Datum: 08-05-2020
Opdracht: Programmeeropdracht EAI
Course: OOSE DEA
Docent: Meron Brouwer
Student: Thomas Brandhorst (586637)

Inhoudsopgave

1. Inleiding

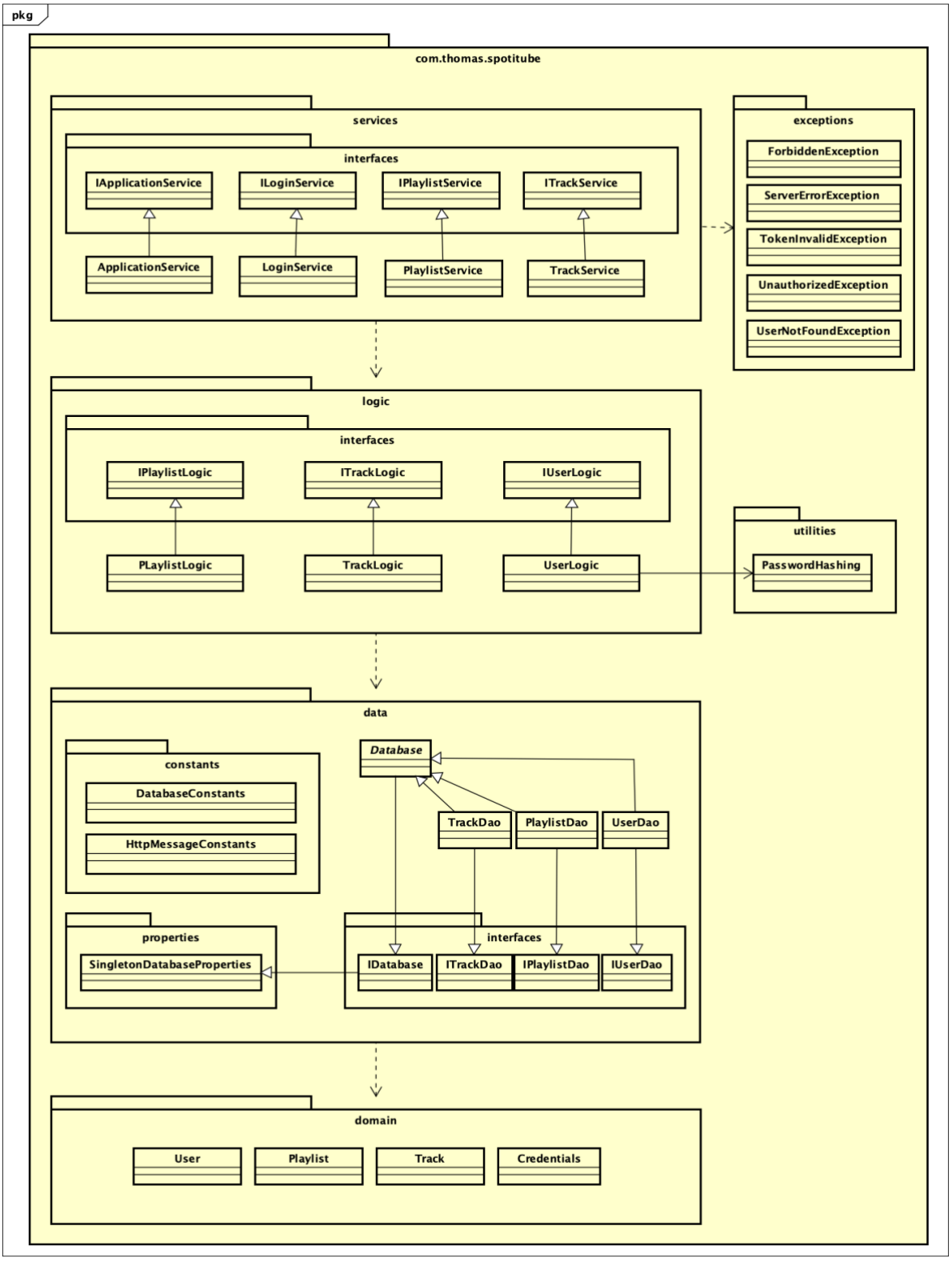
Dit document dient als documentatie voor het Java prototype van Spotitube. Dit document zal verscheidene aspecten van de software bespreken, zoals de architectuur door middel van een package- en een deployment diagram. Ook worden er bepaalde ontwerpkeuzes hierbij besproken.

2. Package diagram

In dit hoofdstuk wordt het package diagram besproken. Een package diagram biedt een overzicht van de structuur van de software. Op de volgende pagina wordt dit diagram getoond.

Een noemenswaardig punt bij het diagram is dat de “presentation layer” is weggelaten om de eenvoudigheid te bewaren. Deze laag zal bovenaan het diagram komen te staan. Dit is de spotitube client die is aangeleverd en uiteindelijk praat met de “service layer” van de software.

2.1 Diagram



Zoals te zien is, zijn de verschillende lagen (layers) duidelijk zichtbaar en toegepast in de spotitube software. Ook is er gebruik gemaakt van interfaces en password-hashing.

2.2 Requirements

Requirement	Onderbouwing
Dependency injection, geïmplementeerd met CDI	Er wordt door de applicatie heen de @Inject decorator gebruikt om de verschillende klassen te injecteren.
Data Mapper, Identity Map en Separated Interface (ISP), werkend met JDBC en MySQL	De "Dao" klassen in de "data" package fungeren als de DataMapper klassen. Deze klassen zorgen ervoor dat de data uit de database wordt opgehaald en omgezet wordt naar een POJO. De POJO is de Identity Map. De "Dao" klassen hebben ieder een eigen interface welke zij implementeren. Dit is handig zodat er gemakkelijk gewisseld kan worden met functionaliteit.
Service Layer/Domain Layer	Er wordt onderscheid gemaakt tussen de meerdere lagen van de applicatie op basis van de functionaliteit. Zo worden de DAO-, Business Logic- en Serviceklassen onderscheiden.
Remote Facade	De Remote Facade wordt geïmplementeerd middels de "Service" klassen in de "service" package. Deze verzorgen de endpoints en het versturen/ontvangen van data.

2.3 Design patterns

Er worden een aantal design patterns gebruikt in het bovenstaande diagram. De meesten zijn al genoemd in het kopje hierboven.

- Dependency Injection middels CDI
- Data Mapper, Identity Map en Separated Interface (ISP), werkend met JDBC en MySQL
- Service Layer
- Remote Facade
- Singleton
    - De “SingletonDatabaseProperties” klasse wordt gebruikt om het “properties” bestand in te laden waarmee een database connectie gemaakt wordt. Deze klasse is maar een keer nodig, en voorkomt het dubbel inladen (of onnodig inladen) van dit soort bestanden.

2.4 Alternatieve oplossingen

De huidige database implementatie kan vervangen worden. Dit betreft nu een traditionele relationele database, namelijk MySQL. Dit kan vervangen worden door een modernere NoSQL Database, zoals bijv. MongoDB. Een voordeel aan zo'n database is dat de complexiteit van de queries minder zal worden, en het daardoor eenvoudiger wordt om met de database te werken.

In de software is een "logic" laag ingebouwd, om zo de functionaliteit te scheiden van de "service" en "dao" lagen van de software. Dit zou weggelaten kunnen worden, zodat de software minder groot wordt.

Er is gebruikt gemaakt van het "hashing" algoritme "PBKDF2", om de wachtwoorden van gebruikers ge-hashed op te slaan in de database en zo ook te ontcijferen als zij willen inloggen. Een alternatieve oplossing hiervoor zou het "bcrypt" algoritme zijn, welke veel sneller en veiliger zou zijn dan het "PBKDF2" algoritme.

2.5 Onderbouwing gemaakte keuzes

Vanuit de opdrachtgever is er een requirement gemaakt om gebruik te maken van een relationele database, in dit geval MySQL. Het is daarom niet logisch om een andere database implementatie te gebruiken voor deze software.

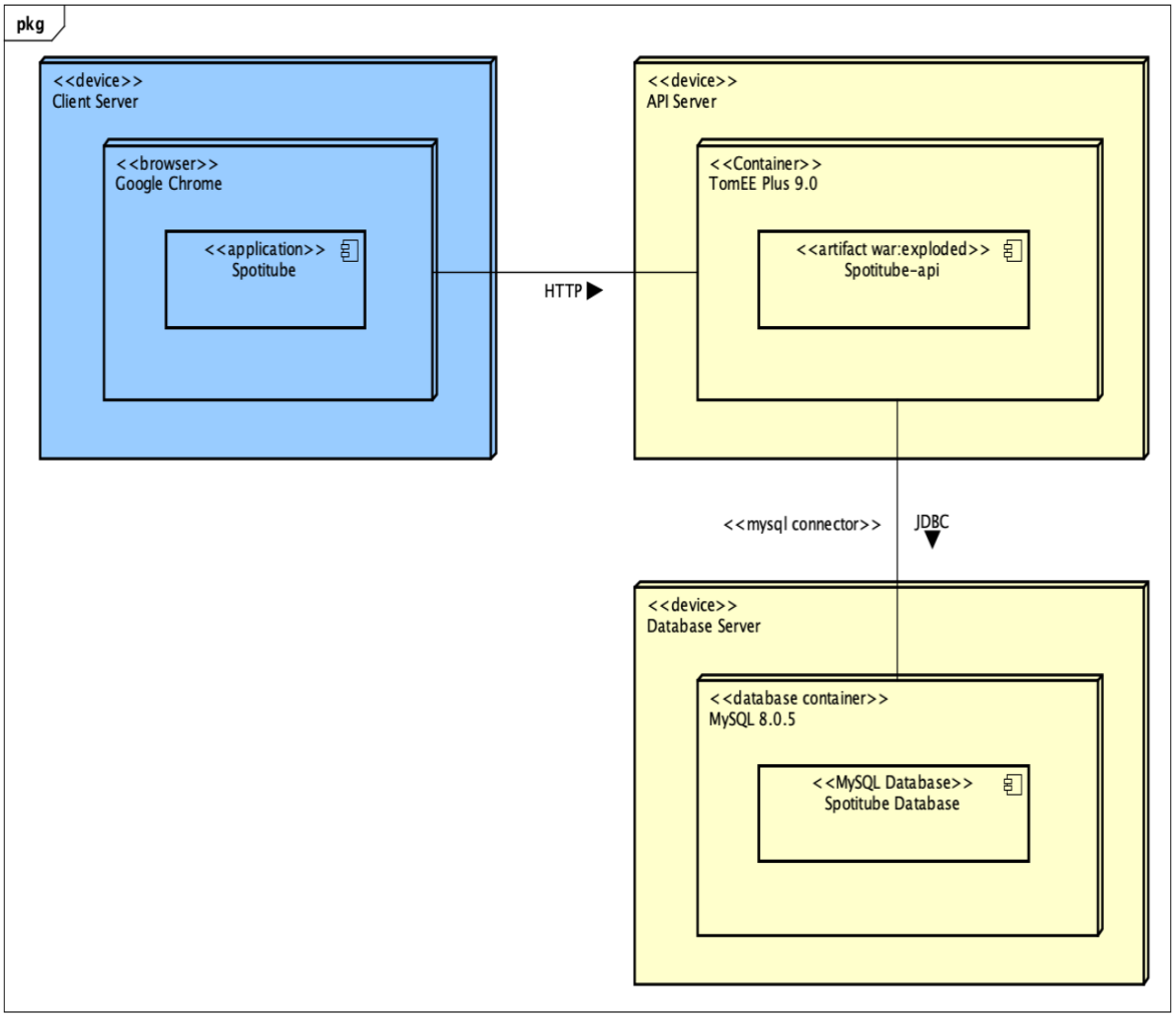
De extra "logic" laag zorgt voor meer scheiding van functionaliteit in de software. Nu is de "service" laag er puur om de endpoints af te handelen, en de aanvragen door te geven aan de "logic" laag, die op zijn beurt de logica afhandelt en de data op de correcte manier ophaalt vanuit de DAO's. Dit is daarom een goede keuze geweeest.

Het hashing algoritme "bcrypt" is weliswaar veiliger en sneller, echter is deze ook lastiger te implementeren. De programmeur moet hiervoor zelf veel code schrijven en er is daarom ook meer ruimte voor fouten in de code. Er is daarom gekozen voor "the next best thing", en dat is op dit moment het "PBKDF2" algoritme.

3. Deployment diagram

In dit hoofdstuk wordt het deployment diagram besproken. Een deployment diagram biedt een overzicht van de componenten en verschillende onderdelen van de software. Ook laat het zien welke besturingssystemen en andere software er gebruikt wordt in de verschillende componenten.

3.1 Diagram



De applicatiestructuur bestaat uit 3 duidelijk herkenbare onderdelen. Er is een client server, dit is de spotitube client applicatie. Deze verbindt met de spotitube-api middels het http-protocol. De spotitube-api verbindt op den duur met de database, welke een MySQL omgeving is. Hiervandaan haalt de applicatie de data waar de client om heen gevraagd.

3.2 Requirements

Requirement	Onderbouwing
Correcte API, geïmplementeerd met JAX-RS	Er is gebruik gemaakt van een REST implementatie met de "service" klassen. Deze bepalen de endpoints er verzorgen correcte afhandeling hiervan.
Correcte JSON, op basis van Data Transfer Objects	Er wordt voor gezorgd dat de software correcte JSON-data terugstuurt, door middel van de "domain" klassen, deze fungeren als de DTO's.
JDBC implementatie	De software praat met de database via een JDBC-implementatie, dit is de driver die geimplementeerd is om met de MySQL database te praten.
TomEE Plus	De software draait in de TomEE Plus omgeving, om zo containerized te werken.

3.3 Design patterns

De client en server (api) zijn gescheiden gehouden in deze implementatie van Spotitube. Dit betekent dat het heel eenvoudig is om de api te hergebruiken, bijvoorbeeld bij het bouwen van een native app voor dit platform.

3.4 Alternatieve oplossingen

Er is een alternatieve mogelijkheid om een MVC-applicatie te implementeren van spotitube. Dat betekent dat alles, de FrontEnd, BackEnd, en Database in een geheel zit. Dit maakt het beheren van de codebase gemakkelijker en er hoeft geen gebruik gemaakt te worden van REST.

3.5 Onderbouwing gemaakte keuzes

Het MVC-model toepassen in spotitube is niet de goede oplossing, ook al lijkt dit verleidelijk. Er is dan later geen ruimte voor doorontwikkeling – zoals eerdergenoemd – met bijvoorbeeld een native app implementatie.