Opleverdocument Spotitube

Door: Stijn Arts (634711)

Docenten: Meron Brouwer

Datum: 16/03/2023

Opleiding/Hogeschool: HBO-ICT bij de Hogeschool van Arnhem en Nijmegen

Vak: OOSE-DEA

Icon

Description automatically generated     
 

Versie 1

Inhoud

[Inleiding 3](#_Toc129875078)

[Package Diagram 4](#_Toc129875079)

[Deployment Diagram 5](#_Toc129875080)

[Ontwerpkeuzes 6](#_Toc129875081)

[Service laag 6](#_Toc129875082)

[SQLParameter interface 6](#_Toc129875083)

[Unit testen 6](#_Toc129875084)

[PreparedStatementHelper class 6](#_Toc129875085)

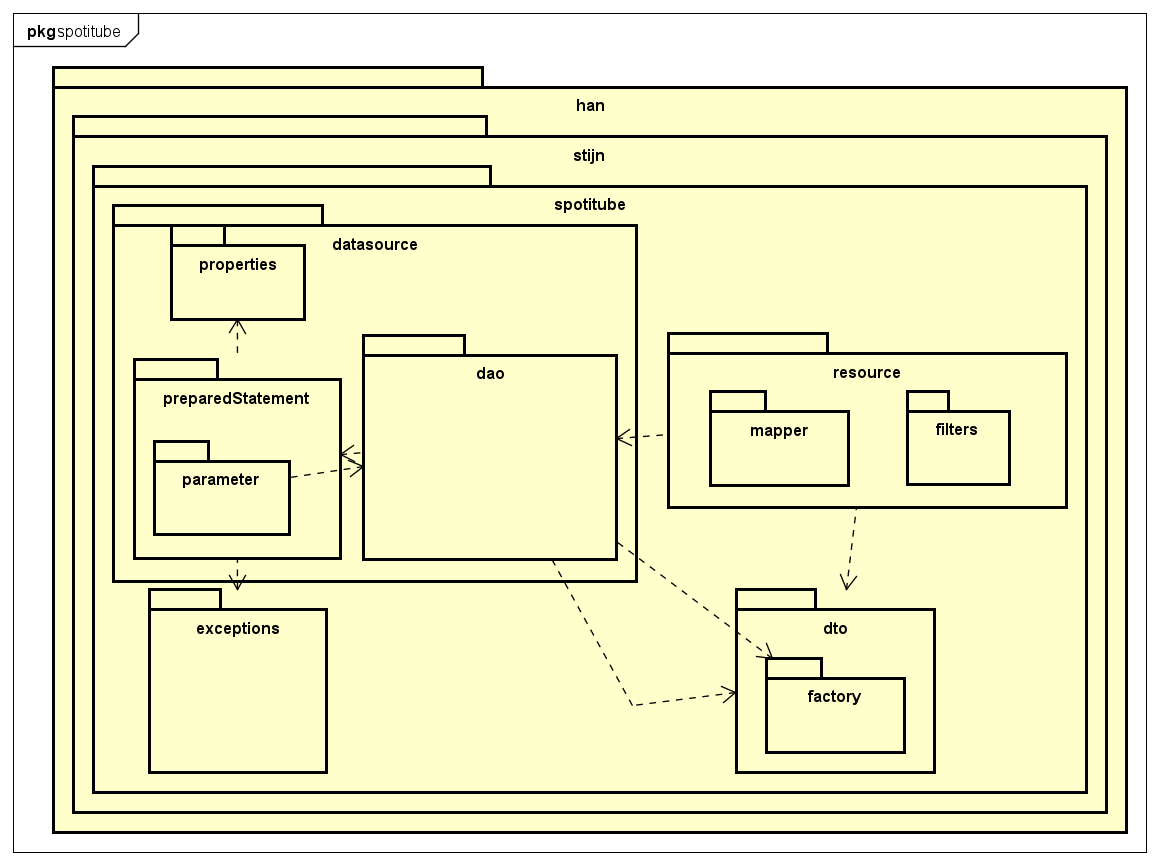
[Resultset reader class 6](#_Toc129875086)

# 1. Inleiding

Voor de nieuwe service van spotify en google (spotitube) werd mij gevraagd een backend te ontwikkelen voor de bestaande client. In dit document worden de package en deployment diagrammen voor de backend beschreven, en de ontwerpkeuzes voor de spotitube backend toegelicht.

# 2. Package Diagram

Hieronder staat de package diagram van de spotitube backend (figuur 1).



Figuur . De package diagram van de spotitube backend.

# 3. Deployment Diagram

Hieronder is de deployment diagram van de spotitube backend te zien (Zie Figuur 2.). De server waar spotitube.war op draait moet voorzien zijn van Java 17.0.2 en de bijbehorende Wildfly versie (25.0.0 final). Verder moet er een database beschikbaar zijn op poort 1433 via TCP/IP die draait op een server waar Microsoft SQL Server (2019 – 15.0.2101.7 x64) op draait.

Diagram

Description automatically generated

Figuur . Deployment Diagram van de spotitube Backend

# 4. Ontwerpkeuzes

## 4.1 Service laag

Ik heb ervoor gekozen om de service laag uit de backend te halen. In de applicatie werd er niks zinnigs binnen de service laag gedaan en produceerde het alleen 3 middleman classes. Omdat dat een bad smell is heb ik die verwijderd en de refactoring strategie ‘Remove Middle Man’ (Refactoring Guru, 2014) toegepast.

## 4.2 SQLParameter interface

Om het testen van de DAO zonder extra dependencies mogelijk te maken heb ik een helper klasse geschreven (PreparedStatementHelper) die prepared statements kan maken en uitvoeren. Het meegeven van parameters aan de prepared statement gebeurt door middel van een SQLParameter klasse. Eerst waren deze opgesplitst in de StringParameter, IntegerParameter en BooleanParameter, zonder interface. Dit bleek echter code smells op te leveren (vergelijkbaar met switch statements), dus is er een interface gecreeerd (SQLParameter) die de parameter klassen implementeren. Met polymorfie is daarmee de code smell verholpen.

## 4.3 Unit testen

Er is een code coverage van 87% bereikt, en 98% van alle paden zijn belopen met de geschreven unit tests. Hieronder worden de uitzonderingen en ontwerpkeuzes rondom unit testen toegelicht.

### 4.3.1 PreparedStatementHelper class

De PreparedStatementHelper class is niet uitbundig getest. Dit is niet gedaan omdat er in de h2 framework op het moment een vulnerability zit. Zonder h2 moest iedere methode in zijn geheel gemockt worden met Mockito, en dat zou niet zo zinvol te zijn.

### 4.3.2 Resultset reader class

De ResultsetReader class was geïntroduceerd toen de noodzaak om Data Access Objects te testen verscheen. Met deze class werd het mogelijk om een veld van een ResultSet individueel uit te lezen, waardoor de return waarde ook met Mockito gemockt kon worden voor het maken van unit tests.

### 4.3.3 OpenPojo

Voor het testen van de POJO’s in de backend is gebruik gemaakt van de framework OpenPojo. OpenPojo wordt hier gebruikt om automatisch de getters en setters van de POJO’s te testen.

# 5. Bronnen

Refactoring Guru. (2014). *Remove Middle Man*. Opgehaald van Refactoring Guru: https://refactoring.guru/remove-middle-man