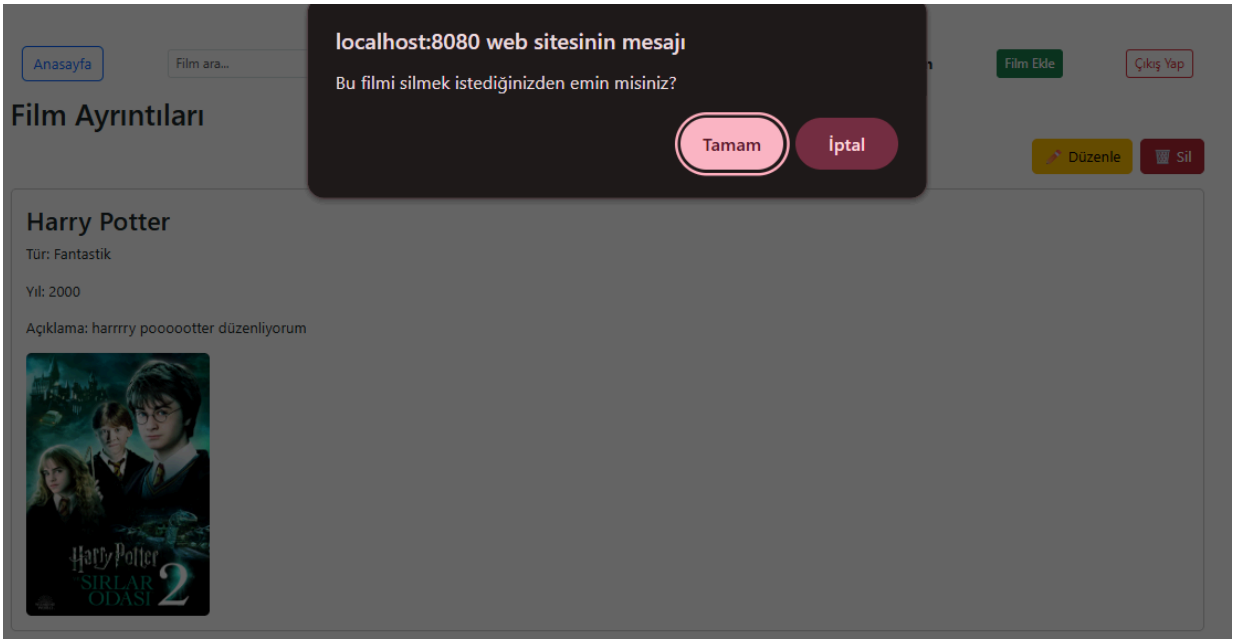
Yıl: 2004

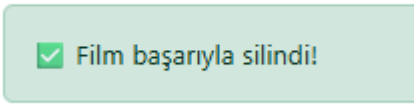
Açıklama: harrrry poooootter

7.8. Film Güncelleme (Admin Paneli)

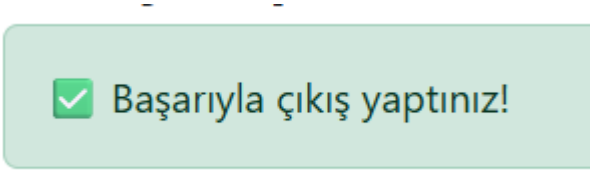


7.9. Film Silme





7.10. Oturumdan Çıkış (Logout)



8. Sonuç ve Değerlendirme

Film Arşivi web uygulaması, kullanıcıların filmleri görüntüleyip değerlendirebildiği, yorum yapabildiği ve yöneticilerin film ekleyip düzenleyebildiği tam işlevli bir sistem olarak başarıyla geliştirilmiştir. Spring Boot, Thymeleaf, Spring Security ve JPA gibi modern teknolojiler kullanılarak, güvenli ve sürdürülebilir bir yapı oluşturulmuştur. Geliştirme sürecinde karşılaşılan teknik sorunlar analiz edilerek uygun çözümler üretilmiş, sistem istenen tüm işlevleri yerine getirecek şekilde test edilmiştir. Proje, hem teknik bilgi hem de uygulama becerisi açısından öğretici ve kapsamlı bir çalışma olmuştur. thanks to beyonceeeeeeee