Programmeer opdracht EAI: Spotitube

1. Inleiding

Deze opdracht bestaat uit verschillende delen:

- Opleverdocument met verantwoording, package- en deployment-diagram.
- Realisatie van de functionaliteit

Je levert deze opdracht individueel in en demonstreert zelf de werking aan de docent. Je schrijft alle code zelf, maar mag je met collega-studenten overleggen en/of elkaar hulp bieden. Je bent en blijft zelf verantwoordelijk voor wat je inlevert en demonstreert.

2. Casus

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.

De applicatie maakt gebruik van de volgende APIs en frameworks:

- JAX-RS v2.0 (REST, JSON)
- CDI (Context & Dependency injection)
- JDBC API

De applicatie moet gedeployed kunnen worden op Apache TomEE Plus.
De front-end en back-end moeten Restful (JSON over HTTP) kunnen communiceren volgens de REST API specificatie.

2.1 Realiseer de applicatie op basis van deze casusbeschrijving

Realiseer de Spotitube-applicatie waarbij je gebruik maakt van de volgende layers:

- **Data access layer**: Implementeer het Data Mapper pattern en Separated Interface door middel van JDBC en een SQL database.
- **Domain layer** of **Service Layer** pattern
- **Remote Facade** pattern doormiddel van REST-services.

2.2 Interactie tussen lagen
Verlaag de afhankelijkheid tussen de lagen door Dependency Injection toe te passen
Toon aan dat de afhankelijkheid verlaagd is door meerdere unittests.

De Presentation layer is gegeven middels de client-applicatie en hoef je niet zelf te realiseren maar je moet deze wel gebruiken om jouw back-end aan te spreken.

2.3 Code coverage
Zorg dat minimaal 80% van de code via Unittests wordt getest.

3. Maak een opleverdocument
Maak een opleverdocument, volgens de ICA Stijlkaart, met daarin minimaal de volgende onderwerpen:

3.1 Een package diagram
Een UML-packagediagram dat de packagestructuur van de broncode illustreert, inclusief een toelichting waarin je de volgende punten raakt:

- Welke requirement je ermee raakt en/of oplost
- Welke design patterns of andere ontwerpprincipes je hebt toegepast
- Wat een alternatieve oplossing zou zijn geweest
- Waarom je voor de huidige oplossing hebt gekozen

3.2 Een Deployment diagram
Een UML deploymentdiagram dat de runtimeomgeving, protocollen, componenten en overige infrastructuur zoals servers, besturingssystemen en databases laat zien, inclusief een toelichting waarin je de volgende punten raakt:

- Welke requirement je ermee raakt en/of oplost
- Welke design patterns of andere ontwerpprincipes je hebt toegepast
- Wat een alternatieve oplossing zou zijn geweest
- Waarom je voor de huidige oplossing hebt gekozen

3.3 Ontwerp-/Craftsmanship -keuzes
Een verzameling ontwerpkeuzes, eventueel geïllustreerd met UML-diagrammen naar keuze, waarmee je aantoont een applicatie te hebben geschreven met Clean Code.