|  |
| --- |
| **CHADMusic Free music and karaoke app** |

**Dragos-Daniel Bonaparte, 315261**

**Dan-Sebastian Ceapa, 315162**

**Chiril Luncasu, 315171**

**A picture containing text, person, white, suit

Description automatically generatedA picture containing person, person, dark, male

Description automatically generatedMatas Armonaitis, 315263**

**A person wearing sunglasses

Description automatically generated with medium confidenceA picture containing text, person

Description automatically generated**

**Supervisor:**

**Steffen Andersen (SVA), Mona Andersen (MWA),**

**Henrik Kronborg Pedersen (HEKP)**

A picture containing shape

Description automatically generated**Text

Description automatically generated with low confidence**

**Number of characters with no spaces: 22884**

**Software Technology Engineering 2nd Semester**

**2.06.2022**

**Table of content**

[Abstract iii](#_Toc105062344)

[1 Introduction 1](#_Toc105062345)

[2 Analysis 3](#_Toc105062346)

[2.1 Requirements 3](#_Toc105062347)

[Functional Requirements 3](#_Toc105062348)

[Non-Functional Requirements 4](#_Toc105062349)

[2.2 Given the product backlog the use case were formulated: 5](#_Toc105062350)

[2.2.1 Log In use case 7](#_Toc105062351)

[2.2.2 Sign up use case 8](#_Toc105062352)

[2.2.3 Play music use case 9](#_Toc105062353)

[2.2.4 See lyrics use case 10](#_Toc105062354)

[2.2.5 Search song use case 11](#_Toc105062355)

[3 Design 12](#_Toc105062356)

[4 Implementation 21](#_Toc105062357)

[5 Test 25](#_Toc105062358)

[6 Results and Discussion 27](#_Toc105062359)

[7 Conclusions 30](#_Toc105062360)

[8 Project future 31](#_Toc105062361)

[9 Sources of information 32](#_Toc105062362)

[10 Appendices 1](#_Toc105062363)

# Abstract

The aim of the project is to create probably the best app in terms of the existing competition when it comes to a music app, featuring a lot of features and functionality and at the same time remaining free of charge and ad-free for every user.

The team aimed for a product that will be able to deliver the existing trend features, and at the same time creating a easy to update and maintain code.

The project was constructed using Unified Process and SCRUM, applying the agile beliefs every time the team coded. As for the classes in the project that implement the back end of the program, the team followed SOLID to make everything efficient and simple.

As for the main design used by the team is MVVM which totally isolates the back end from the front-end. In terms of other used designs, the team also implemented the observer pattern, the factory method and are planning in a future release to implement the proxy pattern as well.

After 5 sprints, each of 3 days, the team managed to develop a product that not that it was satisfactory for the time, but it surpassed some of the team’s expectations when talking about the features.

# Introduction

Today a lot of music apps strive to achieve a better music quality (Anon., 2022b), but the customer is annoyed by the fact that the music apps do not focus on implementing functionalities. Prior to this the customer did some research and based on the feedback of various music app users, concluded that a lot of those apps do not meet expectations when it comes to functionalities like displaying the lyrics of a song, sound design, shuffle playlist, putting the app to sleep when not used, download music/listen offline or if the app includes those features, they come at a monthly subscription fee.

The current alternatives when it comes to music apps are Spotify and Tidal, both having great music quality but at some cost of course. Elaborating more on specific aspects of each of the two alternatives the customer deducted that even thou Spotify is free for you to listen to music but the number of ads in the app are completely ruining the experience, on top of that you are not able to download music offline. Speaking of worse, Tidal offers great functionalities and features but this app does not even allow you to play music with ads. Instead, it requests payment from the first installation, which is not very appealing at the beginning since you do not even know how the experience will be.

Those things alone will make the average teenager to start using the offline music app that their phone comes with from the factory. The market of those music apps are teenagers, thus making it difficult for them to embrace one app or another. Usually as a teenager you are not so happy about needing to make a monthly subscription just to listen to music.

A possible music app that can be used by teenagers is YouTube music, which is easy to use but has a downside, lots of ads and very little songs with lyrics.

In summary the current problem is lack of free access to music and features in those music apps.

It is reasonable to infer the need for a better and free alternative that will satisfy the needs of present youth.

This app will:

* Create a more personal experience having separate users with separate playlists.
* Offer an ad free music experience
* Allow the user to sing along with the song since there is lyrics for every well-known song.
* Access to an unlimited supply of songs

Delimitation-wise, the app will not be able to add songs to playlists in this version, does not provide a shuffle option yet and the app needs optimizing when it comes to sending songs, since it takes a relatively long time to send.

# Analysis

Understanding the problem thoroughly, the product owner was able to infer the product backlog the customer would desire.

## Requirements

## Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Priority** | **Hours estimate** | **Task** |
| 1 | Critical | 72 hours | As a user I want to be able to play and listen to music in a console environment. |
| 20 | Critical | 18 hours | As a user I want to be able to play and listen to music in a program window. |
| 2 | Critical | 36 hours | As a user I want to be able to see the lyrics of the song that I am listening to. |
| 3 | Critical | 36 hours | As a user I want to be able to pick a song from a list. |
| 4 | Critical | 18 hours | As a user I want to be able to pause or resume the song. |
| 21 | Critical | 32 hours | As a user I want to have multiple computers playing from the same list of songs. |
| 5 | Critical |  | As a user I want to be able to change the volume of the song. (Canceled by the product owner) |
| 22 | High | 6 hours | As a user I want to be able to run the program by having a shortcut on my desktop. |
| 6 | High | 10 hours | As a user I want to go to the next or previous song. |
| 7 | High | 10 hours | As a user I want that after a song finishes, another one will start playing automatically. |
| 8 | High | 36 hours | As a user I want to be able to have a liked songs playlist, for a more personal experience. |
| 9 | High | 10 hours | As a user I want to be able to search for a song. |
| 10 | High |  | As a user I want to have the songs categorized by genres. |
| 11 | Medium |  | As a user I want to be able to shuffle or repeat my playlist. |
| 12 | Medium |  | As a user I want to get song recommendation. |
| 23 | Low | 2 hours | As a user I want to know how many hours I spent listening to music. |
| 13 | Low |  | As a user I want to be able to change font size, color and style of the lyrics. |
| 14 | Low |  | As a user I want additional information about the song like the author, length, and year. |
| 15 | Low | 36 hours | As a user I want to have a separate account to not merge my liked songs with another user. |
| 16 | Low | 2 hours | As a user I want to be able to toggle between showing lyrics and not showing lyrics. |

## Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | Non-Functional |  | As a user I want to see a playlist with the most listened songs. |
| 18 | Non-Functional |  | As a user I want to see my activity, for example how many hours I have listened today. |
| 19 | Non-Functional |  | As a user I want the app to go to sleep after some inactivity. |

## Given the product backlog the use case were formulated:

**Diagram

Description automatically generated**

What follows is a table illustrating the relationship between the requirements and the use cases. The numbers in bold represent critical requirements.

|  |  |
| --- | --- |
| Use cases | Covered requirements |
| Sign Up | **15** |
| Log In | 15 |
| Play music | **1,20,3,4,6,7** |
| See lyrics | 2,16 |
| Search songs | **9** |
| No use case in particular | **21,22** |

The two requirements were met but not by any use case in particular, because:

* Requirement number 21, states that we should be able to connect to a server so that maybe the customer would like to connect his computer and laptop at the same time.
* Requirement number 22, states that there should be runnable jar files on the desktop for easy execution of the program rather than running it in intelij.

### Log In use case

When a user launches the program, he has to login to the system to be able to use the whole program. To login user needs to fill in username and password text fields and press login button. Then the system checks if the user exists in the database and opens the main window of the program, if the user is not in the database, he will be asked to put in other credentials.

|  |  |
| --- | --- |
| Use case section | Purpose |
| Use case name | Login |
| Scope | The Login system |
| Level | Logging into system |
| Primary actor | User |
| Stakeholders and interests | The user wants to login to the system to use the program |
| Pre-conditions | The user must have an existing account in the database |
| Success Guarantee | The user has an account that exists in the database |
| Main Success Scenario | 1. The user enters correct credentials in the needed text fields 2. The system finds the user in the database 3. The system opens the main window |
| Extensions | 1. If the user is not in the database, then the system won’t open the main window |
| Special requirements | User must be in the database |
| Technology and Data variations list | The user can login to the system |
| Frequency of occurrence | Always |
| Misc. |  |

### Sign up use case

When a user launches the program for the first time, he has to register to the system to be able to use the whole program. To registering user needs to fill in username, email and password text fields and press register button. Then the system checks if the user does not exists already in the database and opens the main window of the program, if the user is in the database, he will be asked to put in other credentials.

|  |  |
| --- | --- |
| Use case section | Purpose |
| Use case name | Sign Up |
| Scope | The Register system |
| Level | Registering into the system |
| Primary actor | User |
| Stakeholders and interests | The user wants to register to the system to use the program |
| Pre-conditions | The user must have an email address |
| Success Guarantee | The user must not be in the database before |
| Main Success Scenario | 1. The user enters correct credentials in the needed text fields 2. The system does not find the user in the database 3. The system opens the main window |
| Extensions | 1. If the user is in the database, then the system won’t open the main window |
| Special requirements | User must not be in the database |
| Technology and Data variations list | The user can register to the system |
| Frequency of occurrence | First time use |
| Misc. |  |

### Play music use case

After the user logs in or registers to the system, the system will open the main window where the user need to press the “All songs” button to show in the table all the songs in the database. From this point the user will need to select a song and press the play button.

|  |  |
| --- | --- |
| Use case section | Purpose |
| Use case name | Play music |
| Scope | The Player system |
| Level | Listening to music from the system |
| Primary actor | User |
| Stakeholders and interests | The user wants to listen to music |
| Pre-conditions | The user must select a song from the table first |
| Success Guarantee | The user must select a song and hit the play button |
| Main Success Scenario | 1. The user presses the “All songs” button 2. The system retrieves the list of songs 3. The user selects a song from the list 4. The system sends the song to the client machine 5. The user presses the play button |
| Extensions |  |
| Special requirements |  |
| Technology and Data variations list | The user can listen to music from the system |
| Frequency of occurrence | Very often |
| Misc. |  |

### See lyrics use case

|  |  |
| --- | --- |
| Use case section | Purpose |
| Use case name | See lyrics |
| Scope | The Lyrics API |
| Level | Seeing lyrics for a song |
| Primary actor | User |
| Stakeholders and interests | The user wants to see the lyrics from a song. |
| Pre-conditions | The user must select a song from the table first and play it. |
| Success Guarantee | The user must select a song and hit the play button, after which the user presses in the show lyrics label |
| Main Success Scenario | 1. The user presses the “All songs” button 2. The system retrieves the list of songs 3. The user selects a song from the list 4. The system sends the song to the client machine 5. The user presses the play button 6. The user presses the show lyrics label 7. The system opens the lyrics window once the lyrics are fetched |
| Extensions | 1. If the song does not have lyrics it will not show anything |
| Special requirements |  |
| Technology and Data variations list | The user can see lyrics of the songs the user listens to |
| Frequency of occurrence | Very often |
| Misc. |  |

After the user logs in or registers to the system, the system will open the main window where the user need to press the “All songs” button to show in the table all the songs in the database. From this point the user will need to select a song and press the play button, afterwards the user will need to press the show lyrics label.

### Search song use case

|  |  |
| --- | --- |
| Use case section | Purpose |
| Use case name | Search song |
| Scope | The Player system |
| Level | Finding music in the system |
| Primary actor | User |
| Stakeholders and interests | The user wants to find music |
| Pre-conditions | The user must have pressed the “All songs” button |
| Success Guarantee | The user must have pressed the “All songs” button and search for a song |
| Main Success Scenario | 1. The user presses the “All songs” button 2. The system retrieves the list of songs 3. The user chooses what to search for 4. The user types the name of the song or artist 5. The system shows the song |
| Extensions | 1. The song may not be in the database |
| Special requirements |  |
| Technology and Data variations list | The user can listen to music from the system |
| Frequency of occurrence | Not so often |
| Misc. |  |

After the user logs in or registers to the system, the system will open the main window where the user need to press the “All songs” button to show in the table all the songs in the database. From this point the user will need to select from the choice box what is he searching for, f.x. “title” or “artist” after which types the name of the song or artist in the search bar.

# Design

The class diagram for our system is combined, the server and the client being on the same file.

Diagram

Description automatically generated

The top picture represents the server part of our program, it does not have any GUI at all since it is not needed and the product owner didn’t request more than this. The central part of the Server-side is the MainModel where all the magic happens. The MainModel interacts with the sockets, the LoginModel and many other classes.

As for the server type built, the server has been built on Sockets, since there is a lot of control over them and this control was something necessary in the construction of this program.

Up next there is the client side it was chosen the MVVM design pattern meaning that it should have 3 separate packages (model,viewmodel and view) but because the client needs to connect to a server there is also a sockets package and a resources package where the lyrics API relies:

Diagram

Description automatically generated

Like with the server side, on the client side everything resides on the MainModel. This is only the model package since it will be too big to put the whole diagram. The client is missing a lot of the methods from the server but has more classes than the server, fx. fetching lyrics from the internet.

The sockets and the resources package look like this:

A picture containing text

Description automatically generated

The socket part has specialized parts for every action that the client can take and on top of that the client does not need to stay connected continuously, so the client socket will connect to the server’s socket and after the information was received the client will close the connection and it will open a new one when there is a new request.

Diagram

Description automatically generatedThe socket protocol for logging in is the following

When the client model receives the answer true, it will open the next window for client to start selecting songs and using the app properly.

The resources package contains everything related to the lyrics API and the music player, methods that were implemented from a jar file.

Now we have the viewmodel package:

A picture containing text, screenshot, sign

Description automatically generated

The viewmodel is not very complicated, it has indeed a lot of methods in those classes but the way it was designed in MVVM makes it very easy to navigate and make changes. The methods that do not return any properties are just calling for other methods in the model. So the complexity should not be overwhelming.

Diagram

Description automatically generatedThe view package was created with the factory method design pattern:

The view package was the least package that was worked on, since the factory method was done in no time, the MVVM design made it such that we code the logic in the viewmodel so, in terms of the view controllers, it was needed just to bind the properties, and afterwards just call methods on the viewmodel.

Graphical user interface, application

Description automatically generated

This is the interface used for the main window, it contains a lot of functionalities for the user to use. The show lyrics label is a pressable label and it will open a lyrics tab window if a song was selected first, more about this use case on the previous section.

Representing the database: A picture containing graphical user interface

Description automatically generated

There are playlist, liked songs, genre, songwithgenre and artist the table do not contain any information, the only entities that have information is the user table and the song.

Speaking of the database the client does not interact directly with the database but instead makes a request to the server which then interacts with the database. The client does not know anything about the database.

Another design pattern that was implemented was the observer pattern which was used to display the timer in the GUI and display the song name in the label:

A picture containing diagram

Description automatically generated

# Implementation

The team took on the MVVM pattern which was not very complicated since now the model and the view package don’t know anything about each other.

Text

Description automatically generatedThis lead the team to use bindings between the viewmodel and the view.

Text

Description automatically generatedAs a code snippet representing the interaction of the server with the database

This code will connect to the database, it will request the number of songs available, and it will request depending on the count of song all the songs that exist on the database. At the end of the method, it is returning a songList object which then is passed to the client once it has been requested.

Some very interesting code can be found in the ConvertSongToFile class

Text

Description automatically generated

The solution of sending song files from the server to the client was to convert the song file into a bytes array and then send it to the client as a bytes array.

After the byte array gets to the client, this one has a class named ConvertBytesToFile where the bytes are rewritten as a mp3 file once more. Text

Description automatically generated

The code is not complicated but it is interesting to and a lot of the teams time went to accomplish this.

The lyrics api does 100% of the processing so when it comes to the code that we call to display lyrics is very small

Text

Description automatically generated

Throughout the project there were multiple JAR file used to accomplish this, and they are the following:

* jaco-mp3-player-0.9.4.jar
* MyObserver-1.4.jar
* postgresql-42.2.11.jar

As well as javaFX:

* javafx-sdk-17.0.1

The jaco mp3 jar file contributed to the core of the software the team developed. It provided the team with useful functionality and flexibility in making the app work as intended.

# Test

The approach chosen for testing the implemented system was black boxing, which means that the testing was focused more on the output of the program and methods rather than the way they achieve the output.

The team used JUNIT testing and ZOMB+E for both the client side and the server side thus making sure that the use cases are working as expected. The tests are in the APPENDIX FOLDER 3 – SOURCE CODE , SEP-Project-INTELIJ for more details about testing.

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Expected Result** | **Works as expected?** |
| **Sign up** | The user that fills the required fields to register and are not present in the database will be added to the database and redirected to the main window. | Yes |
| **Sign up not successful** | The user fills the required fields to register and is already present in the database. This will prompt the system to show the user an error label | Yes |
| **Log In** | The user that fills required fields to log in and are present in the database will be redirected to the main window. | Yes |
| **Log In not successful** | The user fills out the required fields to log in and is not present in the database. This will prompt the system to show the user an error label | Yes |
| **Play music** | The user that login/registers correctly, presses the “All songs” button, selects a song and plays it. | Yes |
| **See lyrics** | The user that login/registers correctly, presses the “All songs” button, selects a song and plays it the presses the “show lyrics” label will open the lyrics window. | Yes |
| **See Lyrics not successful** | The user that login/registers correctly and then press the “Lyrics” label, the system will open the lyrics window, but no lyrics will be present in the lyrics area | Yes |
| **Search song** | The user that login/registers correctly, presses the “All songs” button, selects what it wants to search for and types in the search bar the name of song/artist will get the song. | Yes |

# Results and Discussion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product backlog | | | | |
| *14/23 completed* | | | | |
| **✓** | **ID** | **Priority** | **Hours estimate** | **Task** |
| TRUE | ~~1~~ | ~~Critical~~ | ~~72 hours~~ | ~~As a user I want to be able to play and listen to music in a console environment.~~ |
| TRUE | ~~20~~ | ~~Critical~~ | ~~18 hours~~ | ~~As a user I want to be able to play and listen to music in a program window.~~ |
| TRUE | ~~2~~ | ~~Critical~~ | ~~36 hours~~ | ~~As a user I want to be able to see the lyrics of the song that I am listening to.~~ |
| TRUE | ~~3~~ | ~~Critical~~ | ~~36 hours~~ | ~~As a user I want to be able to pick a song from a list.~~ |
| TRUE | ~~4~~ | ~~Critical~~ | ~~18 hours~~ | ~~As a user I want to be able to pause or resume the song.~~ |
| TRUE | ~~21~~ | ~~Critical~~ | ~~32 hours~~ | ~~As a user I want to have multiple computers playing from the same list of songs.~~ |
| CANCELED | ~~5~~ | ~~Critical~~ |  | ~~As a user I want to be able to change the volume of the song. (Canceled by the product owner)~~ |
| TRUE | ~~22~~ | ~~High~~ | ~~6 hours~~ | ~~As a user I want to be able to run the program by having a shortcut on my desktop.~~ |
| TRUE | ~~6~~ | ~~High~~ | ~~10 hours~~ | ~~As a user I want to go to the next or previous song.~~ |
| TRUE | ~~7~~ | ~~High~~ | ~~10 hours~~ | ~~As a user I want that after a song finishes, another one will start playing automatically.~~ |
| FALSE | 8 | High | 36 hours | As a user I want to be able to have a liked songs playlist, for a more personal experience. |
| TRUE | ~~9~~ | ~~High~~ | ~~10 hours~~ | ~~As a user I want to be able to search for a song.~~ |
| TRUE | ~~10~~ | ~~High~~ |  | ~~As a user I want to have the songs categorized by genres.~~ |
| TRUE | ~~11~~ | ~~Medium~~ |  | ~~As a user I want to be able to shuffle or repeat my playlist.~~ |
| FALSE | 12 | Medium |  | As a user I want to get song recommendation. |
| FALSE | 23 | Low | 2 hours | As a user I want to know how many hours I spent listening to music. |
| FALSE | 13 | Low |  | As a user I want to be able to change font size, color and style of the lyrics. |
| FALSE | 14 | Low |  | As a user I want additional information about the song like the author, length, and year. |
| FALSE | 15 | Low | 36 hours | As a user I want to have a separate account to not merge my liked songs with another user. |
| TRUE | ~~16~~ | ~~Low~~ | ~~2 hours~~ | ~~As a user I want to be able to toggle between showing lyrics and not showing lyrics.~~ |
| FALSE | 17 | Non-Functional |  | As a user I want to see a playlist with the most listened songs. |
| FALSE | 18 | Non-Functional |  | As a user I want to see my activity, for example how many hours I have listened today. |
| FALSE | 19 | Non-Functional |  | As a user I want the app to go to sleep after some inactivity. |

At the end of sprint 5 the development team is proud to deploy a functional and usable program.

# Conclusions

Giving the time given to the development team they managed in the course of 5 sprints to complete more than half of the requirements. The development required many hours of research when it comes to how to send songs through sockets, how to fetch lyrics, how to design the system and how to make everything work together and be satisfactory for the Product Owner as well.

The team is proud that despite the short time for development, the critical and all but one high priority requirement were met.

# Project future

From a technical perspective the team would like to work with a protection proxy for the login function. Other than that like any other team it was suggested lots of optimisation when it comes to sending the song and to playing the song.

# Sources of information

**Note: Use the standard reference method: Harvard Anglia. A very good reference tool is Mendeley** (Mendeley.com, 2016), **ask VIA Library if you need help.**

Banger, D., 2014. A Basic Non-Functional Requirements Checklist « Thoughts from the Systems front line.... Available at: https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/ [Accessed January 31, 2017].

Business Analyst Learnings, 2013. MoSCoW : Requirements Prioritization Technique — Business Analyst Learnings. , pp.1–5. Available at: https://businessanalystlearnings.com/ba-techniques/2013/3/5/moscow-technique-requirements-prioritization [Accessed January 31, 2017].

Dawson, C.W., 2009. Projects in Computing and Information Systems, Available at: http://www.sentimentaltoday.net/National\_Academy\_Press/0321263553.Addison.Wesley.Publishing.Company.Projects.in.Computing.and.Information.Systems.A.Students.Guide.Jun.2005.pdf.

Gamma, E. et al., 2002. Design Patterns – Elements of Reusable Object-Oriented Software, Available at: http://books.google.com/books?id=JPOaP7cyk6wC&pg=PA78&dq=intitle:Design+Patterns+Elements+of+Reusable+Object+Oriented+Software&hl=&cd=3&source=gbs\_api%5Cnpapers2://publication/uuid/944613AA-7124-44A4-B86F-C7B2123344F3.

IEEE Computer Society, 2008. IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation,

Larman, C., 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,

Mendeley.com, 2016. Homepage | Mendeley. Available at: https://www.mendeley.com/ [Accessed February 2, 2017].

YourCoach, S.M.A.R.T. goal setting | SMART | Coaching tools | YourCoach Gent. Available at: http://www.yourcoach.be/en/coaching-tools/smart-goal-setting.php [Accessed August 19, 2017].

Send|receive|play music files through network. Available at:(Anon., 2022g)

Java Socket Example for sending and receiving byte array. Available at:  
(Anon., 2022h)

JavaFX, table view populated but text is blank/hidden/not visible, solution. Available at: (Anon., 2022f)

Making a search bar in javafx. Available at:(Anon., 2022d)

Playing .mp3 and .wav in Java. Available at: (Anon., 2022a)

Bind TableView items with ObservableList in FXML. Available at: (Anon., 2022e)

Binding Image in Javafx. Available at: (Anon., 2022c)

# Appendices

Appendices can be found inside of the APPENDICES folder inside of the handed-in .zip file.

They are structured by folders in the following manner:

**APPENDIX FOLDER 1 – DOCUMENTS**

APPENDIX 1A – Project Description

APPENDIX 1B – Analysis Document

APPENDIX 1C – Sprint Backlog

APPENDIX 1E – Chad Music Client Tutorial

APPENDIX 1D – Chad Music Server Tutorial

APPENDIX 1F - USER STORIES

**APPENDIX FOLDER 2 – ASTAH**

APPENDIX SUBFOLDER 2A – ACTIVITY DIAGRAMS

APPENDIX 2Aa - Like a song Activity Diagram

APPENDIX 2Ab - Login Activity Diagram

APPENDIX 2Ac - Next Previous song Activity Diagram

APPENDIX 2Ad - Play Pause Activity Diagram

APPENDIX 2Ae - Register Activity Diagram

APPENDIX 2Af - Repeat a song Activity Diagram

APPENDIX 2Ag - Search bar Activity Diagram

APPENDIX 2Ah - Show lyrics Activity Diagram

APPENDIX SUBFOLDER 2B – CLASS DIAGRAM

APPENDIX 2Ba - Class Diagram CHADMusic

APPENDIX SUBFOLDER 2C - DOMAIN MODEL

APPENDIX 2Ca - Domain Model

APPENDIX SUBFOLDER 2E – SEQUENCE DIAGRAMS

APPENDIX 2Ea - Login sequence diagram

APPENDIX 2Eb - Lyrics sequence diagram

APPENDIX 2Ec - Playing sequence diagram

APPENDIX 2Ed - Register sequence diagram

APPENDIX 2Ee - Searchbar sequence diagram

APPENDIX 2Ef - Server-Client login

APPENDIX 2Eg - Server-Client register

APPENDIX 2Eh - Server-Client song

APPENDIX 2Ei - Server-Client songList

APPENDIX SUBFOLDER 2F - USECASE DIAGRAMS

APPENDIX 2Fa - ChadMusic-WholeSystem

APPENDIX 2Fb - Liking a song UseCase Diagram

APPENDIX 2Fc - lyrics UseCase Diagram

APPENDIX 2Fd - pause UseCase Diagram

APPENDIX 2Fe - Register UseCase Diagram

APPENDIX 2Ff - Repeat a song UseCase Diagram

APPENDIX 2Fg - Search for a song UseCase Diagram

APPENDIX 2Fh - UseCase Diagram for login

**APPENDIX FOLDER 3 – SOURCE CODE**

REQUIRED-FILES

SEP-Project-INTELIJ

SEP-DataBase

ARTIFACTS

**APPENDIX FOLDER 4 – JAVADOC**

SERVER (server javadoc)

CLIENT (Client javadoc)

**Appendix A Project Description**

Insert the original Project Description here