|  |  |
| --- | --- |
|  | Opleverdocument Spotitube |
|  |  |
| 3/27/2020 |  |
|  | Julian Roddeman, Nijmegen  Studentnummer: 621655  Klas: OOSE-A-N  Course: OOSE-DEA  Docent: Meron Brouwer  Versie: 1.0 |

Inhoud

[1. Inleiding 2](#_Toc36203422)

[2. Klassen diagrammen 3](#_Toc36203423)

[3. Package diagram 6](#_Toc36203424)

[4. Deployment diagram 7](#_Toc36203425)

[5. Unit test cases 8](#_Toc36203426)

[6. eXTRA ONTWERPKEUZES 9](#_Toc36203427)

# Inleiding

Dit opleverdocument is tot stand gekomen door een opdracht van het Distributed Enterprise Applications. Een vak dat onderdeel is van de opleiding HBO-ICT op de instellingen HAN Arnhem-Nijmegen.

De opdracht is om de applicatie Spotitube te realiseren en te verantwoorden door middel van dit document. Spotitube is een Java EE applicatie waarbij een afspeellijsteigenaar kan inloggen om afspeellijsten, video’s en liedjes te beheren.

In dit opleverdocument worden diverse diagrammen voor de applicatie Spotitube getoond. Deze diagrammen geven weer hoe de gerealiseerde applicatie in elkaar zit en hoe deze ingezet kan worden. Daarnaast wordt de kwaliteit van het gerealiseerde product aangetoond door een tabel die de test dekking weergeeft. Tot slot wordt aangeduid voor welke ontwerpkeuzes ik heb gekozen tijdens de realisatie van Spotitube.

# Klassen diagrammen

Voordat ik begonnen ben met het realiseren van de applicatie, heb ik uiteraard nagedacht over het ontwerp. Tijdens het ontwerpen heb ik een domeinmodel gemaakt op basis van de opdrachtbeschrijving. Dit domeinmodel verder uitgewerkt in sequence diagrammen waar samen het onderstaande klassen diagram uit voort is gekomen.

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

Figuur 1 - Oorsporonkelijk klassen diagram

Wat opvalt in het diagram is dat een Track een abstracte klassen is. Ik heb hiervoor gekozen omdat zowel een video als een lied beide tracks zijn. Video’s en liedjes hebben beide attributen gemeen maar ook attributen die bij de ander niet voor komen (lied heeft een album en een video niet). In deze situatie zou in interface niet gepast zijn omdat deze geen attributen kan hebben.   
Echter ben ik er tijdens de realisatie van Spotitube achter gekomen dat het bovestaande klassen diagram voor een deel het beste op een andere manier geïmplementeerd kon worden. Het resultaat staat op de volgende bladzijde.

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

Figuur 2 - huidig klassen diagram

Als eerst heb ik de relatie tussen Spotitube en verwijderd. De rede hiervoor is dat de controller (afhankelijk van het domein) de klasse Eigenaar gebruikt om in te loggen, de ingelogde eigenaar te controleren en afspeellijsten te wijzigen/maken. Dit gaat dus niet via Spotitube en daarom is de relatie overbodig.   
Ten tweede is de relatie tussen Spotitube en Afspeellijst weg. Dit is zo omdat Spotitube alleen maar een dependency moet hebben met Afspeellijst om de tracks in de afspeellijst, bij het openen van een overzicht, op te halen. Het ophalen van tracks gebeurt door een afhankelijkheid met een DAO uit de data source laag. Dit is alleen mogelijk wanneer afspeellijst zelf ook een afhankelijkheid is (door CDI).  
Ten derde is de relatie weg tussen Eigenaar en Afspeellijst. De rede hiervoor is dat eigenaar de dependency op de DAO’s in de data source laag gebruikt om afspeellijsten toe te voegen en te wijzigen. Daar is de relatie niet voor nodig.

Tot slot heb ik in de domein laag data-mappers toegevoegd die ervoor zorgen dat de resultset uit een DAO omgezet wordt daar objecten van klassen uit het domeinmodel.

# Package diagram

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijvingHieronder zie je het package diagram. Het package diagram geeft de structuur en afhankelijkheden van de applicatie weer. In mijn applicatie heb ik drie lagen: Controller, Domain en Data Source. De dragen lagen zijn in mijn applicatie de drie hoofd packages.   
De controller laag die communiceert met de gebruiker door twee klassen, eigenaarController en afspeellijstController.   
Ik had er voor kunnen kiezen om de afspeellijstController op te splitsen in een afspeellijst- en trackcontroller. Dit heb ik niet gedaan omdat de methodes die tracks ophalen/verwijderen dit doen voor of van een afspeellijst. Dus er moet een afspeellijstId worden meegegeven, waardoor dit onder afspeellijst valt.   
De Data Transfer Objects (DTO) zijn objecten die mee worden gegeven door de server. Omdat ik mijn applicatie in het Nederlands heb geschreven (zie [klassendiagrammen](#_Klassen_diagrammen)), de data van de server in het Engels is en de attributen daartussen verschillen moeten de DTO’s naar domeinobjecten worden vertaald. Dit wordt gedaan in de datamappers package binnen de controller laag.   
Hoe de domein laag in elkaar zit is uitgelegd in het [vorige hoofdstuk](#_Klassen_diagrammen).   
De domein laag is afhankelijk van de DAO’s package in de Data Source laag. De rede hiervoor is dat de klassen in de domein laag data uit de database die de DAO’s leveren. De DAO’s package is afhankelijk van connectie om de connectie met de database tot stand te brengen. Ik had de connectie ook direct in elke klasse kunnen stoppen maar ik heb hiervoor gekozen om code duplicatie te voorkomen.

Figuur 3 - Package diagram

Binnen data source heb ik het Table Data Gateway Pattern gebruikt. De Gateways heb ik Data Acces Objects (DAO) genoemd en bevat alle SQL voor het opvragen, toevoegen, verwijderen en te wijzigen van data voor tabellen. Ik heb hiervoor gekozen omdat de structuur van de database in zijn geheel overeenkomt met de structuur van mijn domein. Voor elke tabel is er dus een DAO (Gateway) en voor elke DAO een klassen in het domeinmodel. Ik had deze situatie ook kunnen kiezen voor het Data Mapper Pattern. Dit heb ik niet gedaan omdat ik naast standaard select, inserts, updates en deletes ook specifiekere query’s nodig heb, zoals het ophalen van het hoogste afspeellijstId. In het Data Gateway Pattern is dit eenvoudig te doen door een methode getMaxId() aan te maken in de DAO.

# Deployment diagram

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijvingIk heb ervoor gekozen om de applicatie en de database op een aparte productie server te laten draaien. Dit heeft namelijk voordelen:  
Als eerst dat er altijd een connectie met een andere database op een andere server kan worden gelegd. De applicatie is er niet aan verbonden.   
Ook kan er meer verkeer worden afgehandeld doordat er twee servers zijn. Als Spotitube een applicatie wordt met veel gebruikers is dit van belang.   
Tot slot zou een andere applicatie de database op de aparte server ook kunnen gebruiken zonder dat er hij iets te maken heeft met de Spotitube applicatie.

Figuur 4 - deployment diagram

# Unit test cases

Alle methodes binnen de Spotitube heb ik getest. Hieronder is een tabel te vinden met de Class, Mehtod en Line Coverage.

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijving

Figuur 5 - overall coverage summary

# eXTRA ONTWERPKEUZES