Spotitube

By Dani Botland (1617442)

datum: 22-03-2024

vak: OOSE-DEA

docent: Bart van der Wal

versie: 2.0

# Inhoud

**Table of Contents**

[Inhoud 2](#__RefHeading___Toc105_810035096)

[Inleiding 3](#__RefHeading___Toc107_810035096)

[Package Diagram 4](#__RefHeading___Toc138_791326171)

[Deployment Diagram 6](#__RefHeading___Toc109_791326171)

[Ontwerpkeuzes 7](#__RefHeading___Toc113_810035096)

[Datamappers 7](#__RefHeading___Toc111_791326171)

[Interfaces 7](#__RefHeading___Toc156_791326171)

[Setter Injection 7](#__RefHeading___Toc113_791326171)

[CORS Filter 7](#__RefHeading___Toc115_791326171)

[Performance consideration 7](#__RefHeading___Toc117_791326171)

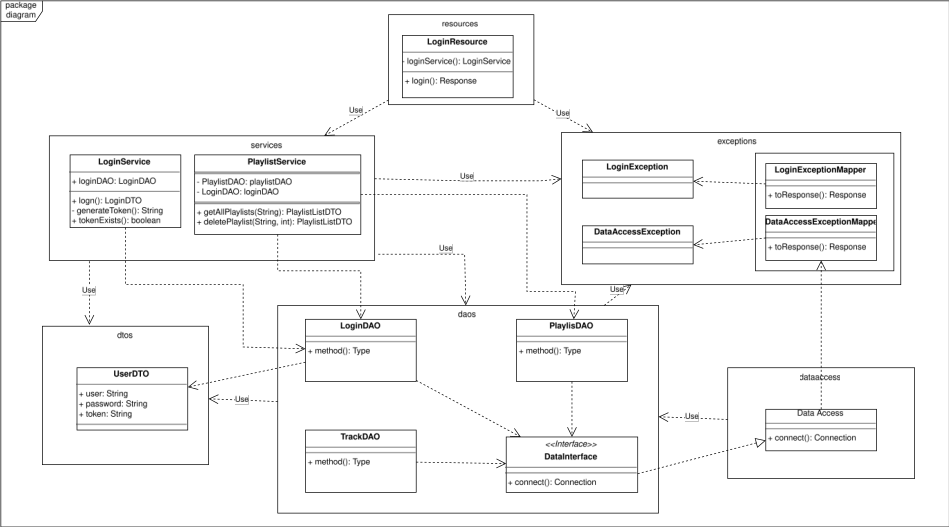
[Bronnen 8](#__RefHeading___Toc119_791326171)

# Inleiding

Op de opleiding HBO-ICT hebben wij tijdens het OOSE semester de spotitube opdracht gekregen. Hierbij gaat het om een casus waarbij youtube en spotify de handen in een hebben geslagen en een gezamelijke site hebben opgericht waar je naar muziek kunt luisteren en naar video kunt kijken. In dit document beschrijf ik mijn uitwerking van de casus en geef ik eventuele toelichting waar nodig.

# Package Diagram

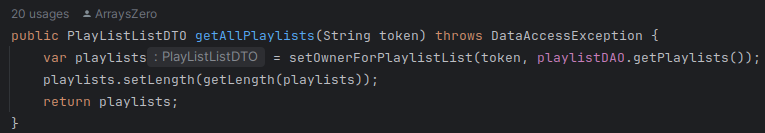
Hierboven is te zien het package diagram van de spotitube applicatie, hierin heb ik geprobeerd te tonen met welke lagen ik werk (resource, service en data laag). Met ieder een eigen opdracht voor de gehele applicatie.

Figure 1: Package diagram

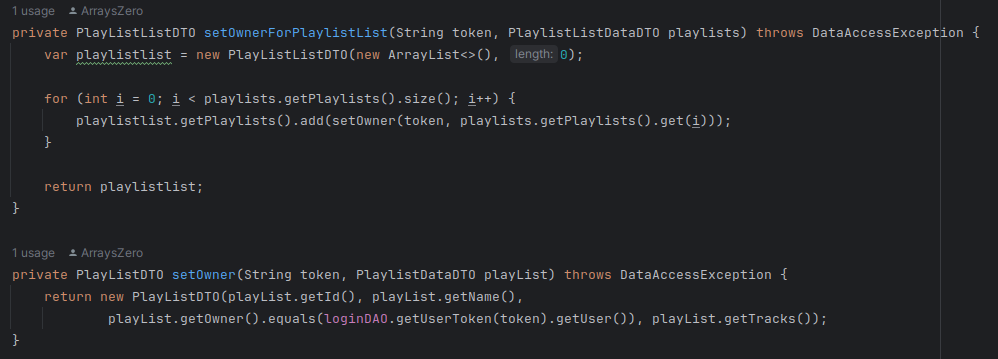
De resource laag houd zich bezig met het ontvangen en reageren op inkomende HTTP requests, hiervoor zijn drie classes in gebruik Login-, Playlist- en TrackResource. Deze laag bevat vrij weinig logica, het delegeert alleen maar wat er moet gebeuren door naar de Service laag. Eerst de authenticatie van de request en daarna het daadwerkelijke verwerken van de request. Zoals te zien is in de voorbeeld functie hieronder

Figure 2: getAllPlaylists in PlaylistResource.java

Als tweede laag hebben we de service laag, deze is verantwoordelijk voor de businesslogica en delegeert het opslaan en ophalen van de data door naar de data laag. Een voorbeeld hiervan is het ophalen van alle playlists en of de gebruiker de eigenaar is van de betreffende playlist.

Figure 3: getAllPlaylists in PlaylistService.java

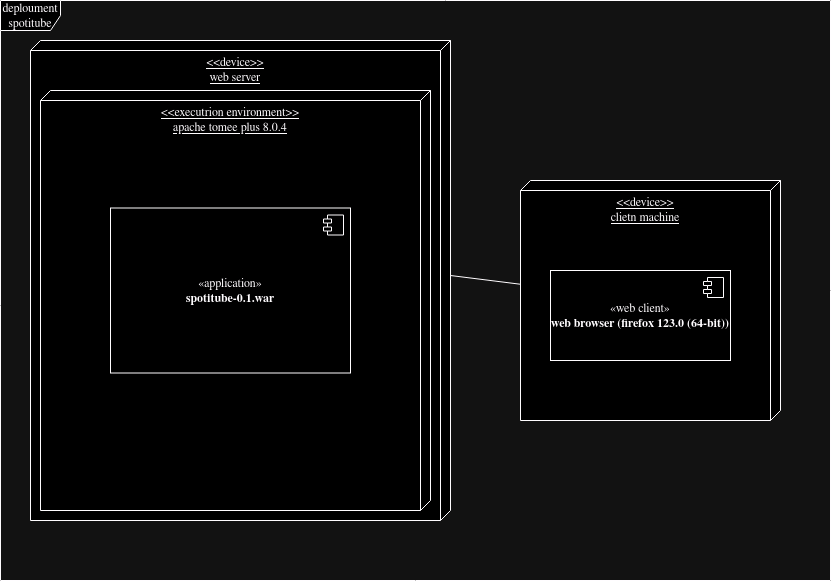
In de twee figuren hier is te zien hoe de vollegige lijst van Playlists wordt opgehaald. Zoals te zien wordt eerst de lijst van Playlists opgehaald door de PlaylistDAO, deze geeft echter een gebruiker door als eigenaar, niet een boolean. Om hier een boolean van te maken wordt de lijst van playlists doorgespeeld aan de functie setOwnerForPlaylists, deze vergelijkt de ‘owner’ String en de gebruiker om te bepalen of de gebruiker de eigenaar is. Voor nu wordt bij elke playlist opnieuw gekeken wie de gebruiker is, iets wat ik besloot te doen omdat ik met een H2 db werk. Bij een externe DB, zoals MySQL of MSSQL zou ik dit anders oplossen om performance redenen.

Figure 4: setOwnerForPlaylistList & setOwner in PlaylistService.java

Als laatste is de data laag hebben we de data laag. Deze laag is verantwoordelijk voor het ophalen van de data en het omzetten in POJO’s zodat de service laag ermee aan de slag kan om het door te sturen naar de resource laag. De belangrijkste classes hier zijn de DAO’s die de verbinding naar de database opzetten en de juiste query uitvoeren, elke functie doet ook niet meer dan het uitvoeren van een query en het resultaat als DTO terug sturen naar de service laag waar het verder wordt verwerkt. Zoals te zien is in PlaylistDAO.java:

Figure 5: getPlaylists in PlaylistDAO.java

# Deployment Diagram

Figure 6: deployment diagram

Zoals te zien is in het diagram hierboven zijn er twee devices die mee doen aan deze applicatie. Dit zijn de webserver en de client, er is bewust gekozen om geen aparte database server te gebruiken omdat ik deze opdracht meer als POC heb gedeveloped. Om dezelfde reden zit de database dus ook in de .war bij in begrepen.

Verder draaien in dit geval beide devices een arch based linux OS met kernel 6.7.x, maar de client zou ook een ander OS kunnen draaien mits het toegang heeft tot een web browser. De war draait op een apache tomEE plus server versie 8.0.4.

# Ontwerpkeuzes

In dit deel ga ik een aantal ontwerpkeuzes bespreken waarvan ik denk dat ze wat meer aandacht vereisen

## Datamappers

Een keuze die misschien wel opvalt is het gebruik vann datamappers, ik gebruik ze wel voor de exceptions, maar niet voor het retourneren van daadwerkelijke data. Dit komt omdat een DTO voor elke request anders wordt gebruikt en de response vaak ook een andere HTTP code bevat, waardoor een enkele mapper niet voldoende zou zijn, waardoor het in mijn ogen voor te veel boiler plate zou leiden. Exceptions zijn een ander verhaal, want wanneer een exception wordt gegooid, geeft die exception altijd dezelfde HTTP code terug, alleen de message is anders. Dit was makkelijk om te zetten in een enkele mapper voor elke exception waardoor ik niet in de resource classes gedrag moest uitwerken voor de verschillende exceptions.

## Interfaces

Zoals te zien is, maak ik vrijwel geen gebruik van interfaces in deze applicatie. Alleen de DataAccess class zit achter een interface. Dit is omdat ik van mening ben dat het pas zin heeft om een interface te gebruiken als het meerdere opties webabstracteerd voor de class die de functie aanroept (zoals in het adapter pattern (Iluwatar, n.d.-a)) . Ik heb wel een interface gebruikt, voor de verbinding naar de database. Doordat ik het seperated interface pattern heb gebruikt zorg ik ervoor dat ik de data source kan veranderen zonder dat ik de code om een verbinding op te zetten hoef aan te raken (Iluwatar, n.d.-b; *P Of EAA: Separated Interface*, n.d.).

## Setter Injection

Dit is een opmerkelijke keuze als je door mijn code kijkt, al was de keuze vrij simpel. Voor een aantal classes moest het op deze manier, ik koos voor uniformiteit en heb er dus voor gekozen dit vrijwel overal te doen.

## CORS Filter

Er zijn een aantal opties om de CORS Filter toe te passen, ik heb gekozen voor de variant waar het te regelen via Javax. Hiermee hoefde ik maar een functie te overschrijven uit de ContainerResponseFilter class, waardoor alles op een plek zit. Wanneer ik het gedrag wil veranderen kan ik dit dan in deze class doen te hoeven zoeken.

## Performance consideration

Zoals eerder in het document is benoemd heb ik deze opdracht uitgewerkt als POC, hierom gebruik ik een H2 database als SQL database. Dit betekent dat alle data al binnen de applicatie zit en het uitvoeren van queries in een lus een vrij kleine performance hit geeft. Hierom heb ik op een aantal plaatsen een Query in een lus staan, bij een volledige database zou ik dit waarscheinlijk oplossen met een join of een stored procedure.

# Bronnen

Iluwatar. (n.d.-a). *Adapter*. Java Design Patterns. https://java-design-patterns.com/patterns/adapter/#explanation

Iluwatar. (n.d.-b). *Separated interface*. Java Design Patterns. https://java-design-patterns.com/patterns/separated-interface/#class-diagram

*P of EAA: Separated Interface*. (n.d.). https://martinfowler.com/eaaCatalog/separatedInterface.html#