# Auftrag Labor Erweiterung (Teil 2)

|  |  |  |  |
| --- | --- | --- | --- |
| Richtzeit: 120‘ |  | Sozialform: Einzelarbeit |  |

## Einleitung

Im Labor haben wir gemeinsam eine Schnittstelle für die Entität User betrachtet. Im Auftrag Teil 1 hast du das Gelernte selber umgesetzt. Das Grundkonzept und Zusammenspiel von Controller, Service, Repository und Model solltest du nun verstanden haben. Ausserdem weisst du wie du Dependencies (build.gradle) sowie Application.properties richtig setzt. In diesem Auftrag gehen wir nun mit diesem erlerntden Wissen weiter und erstellen die restlichen Methoden bzw. Schnittstellen für unsere Entitäten.

Orientiere dich immer am Beispiel des Users, um die restlichen Schnittstellen zu erstellen.

Damit du weisst, was auf dich zukommt, hier ein Screenshot, wie deine Applikation am Ende aussehen sollte, wenn du mit dem Auftrag fertig bist:

Ein Bild, das Text, Screenshot, Karte Menü, Schrift enthält.

Automatisch generierte Beschreibung

Damit du diverse Anhaltspunkte während des Auftrages hast, hier ein Beispiel wie unser bisher umgesetztes Backend aussieht:

UserController.java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http-methode | url | Returned | Logik | Exception |
| GET @Pathvariable | Localhost/users/id | User | findById(id) | EntityNotFoundException  NOT\_FOUND |

UserService.java

|  |  |  |  |
| --- | --- | --- | --- |
| Methode | Erhält Variablentyp | Weiterleitung | returned |
| findById | Integer | Repository.findById | User oder EntityNotFoundException |

Diese Schnittstelle wurde bereits umgesetzt. Beachte das du bei deinen Repositories im Moment noch keinen logischen Code implementieren musst, um den Auftrag zu beenden.

## Erstellung der Entitäten

Bevor wir an die Erstellung der Controller, Services und Repos gehen, erstellen wir zunächst die Models für unsere Movie und Genre Tabelle. Analysiere folgendes ERD:

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Für die Erstellung des User-Models hatten wir keinen Spezialfall. Jedoch müssen wir nun beachten, dass Movie und Genre in einer ManyToOne Beziehung zueinanderstehen. Grundsätzlich müssen wir nicht viel an unseren neuen Models anpassen, da uns Spring Boot mit Annotations enorm dabei unterstützt. Wir können folgenden Code verwenden, um die gesamte Verbindung automatisch vom Backend behandeln zu lassen:

Movie.java

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Movie.java

Ein Bild, das Text, Schrift, Screenshot, Grafiken enthält.

Automatisch generierte Beschreibung

Mit diesen Annotations teilen wir unserem Backend mehrere Sachen mit.

1. Die beiden Models bzw. Entitäten stehen in der Datenbank in einer Beziehung
2. Bei der Ausgabe eines Movies erhalten wir das dazugehörige genre und bei der Ausgabe eines Genres erhalten wir ein Set an Movies
3. JsonBackReference kümmert sich darum, das bei der Ausgabe eines Genres nicht das Problem auftaucht, dass ein Genre Movies enthält, Movies wiederum Genres und diese wieder Movies, etc. Wir unterbinden somit eine Endlosschlaufe bei der Ausgabe.

Mit diesem Wissen erstelle nun die Models zu Movie und Genre. Lasse die Getter und Setter automatisch von deiner IDE generieren um Fehler zu vermeiden.

done

## Repositories

Sobald du deine Entitäten erstellt hast, kannst du die dazugehörigen Repositories erstellen. Da diese keinen logischen Code bis jetzt beinhalten, kannst du dich am Beispiel des UserRepository orientieren.

Done, bei den extends CRUD wurde jeweisl <Movie,Integer> und <Genre,Integer> eingetragen

## Controller & Services

So jetzt geht es um die effektive Logik unserer Schnittstellen. Genau gleich wie im Beispiel im vorherigen Kapitel, musst du nun selbstständig die Logik für die restlichen Methoden erstellen. Beachte, dass sämtliche Methoden für alle Controller & Services gelten (Movie, Genre und auch User).

Beachte, dass beim Beispiel findById wir auf die url …/users/id navigieren und mit @Pathvariable den inhalt von /id lesen. Wenn wir aber nicht mit der URL-Daten übermitteln, sondern mit dem RequestBody (bspw. Ganze Objekte), benötigen wir @RequestBody.

Ausserdem haben wir bisher beim @Getmapping auch noch direkt mitgegeben, dass wir auf den Pfad «id» hören mit: (path = "{id}") . Dies ist bei @PostMapping und @PutMapping anders. Da wir dort einen RequestBody erwarten, müssen wir zusätzlich angeben, dass wir ein json akzeptieren bei einem Request. Anstelle von:



Verwenden wir also für diese Funktionen:





**Controller Implementationen:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http-methode | url | Returned | Logik | Exception |
| GET | /users  /genres  /movies | Iterable<User>  Iterable<Genre> Iterable<Movie> | findAll() | EntityNotFoundException  NOT\_FOUND |
| POST @RequestBody | /users  /genres  /movies | Void | signUp(user)  insert(genre)  insert(movie) | RuntimeException  CONFLICT |
| PUT RequestBody | /users  /genres  /movies | Void | update(user)  update(genre)  update(movie) | RuntimeException  CONFLICT |
| DELETE @Pathvariable | /users/id  /genres/id  /movies/id | Void | deleteById(id) | RuntimeException  NOT\_FOUND |

*Beachte: die Methode signUp sollte nur im UserController signUp heissen. In den anderen Controller wird sie «insert» genannt.*

**Service Implementationen:**

|  |  |  |  |
| --- | --- | --- | --- |
| Methode | Erhält Variablentyp | Weiterleitung | returned |
| findAll | - | Repository.findAll | Iterable<Model> |
| signUp / insert | User / Movie / Genre | Repository.save | Void |
| Update | User / Movie / Genre | Repository.save | Void |
| deleteById | Integer | Repository.deleteById | void |

## Kontrolle

Wenn du nun alle deine Controller, Services, Repositories und Models erstellt hast, solltest du im Stande sein die Applikation zu starten.Überprüfe mit Hilfe von Postman nun ob deine Schnittstellen für Movie, Genre und User funktionieren.

Du solltest im Stande sein, spezifische Movies, Genres und User zu erhalten und auch die gesamte Liste der jeweiligen Entitäten.

Wenn das bei dir funktioniert hat, bist du fertig mit diesem Auftrag. Bravo! Du hast nun das Grundgerüst deines Backends beendet.