Aaron van den Berg  
27-03-2020  
**Studentnummer:** 616650  
**Klas:** ITA-OOSE-A-s  
**Docent:** Michel Portier & Meron Brouwer

Opleverdocument

Oose - Spotitube

Aaron van den Berg

2020

Inhoudsopgave

[1. Opdrachtomschrijving 2](#_Toc36204687)

[2. Package diagram 3](#_Toc36204688)

[Uitleg package diagram 3](#_Toc36204689)

[Design patterns & ontwerpprincipes 4](#_Toc36204690)

[3. Deployment diagram 5](#_Toc36204691)

[4. Ontwerpkeuzes 6](#_Toc36204692)

[Services 6](#_Toc36204693)

[Util 6](#_Toc36204694)

[Exception mapper 6](#_Toc36204695)

# Opdrachtomschrijving

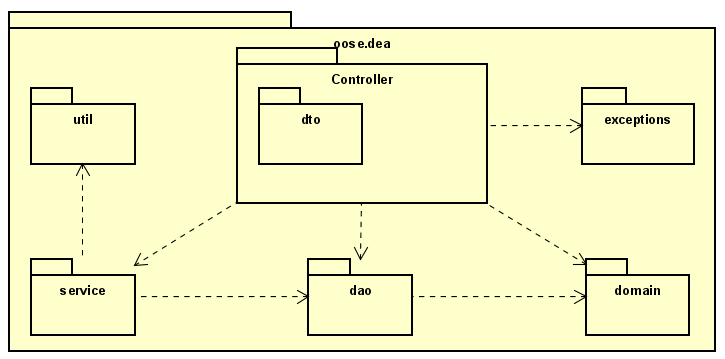
Voor de course DEA moet er een applicatie genaamd Spotitube worden gerealiseerd.

Spotify en Youtube hebben de handen ineengeslagen en werken gezamenlijk aan een app (Spotitube) waarmee een klant een overzicht kan krijgen van afspeellijsten met daarin audio- en videostreams. Ze willen eerst een deel van de back-end ontwikkelen en deze testen via een bestaande webapplicatie alvorens over te gaan tot de ontwikkeling van de app.

De applicatie maakt gebruik van de volgende APIs en frameworks:

* JAX-RS v2.0 (REST, JSON)
* CDI (Context & Dependency injection)
* JDBC API

# Package diagram



## Uitleg package diagram

**Controller 🡪 Exception**Een *controller* kan custom exceptions gooien, zoals *ForbiddenException* of *UnauthorizedException*.

**Controller 🡪 Domain**Een *controller* krijgt vanuit het *Data Access* *Object* (DAO) een domainobject terug. Het is aan de controller de taak om dit domainobject om te zetten naar een *Data Transfer Object* (DTO).

**Controller 🡪 DAO**  
Een *controller* vraagt aan het *Data Access* *Object* (DAO) om data op te halen of te wijzigen.

**Controller 🡪 Service**Een *controller*kan aan een service vragen om een bijvoorbeeld een token te verifiëren. De *service* voert het dan uit.

**Service 🡪 DAO**Een *service* kan data doorsturen naar een *Data Access* *Object* (DAO). Bijvoorbeeld als een service vraagt om een gebruikersgegevens te verifiëren. Als dit is gelukt kan een service bijvoorbeeld een *token* teruggeven.

**Service 🡪 Util**Een *service* kan gebruik maken van een *Utility (Util).* De enigste *Utility* wat in deze applicatie zit is een *Token Generator*.

**DAO 🡪 Domain**Een *DAO* maakt gebruik van een *domain (domainobject)* om dit terug te sturen naar de *Controller*. Vervolgens kijkt de *controller* dan wat die met het *domainobject* doet.

## Design patterns & ontwerpprincipes

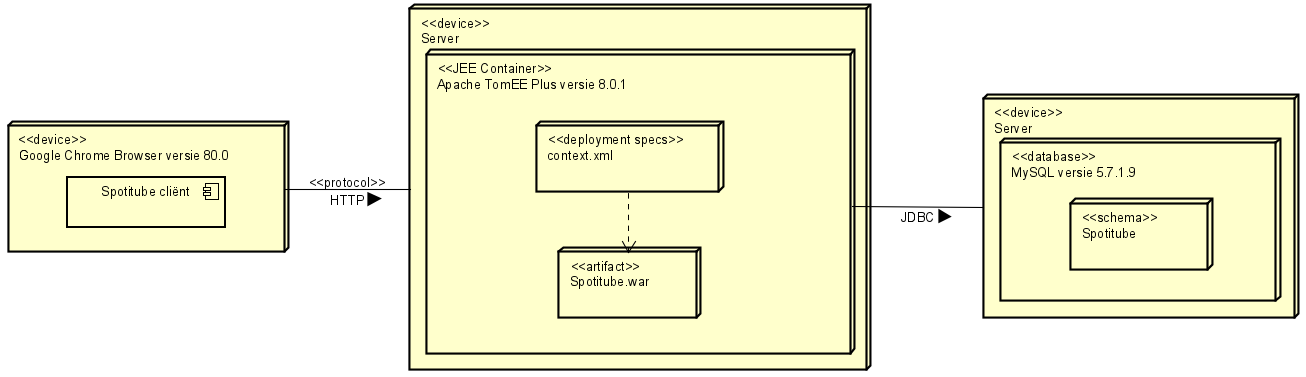
**Remote facade pattern**In de applicatie maak ik gebruik van het *Remote facade pattern*. Een controller krijgt een domainobject van een *Data Access* *Object* (DAO). De controller zet dit vervolgens om naar een Data Access Object (DTO). Deze DTO wordt dan teruggegeven aan de gebruiker, wat dan ook wel de presentatielaag genoemd.

**Custom exception mapper**De controller maakt gebruik van custom exceptions. Deze exceptions worden gebruikt in de controller als er iets niet goed gaat. Bijvoorbeeld als een gebruiker probeert in te loggen met foutieve gegevens, retourneert het systeem een UnauthorizedException. Via een *Exception Mapper* wordt er dan een statuscode, in dit geval statuscode 401 naar de gebruiker gestuurd. Door deze ontwerpkeuze ligt de verantwoordelijkheid niet meer bij de controller, maar bij de exception Mapper. Dit vind ik persoonlijk een goede oplossing en ook aardig ‘clean code’.

Een alternatief is om in de controller dit te regelen, en dan bijvoorbeeld geen exception te gooien maar gelijk de betreffende statuscode te retourneren. Bij kleine projecten zal dit waarschijnlijk geen probleem zijn, maar als je kijkt naar *Maintainability* en *Scalability* vind ik mijn oplossing toch beter.

**Dependency Injection**Om de koppeling tussen de lagen laag te houden heb ik gebruik gemaakt van *dependecy injection*. Hierdoor kan ik in de toekomst makkelijk switchen naar een andere DAO, bijvoorbeeld met NoSQL door deze te injecteren. Ook omdat ik *dependency injection* gebruik zijn de lagen gemakkelijker te testen.

# Deployment diagram



Het deployment diagram bestaat uit drie devices:

* Een cliënt device (waarop het browsers staat)
* Een server waarop de Apache TomEE server draait
* Een server waarop de MySQL database draait.

De client Spotitube draait op een webbrowser. Spotitube communiceert via REST met het http-protocol naar de Apache TomEE server, waarop de artifact Spotitube.war draait.

De Apache TomEE server communiceert met de MySQL database server via JDBC. De connectie wordt geregeld via een container in TomEE. De MySQL database zijn geconfigureerd in de context.xml. Doordat TomEE de database connectie regelt, hoef je alleen nog maar in de applicatie de datasource te injecteren.

Een alternatieve manier was om de database connectie in de applicatie te regelen. Waardoor de applicatie verantwoordelijk is voor connectie. Dit vind ik minder mooi, omdat dit niet het doel is van de applicatie en meer het doel TomEE. Ook scheelt dit weer code en dus extra testen.

# Ontwerpkeuzes

## Services

In de Spotitube applicatie heb ik gebruik gemaakt van twee services:

* Login service
* Token service

**Login service**Het doel van de login service is overnemen van verantwoordelijkheden van de controller. Hierdoor blijft de controller lekker *clean*.

De login service krijgt vanaf de controller een gebruikersnaam en wachtwoord mee. De login service vraagt aan de *UserDAO* of de gegevens overeenkomen met een gebruiker in de database. Als de gegevens overeenkomen retourneert de login service een *TokenDTO* terug naar de controller. De *TokenDTO* bestaat uit een gebruikersnaam en een uniek gegenereerde token.

Ik heb deze oplossing bedacht omdat alle controllers in de applicatie vrij simpel zijn vergelijken met de login controller. Daarom heb ik dus een service bedacht voor de login controller die de meeste logica wegneemt.

**Token generator**Het doel van de token service is ook het overnemen van verantwoordelijkheden van alle controllers. Bij de meeste controllers moet steeds worden gecheckt of de token wel juist is. Omdat dit dus vaak voorkwam heb ik ervoor gekozen hier een service voor te maken.

Omdat hier nu een service voor is, kan er in de toekomst makkelijk wijzingen worden toegepast aan die klasse in plaats van elke controller te hoeven aanpassen.

## Util

Er is een package *util* aangemaakt in de applicatie waarin de Token Generator klasse in zit. Dit is simpele klasse die alleen een random token genereerd van 36 karakters.

Omdat de token generator klasse niet echt specifiek bij deze applicatie hoort heb ik hem dus in een util package gestopt. Hier kunnen in de toekomst alle kleine niet-gerelateerde applicatie extensies en/of plug-ins in.

## Exception mapper

Doordat ik een Exception mapper heb gebruikt neem ik verantwoordelijkheden weg in de controller. Waardoor de controller nog *cleaner* wordt.

Omdat ik een exception mapper heb aangemaakt kon ik custom exceptions aanmaken voor elke gebruikte statuscode. Hierdoor heb ik een nette error afhandeling gerealiseerd. In de exception mapper wordt dan de juiste statuscode teruggestuurd.