|  |
| --- |
| Opleverdocument\_  DEA |

Jannis Geerts

27 oktober 2023

INHOUDSOPGAVE

[1 package diagram 4](#_Toc149335094)

[2 Deployment diagram 5](#_Toc149335095)

[3 Ontwerp-/Craftsmanship -keuzes 6](#_Toc149335096)

[4 BronneLijst 7](#_Toc149335097)

INLEIDING

In dit oplever document komen de volgende dingen in voor:

Een package diagram dat de packagestructuur van mijn broncode illustreert met uitleg eronder over welke requirement ik ermee raak en of oplos, welke design patterns of andere ontwerpprincipes ik heb toegepast, wat een alternatieve oplossing zou zijn geweest en waarom ik voor de huidige oplossing heb gekozen.

Een Deployment diagram dat de runtimeomgeving, protocollen, componenten en overige infrastructuur zoals servers, besturingssystemen en databases laat zien. Ook zit er een toelicht bij waarin ik uitleg welke requirements ik ermee raak en of oplos, welke design patterns of andere ontwerpprincipes ik heb toegepast, wat een alternatieve oplossing zou zijn geweest en waarom ik voor huidige oplossing heb gekozen.

En tot slot een Ontwerp-/Craftsmanship -keuzes waarin een verzameling van ontwerpkeuzes in heb staan met een omschrijving waarom ik die keuzes heb gemaakt.

# package diagram

Afbeelding met diagram, Rechthoek, schermopname, Plan

Automatisch gegenereerde beschrijving

Ik heb mijn mappen structuur opgedeeld in drie belangrijke mappen de presentation laag, application laag en de datasource laag. Ik heb hiervoor gekozen, omdat ik dan per laag dezelfde functionaliteit bij elkaar had. Zoals bijvoorbeeld in de presentation layer zitten alle resources zoals LoginResource, PlaylistResource en TrackResource en in de andere lagen zaten er weer klassen met dezelfde functionaliteit. Dit vond ik logisch en daarom koos ik ervoor. Ook zorgt deze mappen structuur door samen te werken met jakarta dependency injecties voor een hoge cohesie en een lage koppeling. Een alternatieve oplossing zou zijn geweest dat ik de mappen zou indelen op de login, playlist en track. Dit leek mij onhandig, omdat je dan allemaal verschillende klasse met verschillende functionaliteit in één map hebt. Hierdoor zou je dus veel moeten importeren wat weer slecht is voor de cohesie en de koppeling.

# Deployment diagram

Afbeelding met tekst, diagram, Rechthoek, schermopname

Automatisch gegenereerde beschrijving

Met het artifact zoals mockito kan ik de 80 % unitests behalen en de uni test requiremnt halen. Met het artifact jersey-client kan ik dependency injections doen waardoor ik de cohesie hoog kan houden en de koppeling laag. Met wildfly kan ik alle library’s en de code runnen. Met de library mssql-jdbc kan ik de connectie maken met de database en daar dingen uithalen, updaten en verwijderen. Hierdoor haal ik de database requirements. De github webserver comuniceerd met wildfly via Json/HTTP. Hierdoor kan ik de endpoints werkend maken en die requirements behalen. Ik heb voor deze huidige oplossing gekozen, omdat wij tijdens vorige lessen hier uitleg over hadden gekregen en oefening mee hadden gemaakt. Hierdoor had ik al wat meer ervaring in deze oplossingen en was het handiger om die te gebruiken. Een alternatieve oplossing zou zijn om andere library’s te gebruiken voor de database of een andere library dan jakartaee.

# Ontwerp-/Craftsmanship -keuzes

Ik heb ervoor gekozen om de meeste dingen door de database te laten doen. Dit leek mij handig omdat ik goed kan omgaan met databases en de database alle informatie al had die nodig was. Hierdoor kon ik alle informatie die nodig was direct uit de database halen en die meteen te stoppen in de bijbehorende dto. Hierdoor hoefde ik niet hele ingewikkelde logica maken in de serviceses klassen.

Ik heb ervoor gekozen om geen exception te gooien wanneer de database niks returned bij haalPlaylistsOp() aangezien de playlist tabel in de database leeg kan zijn. Dit zorgde voor het probleem dat wanneer je de laatste playlist zou verwijderen dat je de exception response meteen kreeg en de pagina eerst moest refreshen voordat de laatste playlist zou verdwijnen.

Toen ik haalLoginGevenOp() maakte werkte ik eerst met hardcoded values. Hierdoor maakte ik eerst een list van hardcoded login classes en filterde daar de benodigde klasse uit. Mijn loginDoa() heb ik daarom heen geprogrammeerd. Maar nu dat ik een database heb zie ik dat dat helemaal niet meer hoeft. Ik zou namelijk alleen maar 1 login row uit de database krijgen van de gebruiker waarvan ik meteen een login class had kunnen maken en die returnen. Maar door een te kort aan tijd heb ik het niet kunnen aanpassen.

Ik heb geen loginDTO klasse gemaakt, omdat ik dacht dat het een nuttelose dto zou zijn geweest. Want het zou op dat moment helemaal niks doen. Maar achteraf was het best handig voor de unitest getItemsVanLoginDao() in de LoginServiceTest aangezien ik nu hardcode met getal 0 hem uit de lijst moet halen.

Ik heb ervoor gekozen om de functie getPlaylistItems() die in PlaylistDao zit te laten beslissen of isOwner true or false is. Ik heb dit gedaan omdat de functie door de query alle benodigde informatie al had zoals de token van de gebruiker en de token van de playlist. Hierdoor kon ik de tokens vergelijken met elkaar en kijken of de playlist van de user was. Als dat zo was maakte ik isOwner true.

Ik heb ervoor gekozen om twee query’s te hebben in de functie addTrack() die in de TrackDao zit. Inplaats van één. Ik heb daarvoor gekozen, omdat ik er pas later achter kwam dat de isOfflineAvailable eigenlijk in de tabel playlistTrack hoorde. OfflineAvailable hoort in de tabel playlistTrack, want het moet per user verschillen of de track offline available is. Hoe ik het nu heb geschreven is dat als één persoon iets aanpast dat de rest dat ook krijgt. Ook had ik, maar één query nodig in de functie addTrack() in het geval dat isOfflineAvailable in playlistTrack zat. Door de tijdsnood en het feit dat als ik de tabel zou aanpassen ik overal in de code weer aanpassingen zou moeten maken heb ik het niet aangepast.

Ik kan in de class PlaylistResourceTest veel responses en object die terug wordt gestuurd niet testen, omdat ik er een paar functies void heb gemaakt. Toen ik erachter kwam had ik niet genoeg tijd meer om het optelossen.

# BronneLijst

Hanica-Dea. (z.d.). GitHub - HANICA-DEA/Spotitube: OOSE DEA - Spotitube Client. GitHub. <https://github.com/HANICA-DEA/spotitube>

Han. 2023/2024. I-OOSE DEA 2023/2024 S1. Onderwijsonline van

https://han.onderwijsonline.nl/elearning/content/4ymjX1DB

