

Programmieren 2

Exercise 2

Ziele der Übung

- API in Java konsumieren - einfache HTTP Requests/Responses interpretieren
- Libraries im Projekt integrieren und verwenden
- Unterschiedliche Requests an die MovieAPI senden, Responses im JSON Format in Java Objekte parsen und in der Applikation mithilfe von Java Streams verarbeiten

Aufgabenstellung

Im Zuge der Exercise 2 sollen Daten einer Remote-API in der FHMDb App geladen und verarbeitet werden. Dafür werden Requests mit unterschiedlichen Parametern an die MovieAPI gesendet und deren Responses in der FHMDb ausgegeben. Die Requests können mithilfe einer Library (z.B. okhttp) umgesetzt werden. Die MovieAPI stellt die Daten im JSON Format zur Verfügung. Die Responses sollen mithilfe von entweder Gson oder Jackson geparsed und auf passende Java Klassen gemapped werden.

MovieAPI analysieren und FHMDb App erweitern

Analysiert die Endpoints, die möglichen Parameter und die Datenstruktur der Responses der MovieAPI (<https://prog2.fh-campuswien.ac.at/swagger-ui/index.html>). Nutzt Tools wie Postman oder Curl um unterschiedliche Requests/Responses zu untersuchen.

MovieAPI Klasse

Erstellt eine MovieAPI Klasse, welche die Logik zum Senden und Empfangen von Requests/Responses, sowie die Erstellung der geeigneten URL (bspw. Endpoint und notwendige Parameter) implementiert. In weiterer Folge soll eure MovieAPI Klasse von eurem Programm (z.B. vom Controller), mit unterschiedlichen Parametern und Endpoints aufgerufen werden können.

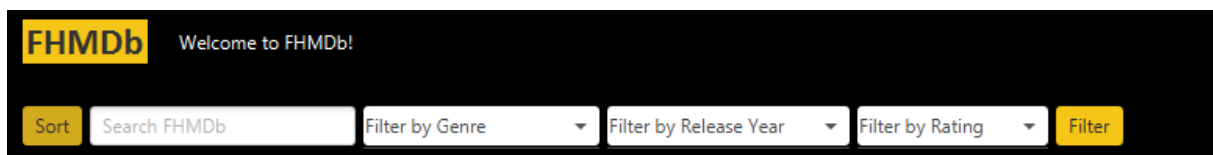
Beispiele:

1. Beim Starten der FHMDb App sollen alle Filme der MovieAPI, ohne weitere Filterkriterien, geladen werden: `https://prog2.fh-campuswien.ac.at/movies`
2. Wenn User*innen einen Wert im Freitextfeld und beim Genre selektieren, sollen dementsprechende Parameter in der URL gesetzt werden: `https://prog2.fh-campuswien.ac.at/movies?query=userinput&genre=ACTION`

Hint: die API erwartet einen gesetzten *User-Agent*-Header im Request. Dieser muss gesetzt werden, bevor der Request abgesendet wird. Beispielsweise „User-Agent http.agent“ (<https://square.github.io/okhttp/4.x/okhttp/okhttp3/-request/-builder/add-header/>)

UI adaptieren

Ergänzt euer UI soweit, dass User*innen die gesendeten Parameter individuell einstellen können (query, genre, releaseYear, ratingFrom). Z.B.:



The screenshot shows the top part of the FHMDb application. It has a black background with a yellow 'FHMDb' logo and the text 'Welcome to FHMDb!'. Below this is a search bar with the placeholder 'Search FHMDb'. To the right of the search bar are four filter buttons: 'Filter by Genre', 'Filter by Release Year', 'Filter by Rating', and a yellow 'Filter' button. Each filter button has a dropdown arrow.

Java Streams

Weiters sind folgende Methoden unter Verwendung von Streams im Controller zu implementieren:

- **String getMostPopularActor(List<Movie> movies):** gibt jene Person zurück, die am öftesten im mainCast der übergebenen Filme vorkommt.
- **int getLongestMovieTitle(List<Movie> movies):** filtert auf den längsten Filmtitel der übergebenen Filme und gibt die Anzahl der Buchstaben des Titels zurück
- **long countMoviesFrom(List<Movie> movies, String director):** gibt die Anzahl der Filme eines bestimmten Regisseurs zurück.
- **List<Movie> getMoviesBetweenYears(List<Movie> movies, int startYear, int endYear):** gibt jene Filme zurück, die zwischen zwei gegebenen Jahren veröffentlicht wurden.

Die Funktionen sind ausschließlich mit Streams zu implementieren (keine Schleifen!)

Bewertungskriterien

- Die Movie Klasse ist um notwendige Attribute (API Response) erweitert
- Filme werden über die API konsumiert und im UI angezeigt
- Die MovieAPI Klasse stellt Methoden bereit, die es ermöglichen unterschiedliche Parameter (Genre, Freitext, etc.) an die API zu senden
- Die Methoden der MovieAPI können über das UI mit individuellen Parametern aufgerufen werden
- Die Responses der API werden in Java Objekte (Movie) geparsed
- Vorgegebene Methoden sind mit Java Streams umgesetzt

Empfohlene Vorgehensweise

- Erweitert eure Applikation von Exercise 1 oder klonst die Solution (<https://github.com/leonardo1710/fhmdb-exercise-1>)
- Untersucht die Endpunkte der API (was sind mögliche Responses? Welche Parameter können übergeben werden?)
- Erweitert eure *Movie* Klasse um die notwendigen Attribute (siehe Response)
- Erstellt und implementiert die *MovieAPI* Klasse (Hint: hier können die Libraries okhttp und gson/jackson verwendet werden)
- Adaptiert die *Controller* Klasse indem ihr anstelle der statischen Liste aus Exercise 1 die Funktion(en) der *MovieAPI* aufruft
- Implementiert die vorgegebenen Methoden mithilfe von Java Streams
- Es muss zusammen mit GIT gearbeitet werden

Abgabe

- Link zum Repository auf Moodle abgeben (letzter Commit vor Abgabeende wird bewertet)
- Das Repository muss public sein!

Links und Ressourcen

- Gson: <https://github.com/google/gson>
- Jackson: <https://github.com/FasterXML/jackson>
- Okhttp: <https://square.github.io/okhttp/>
- Postman: <https://www.postman.com/>