Spotitube Opleverdocument



Leander ten Hoedt

634442

ITA-OOSE-A-f

Michel Portier

26-3-2021

v1

# Inhoud

[**Inhoud**](#_60c459vxnqlk) **1**

[**Inleiding**](#_scqgxik957lb) **2**

[**Package diagram**](#_gojhq1w5bb5o) **3**

[Requirements](#_bkf70zpeoljw) 3

[Ontwerpprincipes](#_puxivi101knf) 4

[Alternatieve oplossing](#_hdrypai0p5em) 4

[Huidige oplossing](#_e15crmlmdgbs) 4

[**Deployment diagram**](#_eyq99dhyq17o) **5**

[Requirements](#_kjbwo0ktc0ht) 5

[Ontwerpprincipes](#_mz62iy6wkvat) 5

[Alternatieve oplossing](#_8spnmfx7v686) 5

[Huidige oplossing](#_ambl4oxxoxy4) 5

[**Ontwerp-/Craftmanship -keuzes**](#_9123ekl9x16a) **6**

# Inleiding

Dit is het opleverdocument van de Spotitube applicatie. In dit document staan 2 diagrammen die de applicatie schematisch uitleggen, daarnaast staat er per diagram extra uitleg. Verder worden er een aantal ontwerpkeuzes uitgelegd.

# Package diagram

# 

## Requirements

Ik gebruik alle layers die gebruikt moeten worden dus:

* Data access layer (dao)
* Domain layer (domain)
* Service layer (service)
* Remote Facade (dto)

## Ontwerpprincipes

Allereerst eerst er een Data access layer geïmplementeerd. Deze bestaat uit de dao package die boven in het diagram staat. Daana komt de domain layer, de domain objects worden gemaakt in de dao’s en worden doorgestuurd naar de service layer. De rest package in de service layer ontvangt de domain objects en maakt daar dto’s van, ook ontvangt de service layer dto’s.

## Alternatieve oplossing

Een alternatieve oplossing zou zijn dat er een extra laag tussen dao en domain komt. Hierdoor heeft de dao geen kennis van de domain models.

## Huidige oplossing

De huidige oplossing is een stuk simpeler.

# 

# Deployment diagram

# 

## Requirements

* Apache TomEE plus server.
* De applicatie praat over HTTP met de cliënt.

## Ontwerpprincipes

* JSON over HTTP is een belangrijk ontwerpprincipe wordt gebruikt door de cliënt om requests te sturen en door de applicatie om data terug te sturen.

## Alternatieve oplossing

Er zou gebruik gemaakt kunnen worden van een andere database, bijvoorbeeld mongodb.

## Huidige oplossing

Ik gebruik sql vooral omdat we daarmee begonnen vanuit de opdracht dus in de toekomst is mongodb ook een optie.

# Ontwerp-/Craftmanship -keuzes

* Wanneer een token niet is meegegeven wordt dat gecheckt door een filter. Dit filter is bij elke route gezet waar token nodig is. Dit filter is zo gemaakt om dubbele code te voorkomen omdat je anders bij elke methode een check zou moeten neerzetten. Nu gaat dat automatisch.
* Voor elke DAO is een interface gemaakt om ervoor te zorgen dat wanneer er een nieuwe DAO wordt gemaakt na het implementeren van alle interface methodes er niks veranderd hoeft te worden aan de rest van de applicatie.
* Eigen exceptions worden gemaakt in de service layer en niet in de data access layer om een dependency tussen de exceptions en DAO te voorkomen.
* DTO is onderverdeeld in `request` en `response` omdat de cliënt een aantal andere datatypes gebruikt dan in de rest van de applicatie. Hierdoor worden de de request dto’s gebruikt om te ontvangen van de cliënt en de response dto’s om te sturen naar de cliënt.
* ID’s hebben het datatype string. Een ID bestaat uit 34 base64 chars waarbij `+` en `/` zijn vervangen met `-` en `\_` (dit om ze url proof te maken). De lengte van 34 is gekozen om zo nooit extra lengte toe te hoeven voegen omdat het aantal mogelijke ID’s nu is.

# Conclusie

Inmiddels heb ik uitgelegd en laten zien hoe de package structuur in elkaar zit daarnaast heb ik nog een alternatieve oplossing gegeven. In het hoofdstuk dat daarna kwam heb ik laten zien hoe de deployment in elkaar zit met ook daarbij een alternatieve oplossing voor de database. Als laatste hoofdstuk heb ik een aantal ontwerpprincipes opgenoemd.