DOCUMENTATIE SPOTITUBE

Bram van den Bogaard (585902)

Titelblad

Titel: Documentatie Spotitube

Plaats: Arnhem

Vak: OOSE-DEA

Docent: Michel Portier

Auteur: Bram van den Bogaard (studentnummer 585902)

Datum: 26 maart 2020

Versie: 1.0

Inhoudsopgave

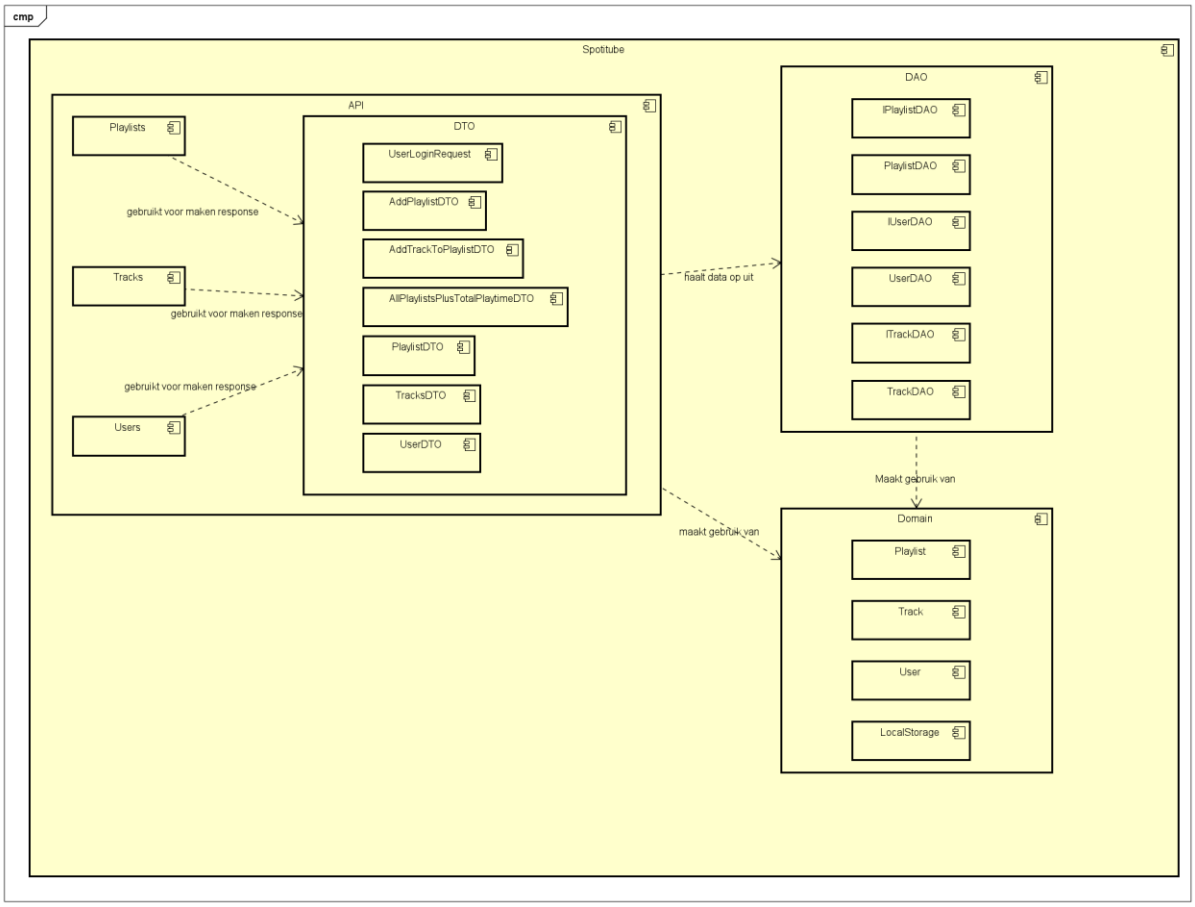
Inleiding

Voor het vak DEA wordt gevraagd om een applicatie te ontwikkelen voor een reeds bestaande frontend (geschreven in Angular) voor een fictief bedrijf genaamd Spotitube. Deze applicatie moet requests vanuit de Angular-applicatie verwerken en aan de hand van het request een bepaalde JSON-response teruggeven. Dit gebeurt door het schrijven van een Java applicatie die draait op een TomEE-webserver.

De Java-applicatie verwerkt een aantal requests vanuit de frontend en maakt aan de hand hiervan aanpassingen aan een MySQL-database. Vervolgens wordt er een response gestuurd.
Ook moet deze applicatie goed ontworpen en getest zijn.

Om aan te tonen dat de applicatie daadwerkelijk van goede kwaliteit is worden in dit document een package diagram en deployment diagram beschreven en worden enkele ontwerpkeuzes toegelicht.

1 Package diagram

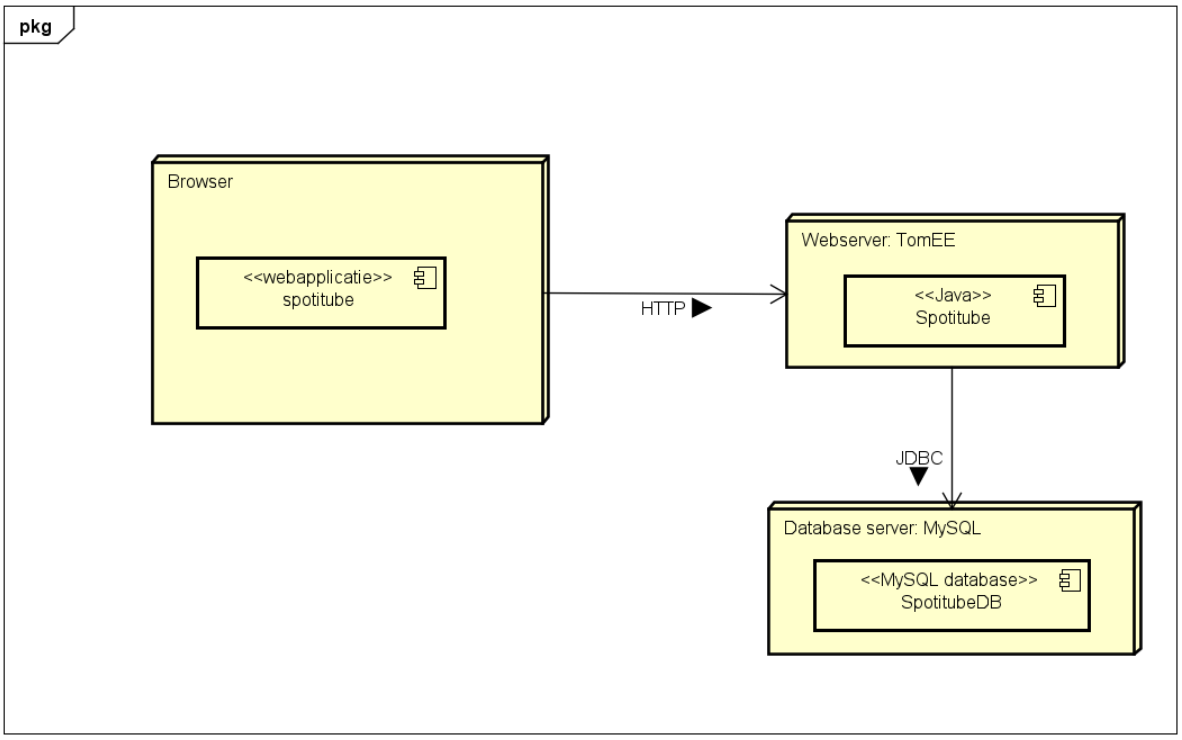


Figuur 1 Package diagram

In de bovenstaande afbeelding is te zien dat er drie duidelijke packages zijn: API, DAO en Domain. De API laag bevat klasses die een request ontvangen vanuit de frontend en deze vervolgens verwerken. Hiervoor wordt data opgehaald uit de DAO-laag. De DAO-laag voert vervolgens SQL-queries uit op een database en geeft aan de hand hiervan data terug. Deze data wordt gebruikt in Data Transfer Objects (DTO's) waar vervolgens een response mee gemaakt wordt die de frontend ontvangt.

Ook maken de API en DAO laag beide gebruik van klasses uit de Domain laag, deze laag bevat de zogenaamde Plain Old Java Objects, ook POJOs genoemd *(Fowler, z.d.)*.

2 Deployment diagram



Figuur 2 Deployment diagram

In de bovenstaande afbeelding is te zien dat er drie verschillende componenten gebruikt worden. Een webapplicatie (draaiend op een server van de ICA), een TomEE webserver en een Database server.

De webapplicatie bevat een frontend gemaakt met Angular die voor gebruikers toegankelijk is. Wanneer de gebruiker input geeft wordt met HTTP een request gestuurd naar de TomEE webserver. Deze webserver bevat de Java Spotitube applicatie als een war.

Vanuit de TomEE webserver wordt er met JDBC verbinding gemaakt met een MySQL database server waarop een database draait met de data die gebruikt wordt voor de Spotitube applicatie.

Er wordt gebruik gemaakt van een MySQL database omdat ik daar bekend mee ben en daardoor gemakkelijk en snel een database kan maken.

3 Ontwerpkeuzes

Gedurende het ontwerpen en programmeren van de applicatie die in dit document beschreven wordt zijn vanzelfsprekend meerdere ontwerpkeuzes gemaakt. In dit hoofdstuk worden enkele hiervan toegelicht: Het lagenmodel (layer-pattern), het gebruik van een LocalStorage klasse en het hergebruiken van functies.

3.1 Layer pattern

Het layer-pattern is een architectural pattern die gebruikt wordt voor het verdelen van een applicatie in verschillende lagen. *(Hogeschool van Arnhem en Nijmegen, 2020)*
Hierbij mag elke laag gebruik maken van de laag onder zich maar niet omgekeerd.
Gebruikelijk is dat de lagen verdeelt zijn in: presentational, logica, domain en dao.

Echter heb ik de keuze gemaakt om geen gebruik te maken van een logica laag. Dit heb ik gedaan op advies van mijn docent en omdat er weinig logica toegepast wordt in de applicatie. Er wordt namelijk vooral data ontvangen (presentational layer, in dit geval API genoemd), aan de hand waarvan database calls worden uitgevoerd (dao layer).

3.2 LocalStorage

De frontend stuurt in veel gevallen een request aan de hand waarvan iets aangepast moet worden en vervolgens een hele lijst moet worden teruggegeven. Een voorbeeld hierbij is het toevoegen van een playlist. Hierbij bevat het request een nieuwe playlist maar verwacht de frontend vervolgens alle playlists (inclusief de zojuist toegevoegde).

Een eis aan de applicatie is dat er gebruik gemaakt moet worden van een token voor iedere gebruiker die opgeslagen wordt in de applicatie. Om de performance te verbeteren heb ik ervoor gekozen om dit uit te breiden met de klasse LocalStorage. In plaats van alleen de tokens van gebruikers op te slaan slaat deze klasse ook alle playlists op waardoor niet bij elke request een nieuwe database call hoeft te worden uitgevoerd om het geheel op te halen.

Wanneer bijvoorbeeld een request gestuurd wordt voor het aanpassen van een playlist wordt niet eerst een database-call uitgevoerd om alle playlists op te halen maar worden deze uit de LocalStorage klasse gehaald.

3.3 Hergebruiken van functies

Zoals in de vorige alinea is aangegeven verwachten veel requests een response waarbij een hele lijst aan objecten wordt teruggegeven in plaats van alleen een enkele aangepaste variant. Om ervoor te zorgen dat de code voor het opvragen van een lijst niet bij elke functie hoeft te worden gekopieerd zijn er functies die teruggegeven worden door andere functies.

Een voorbeeld hiervan is de functie “getAllPlaylists”. Deze functie geeft alle playlists uit de database terug. Wanneer een playlist toegevoegd moet worden wordt de functie “addPlaylist” aangeroepen. Ook deze functie moet alle playlists teruggeven. Daarom heb ik ervoor gekozen om in deze functie de functie “getAllPlaylists” te gebruiken voor de return waarde.

Bronvermelding

Fowler, M. (sd). *POJO*. Opgehaald van martinfowler.com: https://www.martinfowler.com/bliki/POJO.html

Informatica Communicatie Academie. (2014, juli 1). Controlekaart documenten. Arnhem, Gelderland, Nederland.

Hogeschool van Arnhem en Nijmegen. (2020, 26 maart). *Het layer-pattern* [Presentatieslides]. Geraadpleegd van https://onderwijsOnline.nl