Spotitube

Opleverdocument

Joep Hoffland

Door: Joep Hoffland

Studentnummer: 601552

Klas: ITA-OOSE-A-f 2018

Docent: Michel Portier

Datum: 2 april 2019

Versie: 1.0

2019

## Inhoudsopgave

[Spotitube 0](#_Toc5139313)

[Inhoudsopgave 1](#_Toc5139314)

[1. Inleiding 2](#_Toc5139315)

[2. Package diagram 3](#_Toc5139316)

[2.1 Toelichting 3](#_Toc5139317)

[2.1.1 Context & Dependency injection 3](#_Toc5139318)

[2.1.2 Layer en Remote Facade pattern 3](#_Toc5139319)

[2.1.3 Open Closed principle (OCP) - SOLID 3](#_Toc5139320)

[3. Deployment diagram 4](#_Toc5139321)

[4. Ontwerpkeuzes 5](#_Toc5139322)

[4.1 DataSource i.p.v. DriverManager 5](#_Toc5139323)

[4.2 Validatie 5](#_Toc5139324)

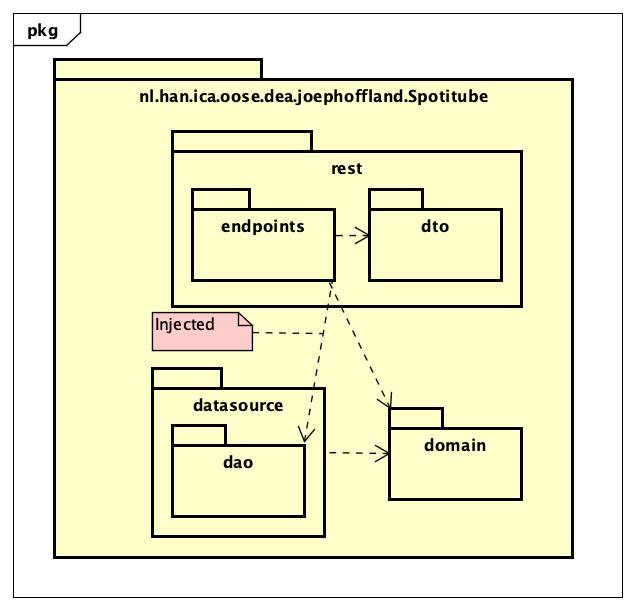
## 1. Inleiding

Spotitube is een gezamenlijke app van Spotify en YouTube waarmee een klant een overzicht kan krijgen van afspeellijsten met daarin audio- en videostreams.

Voor het vak DEA binnen het OOSE-semester aan de HAN-ICA heb ik de opdracht gekregen om de back-end te ontwikkelen. Deze applicatie moet gebruik maken van JAX-RS, CDI (Context & Dependency injection), de JDBC API en gedeployed kunnen worden op Apache TomEE Plus. De client (front-end) is aangeleverd door DEA en moet RESTful kunnen communiceren met de back-end volgens de REST API specificatie.

De applicatie heb ik gerealiseerd op basis van de casusbeschrijving, waarbij ik gebruik heb gemaakt van een Data access layer, het Domain layer pattern en het Remote Facade pattern. Daarnaast wordt 100% van de code via Unittests getest en is dit document een toelichting op de applicatie.

## 2. Package diagram



Figuur 1 Package diagram Spotitube

### 2.1 Toelichting

#### 2.1.1 Context & Dependency injection

Door gebruik te maken van Dependency injection worden de dependencies door de container "injected" tijdens runtime, waardoor het onder andere makkelijker wordt om te wisselen naar een nieuwe implementatie. CDI wordt toegepast op alle DAO objecten die gebruikt worden in de rest.endpoints package, door gebruik te maken van de JAX @Inject annotatie.

#### 2.1.2 Layer en Remote Facade pattern

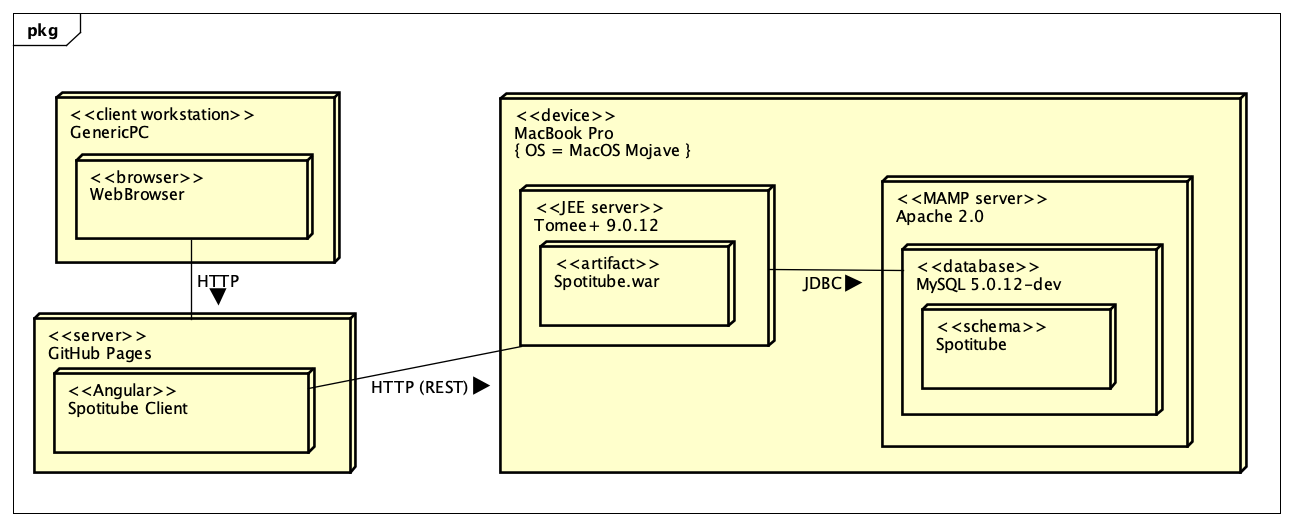
Layer pattern: Het endpoint vraagt aan de DAO (Data Access Object) uit de Datasource layer informatie uit de database op. Vervolgens vult de DAO een Domain-object uit de Domain Layer en retourneert deze aan het endpoint.

Remote Facade pattern: Het endpoint zet het Domain-object om naar een DTO (Data Transfer Object) die als HTTP-response wordt teruggegeven aan de client (Remote Facade pattern).

#### 2.1.3 Open Closed principle (OCP) - SOLID

Elke DAO implementeert een interface. Deze interface wordt ook in de rest.endpoints gebruikt om het type object aan te geven.

## 3. Deployment diagram



Figuur 2 Deployment diagram Spotitube

## 4. Ontwerpkeuzes

### 4.1 DataSource i.p.v. DriverManager

Ik heb ervoor gekozen om gebruik te maken van DataSource in plaats van DriverManager om de databaseconnectie op te bouwen, omdat TomEE de databaseconnecties maakt en sluit en de databasgegevens ook opgeslagen worden in de serverconfiguratie.

### 4.2 Validatie

Ik heb ervoor gekozen om de controle op de data dat door de client wordt ingestuurd te doen in de endpoints. Het is mogelijk om aan de `SQLException` te herkennen dat de gegevens die je op probeert te slaan niet geldig zijn. Een voorbeeld van een ongeldig gegeven is een te lange naam van een afspeellijst. Door direct in de endpoints te controleren of de gegevens geldig zijn, kan er een betere foutmelding als HTTP-response terug worden gegeven. Het nadeel hiervan is wel dat je het verstand van de databasesstructuur eigenlijk ook bij de database wil houden.