

Projektdokumentation

Github Repo's

Backend:

- [FloCodin/Modul223Backend](#)

Frontend:

- [FloCodin/m233_frontend](#)

Movie Watchlist App – Projektdokumentation

Projektübersicht

Dies ist eine Fullstack Webapplikation zur Verwaltung einer persönlichen **Movie-Watchlist** mit integrierter **Bewertungsfunktion**, **TMDB-Integration**, **JWT Authentifizierung** und **Rollensteuerung**.

- **Frontend:** React + TailwindCSS + Vite
- **Backend:** Spring Boot + JPA + JWT + MySQL
- **API:** The Movie Database (TMDB)

Features

- ☒ Nutzerregistrierung & Login
- ☒ JWT-basierte Authentifizierung
- ☒ Rollen: `USER` , `ADMIN`
- ☒ Watchlist pro User
- ☒ Bewertungen mit Score (1–5) + Kommentar
- ☒ Automatisches Speichern von TMDB-Filmen,
- ☒ Filmübersicht mit Bild, Beschreibung, Jahr
- ☒ Integration von Toast-Messages für Feedback

Technische Architektur

```
React (Frontend)
|
|— Axios → /api/*
|
|— REST API
|   |— Spring Boot
|       |— Controllers
|       |— Services
|       |— Repositories
|       |— Entities: Movie, User, Rating, WatchlistEntry
|       |— Security: JWT + Role-based Access
|       |— Database: MySQL
```

Wichtige Backend-Klassen

RatingService.java

```
public Rating createRating(Long userId, RatingDTO ratingDto) {
    Movie movie = movieRepository.findById(ratingDto.movieId)
        .orElseGet(() -> {
            Movie fetched = tmdbService.fetchMovieFromTmdb(ratingDto.movieId);
            return movieRepository.save(fetched);
        });

    User user = userRepository.findById(userId)
        .orElseThrow(() -> new RuntimeException("User not found"));

    Rating rating = new Rating(user, movie, ratingDto.score,
ratingDto.comment);
    return ratingRepository.save(rating);
}
```

TmdbService.java

```
public Movie fetchMovieFromTmdb(Long movieId) {
    RestTemplate restTemplate = new RestTemplate();
    String apiKey = "DEIN_TMDB_API_KEY";
    String url = "https://api.themoviedb.org/3/movie/" + movieId + "?api_key="
+ apiKey + "&language=de-DE";
    ResponseEntity<Map> response = restTemplate.getForEntity(url, Map.class);
}
```

```
        if (!response.getStatusCode().is2xxSuccessful()) throw new
RuntimeException("Movie fetch failed");

        Map<String, Object> data = response.getBody();
        Movie movie = new Movie();
        movie.setId(movieId);
        movie.setTitle((String) data.get("title"));
        movie.setDescription((String) data.get("overview"));
        movie.setDuration((Integer) data.get("runtime"));
        movie.setYear(Integer.parseInt(((String)
data.get("release_date")).substring(0, 4)));
        movie.setDirector("Unbekannt");

        return movie;
    }
}
```

⚙️ Beispiel-Testfälle

✅ Bewertung hinzufügen (Integration)

- ♦ **Gegeben:** User ist eingeloggt
- ♦ **Wenn:** User klickt auf „Rezension schreiben“
- ♦ **Und:** Gibt Score 4 + Kommentar „War cool“ ein
- ♦ **Dann:** POST /api/ratings ⇒ Status 200, neue Bewertung ist sichtbar

✅ Movie automatisch speichern (Backend)

- ♦ **Gegeben:** Movie mit TMDB-ID 123 existiert nicht in DB
- ♦ **Wenn:** User bewertet den Film
- ♦ **Dann:** `TmdbService.fetchMovieFromTmdb()` wird aufgerufen → `movieRepository.save()` speichert Film




❌ Fehlerhafte Bewertung

- ♦ **Gegeben:** Bewertung wird ohne gültigen Token gesendet
- ♦ **Dann:** Backend gibt HTTP 403 zurück

- Postman oder Thunder Client zum Testen von Endpoints
 - MySQL Workbench zur Datenbankanalyse
 - IntelliJ IDEA mit Spring Boot Plugin
 - Vite + React + Toastify für UI Feedback
-



Bekannte Probleme

- ☒  `ObjectOptimisticLockingFailureException` bei mehrfacher Speicherung (gelöst durch DTO & Update-Logik)
 - ☒  `NullPointerException` wenn `movieRepository` im `TmdbService` nicht `@Autowired` ist
 - ☐  `UNSAFE_componentWillReceiveProps` Warnung in `react-rating` (nicht kritisch)
-



Erweiterungen

- ☐ Bewertungen editierbar machen
 - ☐ Likes/Dislikes zu Bewertungen
 - ☐ Adminbereich mit Userübersicht
 - ☐ Review-History pro User
-



Security Hinweis

- Bei **403 Forbidden** ohne Token → Token per `Authorization: Bearer <token>` senden
 - Verwende `@PreAuthorize("hasRole('USER') or hasRole('ADMIN')")` an den REST-Endpunkten
-



Projektstruktur (Frontend)

```
src/  
├── components/  
│   ├── RatingForm.jsx  
│   └── MovieCard.jsx
```

```
└─ pages/
  └─ Watchlist.jsx
  └─ MyRatings.jsx
  └─ TmdbSearch.jsx
```

Letzte Änderung: 2025-05-26 14:29



Datenmodell (PlantUML)

```
@startuml
entity User {
    *id : Long
    --
    username : String
    password : String
    email : String
}

entity Role {
    *id : Long
    --
    name : String
}


entity Movie {
    *id : Long
    --
    title : String
    description : String
    year : Integer
    duration : Integer
    director : String
}


entity Rating {
    *userId : Long
    *movieId : Long
    --
    score : Integer
    comment : String
}
```


```
}
```


```
entity WatchlistEntry {  
    *userId : Long  
    *movieId : Long  
}
```


```
User ||--o{ Rating : rates  
User ||--o{ WatchlistEntry : has  
Movie ||--o{ Rating : rated  
Movie ||--o{ WatchlistEntry : appears_in  
User ||--o{ Role : assigned  
@enduml
```


watchlist_entries 	
movieId	string pk
status	string
userId	string pk

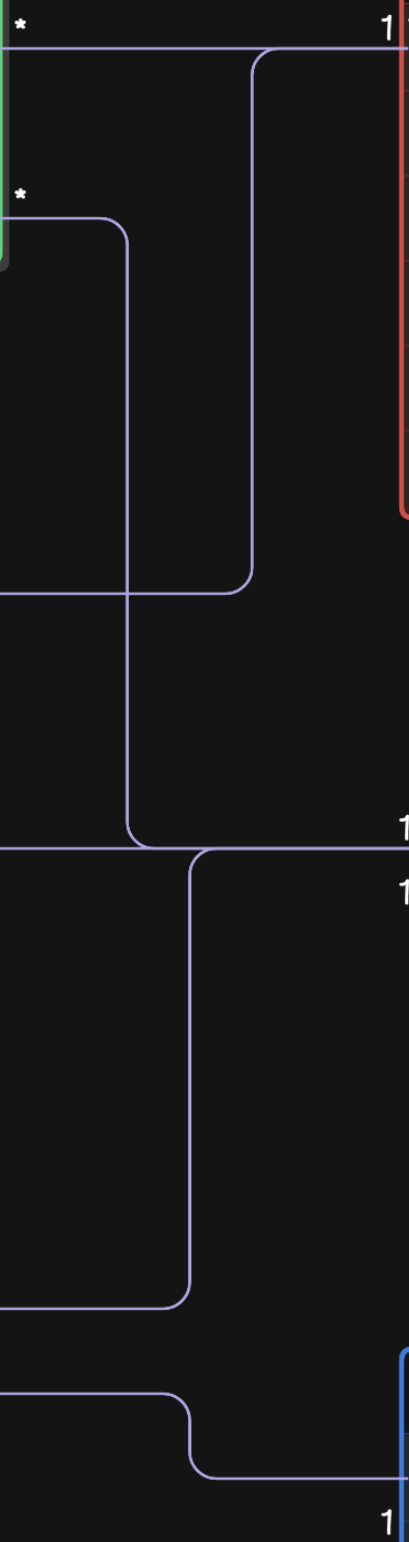
movies 	
id	string pk
title	string
description	string
year	int
duration	int
director	string

ratings 	
movieId	string pk
score	int
comment	string
userId	string pk

users 	
id	string pk
username	string
password	string
email	string

user_roles 	
userId	string pk
roleId	string pk

roles 	
id	string pk
name	string
description	string



🔧 Installation & Setup

1. Backend starten

```
./mvnw spring-boot:run
```

Voraussetzungen:

- Java 22
- MySQL mit Datenbank `demo`

2. Frontend starten

```
npm install  
npm run dev
```

3. Umgebungsvariablen (optional)

- `.env` für TMDB-API-Key:

```
VITE_TMDB_API_KEY=dein_tmdb_key
```

Beispiel-Testfälle (manuell)

- ✓ Benutzer kann sich registrieren & einloggen
- ✓ Bewertung für neuen TMDB-Film wird gespeichert und Film automatisch hinzugefügt
- ✓ Bewertung wird mit bestehender überschrieben bei erneutem Speichern
- ✓ Watchlist funktioniert unabhängig von Bewertungen

✓ Use Cases & Tests (automatisierbar)

Backend Unit Tests (JUnit 5)

✓ `RatingServiceTest.java`

- `createRating_shouldSaveRating_whenMovieExists()`

- `createRating_shouldFetchAndSaveMovie_whenNotInDb()`
- `deleteRating_shouldRemoveRatingFromDatabase()`
- `getRatingsByUser_shouldReturnUserRatings()`

✓ `TmdbServiceTest.java`

- `fetchMovieFromTmdb_shouldReturnValidMovieObject()`
- `fetchMovieFromTmdb_shouldThrowExceptionForInvalidId()`
- `getPopularMovies_shouldReturnList()`

✓ `AuthTokenFilterTest.java`

- `shouldExtractValidJwt()`
- `shouldRejectInvalidJwt()`

Frontend Unit Tests (Jest + React Testing Library)

✓ `RatingForm.test.jsx`

- `should render input fields correctly`
- `should submit rating and reset form on success`
- `should show error toast on API failure`

✓ `MovieCard.test.jsx`

- `should render movie details`
- `should call handlers for watchlist and rating`

✓ `AuthService.test.js`

- `should login and return user with token`
- `should logout and clear session storage`

Integration Tests (Postman)

- POST `/api/auth/login` → Login mit gültigem Benutzer
 - GET `/api/tmdb/popular` → Liste der Filme von TMDB
 - POST `/api/ratings` → Bewertung eines Films (mit Token)
 - GET `/api/ratings/me` → Bewertungen des aktuellen Nutzers
-



Zusätzliche Ideen für Validierungstests

- Token abgelaufen → Zugriff verweigert (401)
- Bewertung mit Score > 5 oder < 1 → Fehler (422)
- Film-ID nicht existent → `TmdbService` gibt Fehler zurück
- Rollenwechsel (`USER` → `ADMIN`) ohne Neuladen → keine Adminrechte sichtbar