<Project Name>

Analysis and Design Document

**MusiCast Application**

Student: Neagu Lorena

**Group:30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

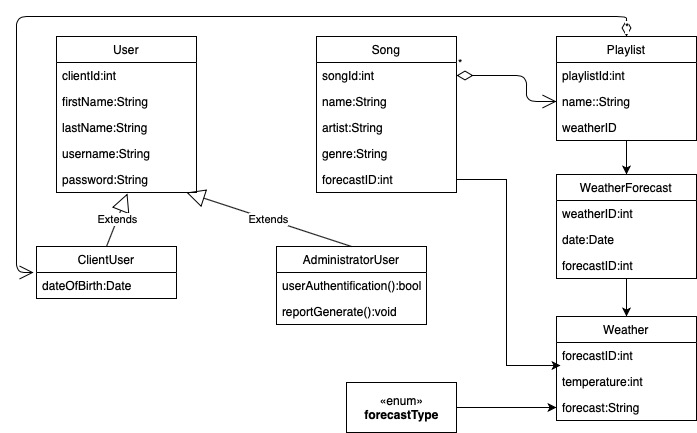
# Project Specification

*The MusiCast app is designed to be a playlist generator music application, with Java as it’s main programming language. The user will have an account which he’ll be able to create based on it’s personal data, in which he’ll log in once he’ll enter the app for a personalized user experience. He will be able to listen to music, check the weather forecast and have personalized playlists based on the outside weather that day or time or create a playlist on their own. The user can perform CRUD operations on the songs, playlists and also generate reports.*

# Elaboration – Iteration 1.1

# Domain Model

*The Domain model is a structured visual representation of the connected concepts of a domain that incorporates both behavior and data. There is a user class that contains relevant information about the user, such as name, username and password. Then, the application should also contain classes for the songs, a playlist which contains multiple songs and can be owned by users. Songs are associated to weather forecasts, for example if the song is slower it can be associated to a rainy day. Depending on the weather that day, a customized playlist will be generated.*



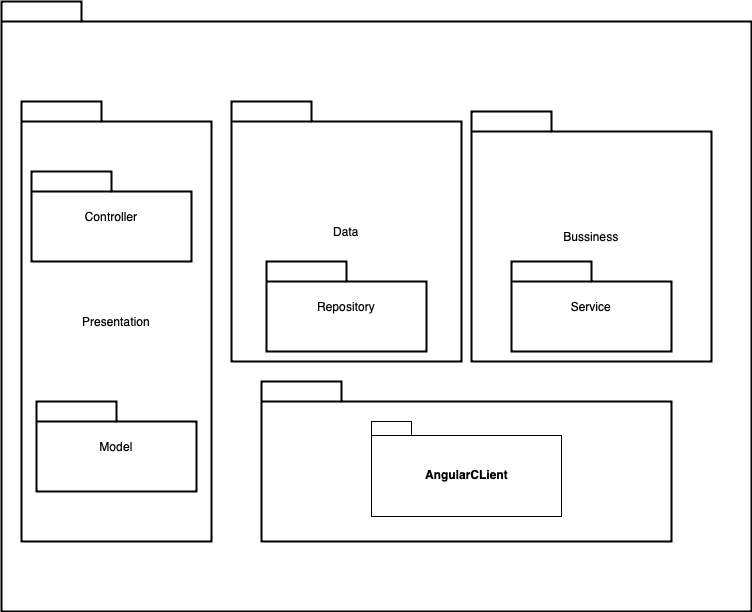
# Architectural Design

## Conceptual Architecture

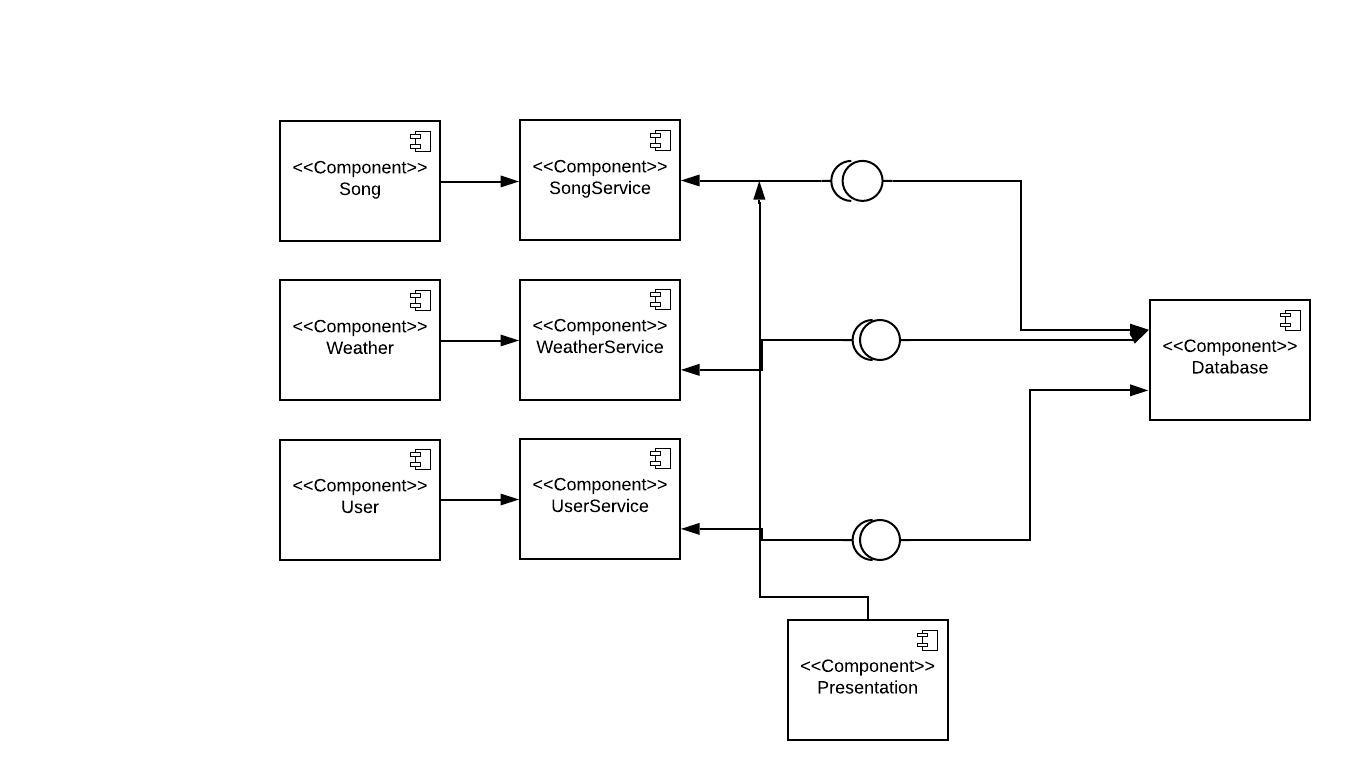
*The application is best fit to follow a Layered Architecture. This approach is probably a good one because the application is mainly built around the database. A good architectural style would be Client-Server because the client will request data from a server, and the server will handle all services and data operations.*

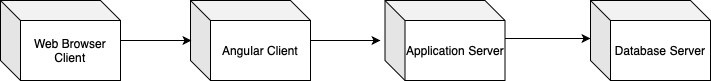
*The code is arranged so the data enters the top layer and works its way down each layer until it reaches the bottom, which is a database. Along the way, each layer has a specific task, like checking the data for consistency or reformatting the values to keep them consistent. The pattern is a 3-tier architecture which consists of 3 layers: The Presentation layer is where users interact with a web application via requests and responses, the Application layer processes requests from users and their responses, and the Data layer provides persistence and retrieval services for the database.*

## Package Design



## Component and Deployment Diagram

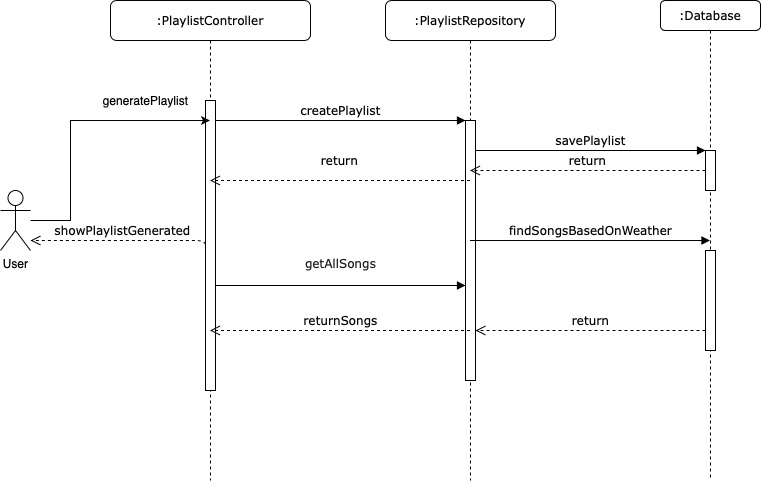


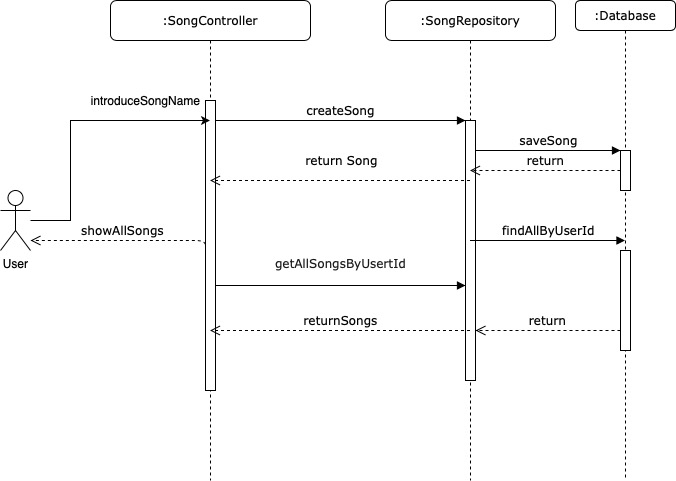


# Elaboration – Iteration 1.2

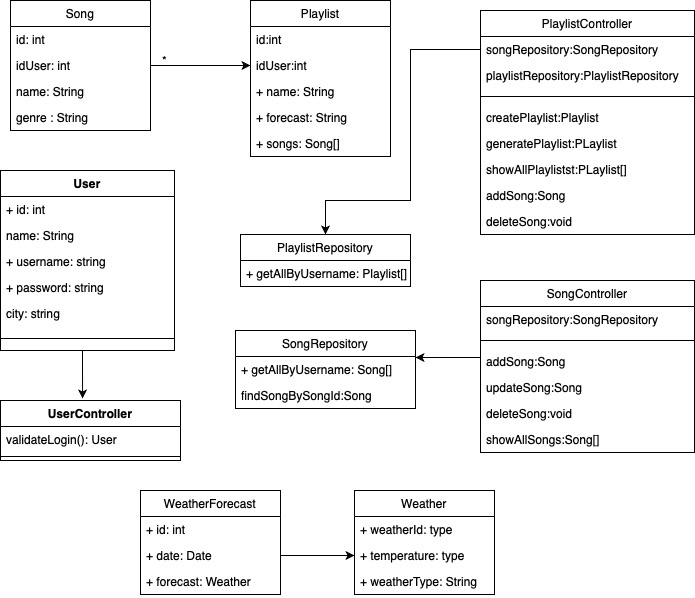
# Design Model

## Dynamic Behavior

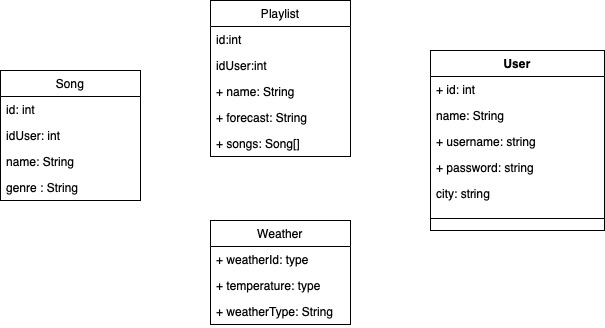




## Class Design



# Data Model



# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

*The integration testing units are built based on the use-case diagram. The test cases, based on the use-cases, will show the correct functioning of the system.*

*Test cases:*

* *Login – the users should be able to login successfully using their username and password; if the users introduces incorrect login credentials, they will be notified*
* *Register – new regular users can register easily using their data; if the user is created successfully, they will be added to the database*
* *Logout – the users are able to log out if they are already logged in*
* *Edit song/playlist – the administrator can edit any song/playlist in the database; if the update is realized correctly, the database will also be updated*
* *Delete song/playlist – administrator can delete any song or playlist from the database*
* *Browse songs– a regular user can see the list of all songs, and for each song it/s information like artist, genre*
* *Create playlist - the user can create a personalized playlist based on the outside weather*
* *Generate report – the administrator will be able to generate reports of the restaurants and their items in .pdf format or in .txt format; a new window will appear for selecting the path for the new file to be created; the report will be created in the specified location*

# Future improvements

*The application could be improved in many ways, one of them would be that each user could have a profile which can be edited and seen by all the other users. Each user could also delete their account. The administrator could also perform CRUD operations on users. Another improvement is that more and more songs can be added to the database and maybe some new functionalities added like users can review songs or post their playlists such that other users can view them.*

# Bibliography