SRS Spotitube

1. Introduction

1.1. Overall Description

Spotify en Youtube hebben de handen ineen geslagen 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 eenvoudige webapplicatie alvorens over te gaan tot de ontwikkeling van de app.

1.2. User Classes and Characteristics

Voor het systeem is alleen de actor Abonnee van belang.

1.3. Operating Environment

De applicatie moet platformonafhankelijk zijn en ontwikkeld worden met Java versie 8 en draaien in TomEE. Er zijn geen koppelingen met bestaande applicaties.

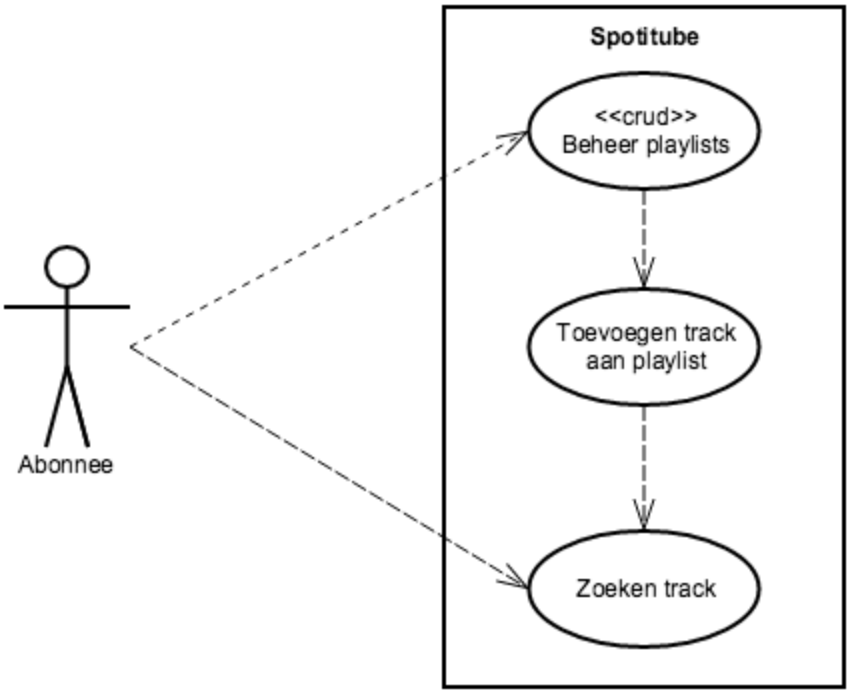
1.4. Design and Implementation Constraints

- De applicatie moet naast MySQL5.1 ook minimaal 1 andere relationele database kunnen ondersteunen.
- De applicatie maakt gebruik van de volgende APIs en frameworks:
    - JSP
    - Servlet v3.0
    - JAX-RS v2.0 (REST, JSON)
    - CDI (Context & Dependency injection)
    - JDBC
    - JDBC driver v5.1.34 voor MySQL
- De applicatie wordt ontwikkeld met Java versie 8 en moet draaien in Apache TomEE.
- De front-end en back-end moeten Restful (JSON over HTTP) kunnen communiceren.

Voor meer details zie de paragraaf *Non-functional Requirements*.

1.5. Product Functions

In het onderstaande figuur staan de architectureel significante use cases, ofwel de use cases die als functionaliteit gekozen zijn om aan te tonen dat aan de niet-functionele eisen voldaan kan worden.

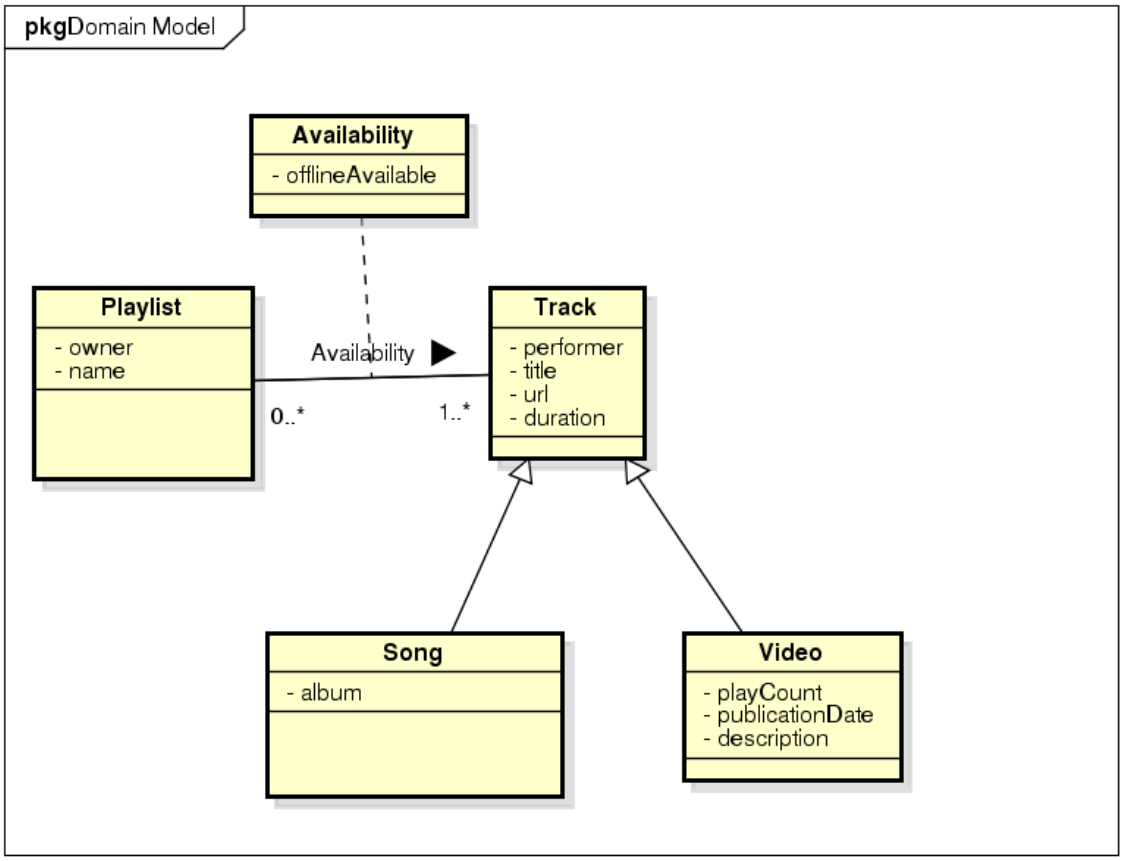


Een abonnee start de applicatie en de applicatie toont de playlists waar de abonnee eigenaar van is. Als de abonnee besluit om een playlist toe te voegen vraagt het systeem om een naam voor de playlist en om minimaal 1 track (dat kan een video of song zijn) toe te voegen aan de playlist. Als de abonnee besluit om een playlist te wijzigen dan kan hij de naam wijzigen en 1 of meer tracks toevoegen. Om een track toe te kunnen voegen is het noodzakelijk om deze eerst te zoeken.

2. Domain Model

Een playlist heeft een naam en een eigenaar. Er kunnen 2 soorten tracks in een playlist worden opgeslagen namelijk liedjes (song) en filmpjes (video). Elke track in een playlist kan offline beschikbaar zijn of niet.

2.1.1.1.1.1. spotitube DM



powered by Astah

3. Use-case Descriptions

In deze sectie wordt elke use case in detail beschreven in het fully dressed formaat.

3.1. Beheer playlists

3.1.1. Fully-dressed use case description

Primary actor	Abonnee	
Brief Description	Een abonnee start de applicatie en de applicatie toont alle afspeellijsten voor de abonnee en de totale lengte van alle afspeellijsten.	
Preconditions	De abonnee is ingelogd.	
Postconditions (Success Guarantee)	De abonnee heeft een overzicht van zijn afspeellijsten en de totale lengte van de afspeellijsten.	
Main Succes Scenario	**Actor Action**	**System Responsibility**

	1. De abonnee start de applicatie	2. Het systeem toont de afspeellijsten en de totale lengte van de afspeellijsten. De abonnee kan vanaf hier een van de volgende use cases starten: • Toevoegen track aan playlist • Aanmaken playlist • Verwijderen playlist • Wijzigen playlist • Details bekijken playlist • Zoeken track
Alternative Scenario		2a. Er zijn nog geen afspeellijsten, het systeem toont dus geen afspeellijsten en de totale lengte van de afspeellijsten wordt niet getoond. De abonnee kan vanaf hier een van de volgende use cases starten: • Aanmaken playlist • Zoeken track

3.2. Aanmaken playlist

3.2.1. Fully-dressed use case description

Primary actor	Abonnee	
Brief Description	Een abonnee geeft maakt een playlist aan en geeft deze playlist een verplichte naam.	
Preconditions	De abonnee is ingelogd.	
Postconditions (Success Guarantee)	De abonnee heeft een nieuwe playlist die hij zelf een naam heeft gegeven en waarvan hij zelf de eigenaar is. • Er is een nieuwe instantie pl van Playlist • Het attribuut pl.owner is gelijk aan de naam van de huidige ingelogde gebruiker • Het attribuut pl.naam is gelijk aan de ingegeven playlist-naam	
Main Succes Scenario	**Actor Action**	**System Responsibility**
	1. De abonnee geeft aan een playlist te willen maken.	2. Het systeem geeft de mogelijkheid om een naam aan de playlist te geven.
	3. De abonnee geeft de playlist een naam en geeft aan dat de playlist opgeslagen mag worden.	4. Het systeem slaat de playlist op en maakt de huidige ingelogde gebruiker de eigenaar van de playlist. De use case "Beheer playlists" start.
Alternative Scenario	3a. De abonnee vergeet de playlist een naam te geven en geeft aan dat de playlist opgeslagen mag worden.	4a. Het systeem toont een foutmelding dat de playlist een naam moet hebben. De use case gaat verder bij stap 2.

3.3. Wijzigen playlist

3.3.1. Fully-dressed use case description

Primary actor	Abonnee
Brief Description	Een abonnee wijzigt een playlist en geeft deze playlist een verplichte naam.
Preconditions	De abonnee is ingelogd en heeft minimaal 1 Playlist (pl) waarvan hij eigenaar is.
Postconditions (Success Guarantee)	De abonnee heeft een bestaande playlist die hij zelf een andere naam heeft gegeven en waarvan hij zelf de eigenaar blijft. • Het attribuut pl.owner is gelijk gebleven aan de naam van de huidige ingelogde gebruiker • Het attribuut pl.naam is gelijk aan de ingegeven playlist-naam

Main Succes Scenario	Actor Action	System Responsibility
	1. De abonnee geeft aan een playlist te willen wijzigen.	2. Het systeem geeft de mogelijkheid om een naam aan de playlist te geven.
	3. De abonnee geeft de playlist een naam en geeft aan dat de playlist opgeslagen mag worden.	4. Het systeem slaat de playlist op. De use case "Beheer playlists" start.
Alternative Scenario	3a. De abonnee vergeet de playlist een naam te geven en geeft aan dat de playlist opgeslagen mag worden.	4a. Het systeem toont een foutmelding dat de playlist een naam moet hebben. De use case gaat verder bij stap 2.

3.4. Verwijderen playlist

3.4.1. Fully-dressed use case description

Primary actor	Abonnee	
Brief Description	Een abonnee verwijdert een van zijn playlists.	
Preconditions	De abonnee is ingelogd en heeft minimaal 1 playlist.	
Postconditions (Success Guarantee)	De abonnee heeft een playlist minder. • Er is een instantie pl van Playlist verwijderd • Alle eventuele associaties tussen pl en instanties van Track zijn verwijderd evenals de instanties van associatieklassen (Availability) die aan deze associaties hingen.	
Main Succes Scenario	**Actor Action**	**System Responsibility**
	1. De abonnee geeft aan een playlist te willen verwijderen.	2. Het systeem verwijdert de aangegeven playlist. De use case "Beheer playlists" start.

3.5. Toevoegen track aan playlist

3.5.1. Fully-dressed use case description

Primary actor	Abonnee	
Brief Description	Een abonnee wil een van zijn playlists uitbreiden met 1 of meerdere Tracks.	
Preconditions	De abonnee is ingelogd en heeft minimaal 1 playlist. • Er is een instantie pl van Playlist met een naam pl.naam waarvan de eigenaar overeenkomst met pl.owner • Er bestaan instanties gecreëerd van Track, genaamd t1 t/m tn	
Postconditions (Success Guarantee)	De playlist is uitgebreid met 1 of meerdere Tracks. • Er is een associatie tussen pl en t1 t/m tn met een assocatieklasse Availability met een waarde offlineAvailable die gelijk is aan de waarde die de gebruiker heeft ingegeven.	
Main Succes Scenario	**Actor Action**	**System Responsibility**
	1. De abonnee geeft aan dat hij zijn playlist wil uitbreiden met Tracks.	2. De use case "Zoeken Track" start.
	3. De abonnee selecteert een van de gevonden Tracks.	4. Het systeem vraagt de gebruiker om aan te geven of de Track online of offline in de playlist moet worden geplaatst.
	5. De abonnee kiest een waarde voor offline of online en geeft aan dat de Track mag worden toegevoegd.	6. Het systeem vraagt de gebruiker of hij nog een Track wil toevoegen of dat hij klaar is.
	7. De abonnee geeft aan dat hij klaar is met het uitbreiden van de playlist.	8. De use case "Beheer Playlist" start.

Alternative Scenario	5a. De abonnee geeft niet actief een waarde in. De Track wordt opgeslagen met de standaard waarde voor availability (*online*). De use case gaat verder bij stap 6.	
	7a. De abonnee geeft aan nog een Track te willen toevoegen. De use case gaat verder bij stap 2.	

3.6. Details bekijken playlist

3.6.1. Fully-dressed use case description

Primary actor	Abonnee	
Brief Description	Een abonnee wil de details van een zijn playlists bekijken.	
Preconditions	De abonnee is ingelogd en heeft minimaal 1 playlist. • Er is een instantie pl van Playlist met een naam pl.naam waarvan de eigenaar overeenkomst met pl.owner	
Postconditions (Success Guarantee)	Het systeem heeft van de betreffende playlist de volgende gegevens getoond: • Titel, uitvoerende en afspeelduur • Album (alleen als het stream uit Spotify betreft) • Aantal keren afgespeeld, publicatiedatum en beschrijving (alleen als het een video uit YouTube betreft)	
Main Succes Scenario	**Actor Action**	**System Responsibility**
	1. De abonnee geeft aan dat hij details van zijn playlist wil bekijken.	2. Het systeem toont de naam van de playlist met daarin alle tracks en de details van de tracks: • Titel, uitvoerende en afspeelduur • Album (alleen als het stream uit Spotify betreft) • Aantal keren afgespeeld, publicatiedatum en beschrijving (alleen als het een video uit YouTube betreft)

4. Non-functional Requirements

4.1. Testability

Code	Description	Explanation
T1	De communicatielaag en databasetoegang moeten los van elkaar testbaar zijn.	Zie IM2. Toon dit aan middels minimaal een unit-test.

4.2. Technical Constraints

Code	Description	Explanation
TC1	De applicatie moet draaien in TomEE.	Een andere applicatieserver is een mogelijk onderzoeksonderwerp.

4.3. Interoperability

Code	Description	Explanation

IO1	De front-end en back-end van de applicatie moeten samen in 1 container kunnen draaien zodat ze lokaal met elkaar kunnen communiceren.	JSPs en Servlets moeten services en/of DAO's kunnen aanroepen via directe methodeaanroepen.
IO2	De front-end en back-end van de applicatie moeten elk in een eigen container kunnen draaien zodat ze via een protocol met elkaar kunnen communiceren.	De servicelaag is Restful waardoor een nieuw JavaScript of App-frontend deze zou kunnen aanroepen. • Je hoeft dus geen JSP/Servlet oplossing via remoting met de servicelaag te koppelen, dat mag lokaal. • Je hoeft geen REST-communicatie tussen front- en back-end mogelijk te maken, dit is een optie voor de onderzoeksopdracht.

4.4. Maintainability

Code	Description	Explanation
M1	Views (JSP-bestanden) bevatten geen Java-code, alleen JSP of JSTL markup.	Scheiding van view, controller en model.
M2	PageControllers (Servlets) bevatten geen markup, alleen Java-code.	Scheiding van view, controller en model.

4.5. Interfaces

Code	Description	Explanation
IN1	De front-end en back-end moeten Restful (JSON over HTTP) kunnen communiceren.	De servicelaag is Restful waardoor een nieuw JavaScript of App-frontend deze zou kunnen aanroepen. • Je hoeft dus geen JSP/Servlet oplossing via remoting met de servicelaag te koppelen, dat mag lokaal. • Je hoeft geen REST-communicatie tussen front- en back-end mogelijk te maken, dit is een optie voor de onderzoeksopdracht.

4.6. Implementation

Code	Description	Explanation
IM1	De applicatie moet naast MySQL5.1 ook minimaal 1 andere relationele database kunnen ondersteunen.	Implementeer minimaal MySQL en zet je code zo op dat je snel kunt wisselen naar een andere relationele database.
IM2	De applicatie maakt gebruik van de volgende APIs en frameworks: • JSP • Servlet v3.0 • JAX-RS v2.0 (REST, JSON) • CDI (Context & Dependency injection) • JDBC • JDBC driver v5.1.34 voor MySQL	Architectuurlagen zijn van elkaar gekoppeld op basis van interfaces. Creatie van classes (m.u.v. domein-klassen) uit deze architectuurlagen vindt zoveel mogelijk plaats op basis van het DIP-principe.

4.7. Design

Code	Description	Explanation
D1	Bij het wisselen van database moet de code niet opnieuw gecompileerd hoeven te worden.	Databaseconfiguratie opslaan in een properties of XML bestand.
D2	De applicatie moet eenvoudig kunnen wisselen van een relationele (RDBMS) opslag naar een andere opslag zoals NoSQL of flat files.	Pas het Table Data Gateway pattern toe (ofwel Data Acces Object, DAO)

D3	De broncode is georganiseerd op functionaliteit, niet op technische rol.	Packages zijn dus primair ingedeeld op functie dan pas op architectuurlaag.

5. User interface sketches

In onderstaand diagram wordt het hoofdscherm globaal weergegeven samen met een later te ontwikkelen app. Beiden maken gebruik van een REST-ful API die door de backend geboden wordt.