



























Projektdokumentation

Florian Rogenmoser

Inhaltsverzeichnis-

- [Github Repo's](#)
-  [Projektübersicht](#)
-  [Features](#)
-  [Technische Architektur](#)
-  [Wichtige Backend-Klassen](#)
 - [`RatingService.java`](#)
 - [`TmdbService.java`](#)
-  [Beispiel-Testfälle](#)
 -  [Bewertung hinzufügen \(Integration\)](#)
 -  [Movie automatisch speichern \(Backend\)](#)
 -  [Fehlerhafte Bewertung](#)
-  [Tools](#)
-  [Bekannte Probleme](#)
-  [Erweiterungen](#)
-  [Security Hinweis](#)
-  [Projektstruktur \(Frontend\)](#)
-  [Datenmodell \(PlantUML\)](#)
-  [Installation & Setup](#)
-  [Beispiel-Testfälle \(manuell\)](#)
-  [Use Cases & Tests \(automatisierbar\)](#)
 -  [Backend Unit Tests \(JUnit 5\)](#)
 -  [`RatingServiceTest.java`](#)
 -  [`TmdbServiceTest.java`](#)
 -  [`AuthTokenFilterTest.java`](#)
 -  [Frontend Unit Tests \(Jest + React Testing Library\)](#)
 -  [`RatingForm.test.jsx`](#)
 -  [`MovieCard.test.jsx`](#)
 -  [`AuthService.test.js`](#)
 -  [Integration Tests \(Postman \)](#)

- 📌 Zusätzliche Ideen für Validierungstests
 - 📅 Arbeitsjournal – Projekt Movie Watchlist
 - 📄 Vorgehen
 - 📅 Arbeitsjournal (chronologisch)
 - 📄 Zusätzliche User Stories
 - 🟡 1. Als registrierter Benutzer möchte ich meine eigene Watchlist einsehen können
 - 🔴 2. Als Benutzer möchte ich Bewertungen abgeben und einsehen können
 - 🟡 3. Als Benutzer möchte ich einen Film zur Watchlist hinzufügen
 - 🟢 4. Als Benutzer möchte ich nach Filmen suchen und Details sehen (funktioniert!)
 - 🔴 5. Als Benutzer möchte ich mein Benutzerprofil (Username, Rolle) einsehen
 - 📊 Zusätzliche Soll-Ist-Analyse (bezogen auf User Stories)
-

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

Tools

- `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

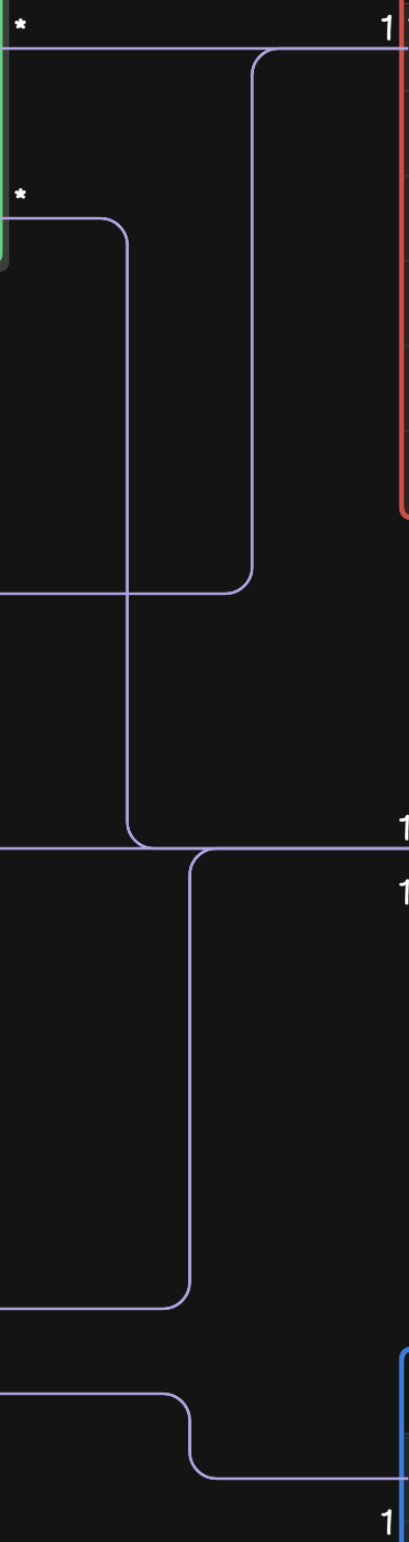
movies	
id	string pk
title	string
description	string
year	int
duration	int
director	string

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



Arbeitsjournal – Projekt Movie Watchlist



Vorgehen

Wir haben uns für ein iteratives Vorgehen entschieden, orientiert am **agilen Modell**. Die Arbeit wurde in folgende Phasen aufgeteilt:

1. **Analyse und Planung**
2. **Backend-Setup inkl. Authentifizierung und Datenmodellierung**
3. **Frontend-Aufbau mit Design & Struktur (React + Tailwind)**
4. **Integration der TMDb-API & Bewertungssystem**
5. **Testing, Feinschliff und Dokumentation**



Arbeitsjournal (chronologisch)

Datum	Aufgabe	Beschreibung	Tools / Technologien
03.05.2025	Setup Projektstruktur	Frontend- und Backend-Repository erstellt, Docker Container	GitHub, IntelliJ, Vite
03.05.2025	Authentifizierung Backend	JWT Login, User Entity & Role-System implementiert	Spring Boot, JPA
03.05.2025	Datenbankanbindung	Lokale MySQL mit User-, Movie- und Rating-Tabelle	MySQL, Hibernate
10.05.2025	Frontend Grundlayout	Routing, Pages, Tailwind CSS Styling	React, Tailwind
17.05.2025	Watchlist & Bewertung	Entitäten + POST/GET/DELETE API-	REST API, DTO

Datum	Aufgabe	Beschreibung	Tools / Technologien
		Endpunkte erstellt	
-24.05.2025	TMDB API Integration	Dynamisches Abrufen von Movie-Details	TMDB v4, RestTemplate
24.05.2025	Bewertungssystem im Frontend	React-Komponente mit Sternsystem	react-rating, Toastify
31.05.2025	Tests Backend	CRUD-Tests für RatingController & Services	JUnit, MockMvc
31.05.2025	Tests Frontend	Manuelle Tests: Login, Bewertung, Watchlist	Browser, Postman
26-31.05.2025	Dokumentation	Technische Dokumentation & ReadMe begonnen	Markdown, PlantUML
31.05.2025	Soll-Ist-Vergleich & Feinschliff	Übersicht, Analyse offener Punkte	Tabellen, Rückblick



Zusätzliche User Stories

1. Als registrierter Benutzer möchte ich meine eigene Watchlist einsehen können

Akzeptanzkriterien:

- Nach dem Login sehe ich unter „Meine Watchlist“ alle hinzugefügten Filme
- Die Watchlist zeigt Titel, Bild und Beschreibung an
- Filme lassen sich aus der Watchlist entfernen

Technisch: GET /api/watchlist , DELETE /api/watchlist/{id}

2. Als Benutzer möchte ich Bewertungen abgeben und einsehen können

Akzeptanzkriterien:

- Es gibt eine Oberfläche zur Eingabe von Score (1–5) und Kommentar

- Bereits vorhandene Bewertungen werden angezeigt
- Bewertung wird gespeichert oder überschrieben

Technisch: `POST /api/ratings` , `GET /api/ratings/me`

● 3. Als Benutzer möchte ich einen Film zur Watchlist hinzufügen

Akzeptanzkriterien:

- In der Detailansicht eines Films gibt es einen „Zur Watchlist hinzufügen“-Button
- Der Button reagiert auf Klick und gibt Rückmeldung (Toast)
- Der Film erscheint danach auf der Watchlist-Seite

Technisch: `POST /api/watchlist`

● 4. Als Benutzer möchte ich nach Filmen suchen und Details sehen (funktioniert!)

Akzeptanzkriterien:

- Eingabe eines Suchbegriffs zeigt TMDB-Ergebnisse
- Klick auf Film zeigt Poster, Beschreibung, Jahr
- Integration funktioniert unabhängig von Watchlist

Technisch: `GET /api/tmdb/search?q=xyz`

● 5. Als Benutzer möchte ich mein Benutzerprofil (Username, Rolle) einsehen

Akzeptanzkriterien:

- Nach dem Login wird mein Username angezeigt
- Die Rolle (USER / ADMIN) beeinflusst UI-Komponenten
- Ein Logout-Button funktioniert

Technisch: `GET /api/me` (optional), `localStorage` , JWT

Zusätzliche Soll-Ist-Analyse (bezogen auf User Stories)

User Story Nr.	Beschreibung	Soll-Zustand	Ist-Zustand	Abweichung / Bemerkung
1	Watchlist anzeigen	Filme aus Watchlist sichtbar und entfernbar	● 403 Fehler bei GET /api/watchlist	🔥 Auth funktioniert nicht im Frontend
2	Bewertung schreiben	Bewertung mit Kommentar möglich	● UI vorhanden, aber POST scheitert	🔥 API wird nicht korrekt aufgerufen
3	Watchlist hinzufügen	Button ruft POST auf, zeigt Toast & aktualisiert Watchlist	● Button sichtbar, aber hat keine Wirkung	🦎 API-Call oder AuthHeader fehlt
4	Film-Suche & Anzeige	TMDB-Suche zeigt Treffer + Details	✅ Funktioniert wie erwartet	—
5	Profil/Logout	Username/Rolle sichtbar, Logout löscht Token	● Keine Benutzeranzeige, Logout ungetestet	⚠️ Nur LocalStorage ohne echte Anzeige