**SPOTITUBE**

Azez Salami

618646

Vak: DEA

Datum: 22-05-2020

Versie 2.0

Inhoudsopgave

[1. Inleiding 3](#_Toc41035978)

[2. Package diagram 4](#_Toc41035979)

[3. Deployment diagram 5](#_Toc41035980)

[4. Ontwerpkeuzes 6](#_Toc41035981)

[4.1 Design choices and patterns 6](#_Toc41035982)

[4.2 Het Testen: 6](#_Toc41035983)

[5. Slot 7](#_Toc41035984)

# Inleiding

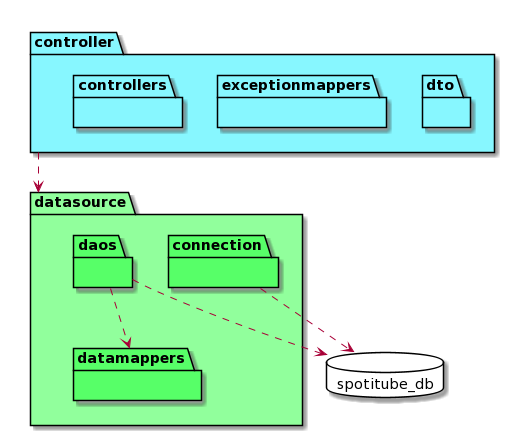
Dit document dient als ontwerp over hoe de te realiseren back-end API van Spotitube applicatie moet gaan werken.

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.

Een Playlist heeft een naam en een eigenaar, een eigenaar heeft een gebruikersnaam en wachtwoord. Er kunnen 2 soorten tracks in een playlist worden opgeslagen namelijk liedjes (song) en filmpjes (video). Een track heeft een performer, titel, url en afspeelduur. Een song heeft naast alle eigenschappen van een track een album, een video heeft naast alle eigenschappen van een track een publicatiedatum en een beschrijving. Elke track kan offline of online beschikbaar zijn in een specifieke playlist. De applicatie moet meerdere verschillende relationele databases ondersteunen. Het wisselen van database moet mogelijk zijn zonder de applicatie opnieuw te moeten compileren.

# Package diagram

Het onderstaande package diagram illustreert de indeling van de applicatie op package-niveau.



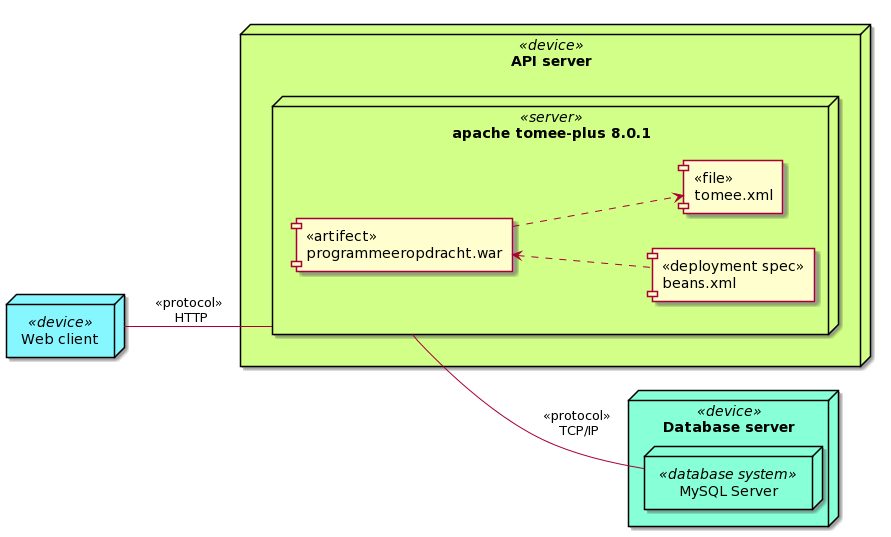
Figuur 1 Package diagram

Tijdens het opzetten van de architectuur van de applicatie heb ik besloten om de packages te groeperen op serviceniveau. Het grote voordeel hiervan is dat als de applicatie later moet worden onderhouden, is het makkelijk om te vinden welke package moet aangepast worden. Een nadeel van deze aanpak is dat er bij een wijziging aan het domein in bijna alle bovenliggende packages aanpassingen gemaakt moeten worden

Een alternatief is om de packages te groeperen op functionaliteit. Dit heeft als een voordeel dat de bovenste laag packages heel losgekoppeld is.

# Deployment diagram

Het onderstaande deployment diagram laat zien hoe de applicatie uiteindelijk draait op een deployment-server.



Figuur 2 Deployment diagram

# Ontwerpkeuzes

### Design choices and patterns

* Ik hen gebruik gemaakt van exception mapper. Hierdoor kunnen de exception goed afgevangen wordt zonder veel code duplicatie.  Ook kan op een eenvoudige manier een Response teruggegeven worden aan de eindgebruiker.
* Daarnaast is er gebruik gemaakt van twee lagen. De controller en datasource laag. Door de API op te splitsen in deze lagen zijn de verschillende onderdelen minder afhankelijk van elkaar.
* Ik heb gekozen om MySQL server te gebruiken voor mijn database. Er zijn niet zo veel verschillen tussen MySQL Server en SQL Server maar ik heb MySQL gekozen omdat ik een kleine ervaring daarmee wil opdoen.

### Het Testen:

* Ik heb DTOs-classes niet getest omdat die classes bevatten alleen maar getter en setter methodes van lokale variabels die altijd werken en dat is niet nuttig om te testen.
* DatabaseConnection en DatabaseProperties classes heb ik niet getest omdat het lijkt dat is moeilijk om te testen door middel van junit unit testing.
* Ik heb van DAOs-classes alleen één class getest omdat het testen van DAOs-classes is lastig en dat zou zoveel tijd kosten.

# Slot

In dit verslag heb ik een overzicht gegeven van de opdracht. Ik heb een package diagram en deployment diagram gemaakt. Ik legde ook mijn keuzes uit.