|  |
| --- |
| C:\year 5\Git\Report\FYP-Report\gmit-logo-2012rgb.jpg |
| Music Host Interface |
| B.Eng (Hons) in Computer and Electronic Engineering  GMIT |
|  |
| **Thomas Flynn** |
| **May, 2016** |

|  |
| --- |
|  |

# 

# Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer & Electronic Engineering at GalwayMayo Institute of Technology. This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

# Acknowledgements

First of all, I would like to thank my classmates. Throughout the past few years they have assisted and encouraged my course work and provided some invaluable feedback on this project. Developing a project with a completely personal scope can result in some disorientation and uncertainty of progress and evaluation. I could not have completed this without their help and I wish them the best for the future.

I would also like to thank the course lecturers, staff and project supervisors for their guidance and assistance during the course of this project and over the past few years. They have always encouraged creativity in the course and I feel this has motivated me to undertake a project at this scope.

A special thanks to my project supervisor Brian O'Shea for providing me with valuable insight and support throughout the development of this project.

And a final thanks to my close friends and supportive family for all their help, both related and unrelated, to the development of this project and over the duration of my college years.

# Summary

Music has been an important tbf

The foundation of this project was built upon the idea of a tbf

The finished application is the combination of these ideas. The end user is presented

# Contents

# 1. Introduction

## 1.1 Project goals

This project ....

## 1.2 Project motivation

My motivation...

## 1.3 About Music Hosting

Music hosting is...

## 1.4 About The Internet of Things

The internet of things...

## 1.5 Report Overview

# 2. Project Plan

## 2.1 Gantt chart

## 2.2 Trello and Time Management

# 3 Block Diagrams

The block diagram should be a ‘system’ block diagram. For example, if you have a networked project, show this on the block diagram; if you have layers of software entities you are using, show this on the block diagram.

# 4 Flow Charts

## 4.1 JavaFX Flow Chart

## 4.2 Android Flow Chart

## 4.3 Communication Flow Chart

# 5 JavaFX Music Host Graphical User Interface

The workflow for this project was relatively straightforward. Research and Investigation was handled through prototype development. These prototypes then segmented the project into core functionality features and ultimately decided the most efficient method to implement these features into an application. At this stage, the basic requirements for operation were listed so a rational decision could be made for which tools to use to implement the main features. Once the tools had been selected, features were gradually implemented with core functionality features being a priority. As the project deadline approached, the final feature set was completed so the project could reach a "finished" stage.

My mentality for building this project was to avoid any guidelines or set paths, so I could be free to include any functionality I deemed relative to the project at any stage without diverging from a set plan or timescale. This made the entire process much more adaptive and creative allowing for a much more instinctive workflow.

## 5.1 Research and Investigation

The solution for this project was designed to be a software application with an interface for tbf . From a basic perspective, this solution requires both knowledge of tbf and how it can be used for tbf and the selection of a programming language to write the application in.

### 5.1.1 Graphical User Interface API

Graphical User Interface libaries are tbf...

### 5.1.2 Cloud Services

Cloud services are tbf.

AWT and Google Cloud are tbf..

### 5.1.3 Bluetooth

Bluetooth has been around since tbf. It operates using frequency hopping spread dspectrum technique at 2.14 GHZ. tbf

## 5.2 Requirements

From the features defined, a number of requirements and dependencies were raised for the project to meet functionality requirements. These requirements were then segmented into 3 core feature elements. tbf

### 5.2.1 Graphical User Interface

Microsoft JDBC 4.0 is the minimum driver version required in order to make a connection to a Microsoft Azure SQL Server. tbf

### 5.2.2 Sound Engine

The sound engine should be capable of playing mp3 files. tbf

### 5.2.3 Database as a service

The remote database...tbf

### 5.2.4 Bluetooth Server

The Bluetooth library should provide the necessary API to run a server.

## 5.3 Tool Choices

### 5.3.1 Microsoft Azure SQL Server

Microsoft JDBC 4.0 is the minimum driver version required in order to make a connection to a Microsoft Azure SQL Server. tbf

### 5.3.2 JavaFX API

The Media API for javaFX is tbf.

### 5.3.3 Bluecove-2.1.1-SNAPSHOT Library

The Bluecove-2.1.1-SNAPSHOT Library offers many different tbf.

### 5.4 Foundation

The foundation for the GUI was built on tbf@github example. Taking advantage of the functionality already implemented in the scene switching feature. The example has the ability to add and remove Pane Nodes to the scene graph.

This provided the necessary functionality for the *LoginView.fxml* and *MainView.fxml* to be able to fade in and out. Business logic for logging in and out was built on this transition feature.

### 5.4.1 Foundational Architecture

Figure 1 shows tbf

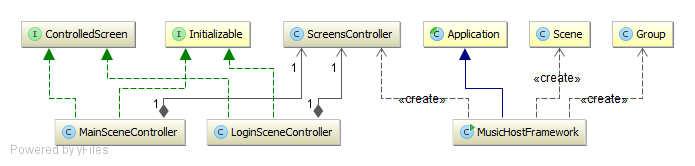


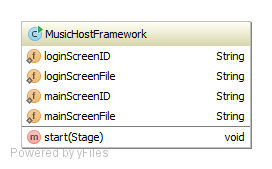
Figure : Foundational Architecture Design

Figure 2 tbf

## 5.5 Realisation

With all of the tools selected and a basic architectural layout designed, the realisation process begun. The application went through much iteration before reaching the final stage. Features were implemented gradually in different versions[[1]](#endnote-2) starting with core functionality such as motion tracking and audio file playback.

### 5.5.1 MusicHostFramework

MusicHostFramework tbf

sada

Figure: MusicHostFramework Class

asd

asd

a

asasdasd

Figure tbf Shows the relationship that was previously described between the classes.

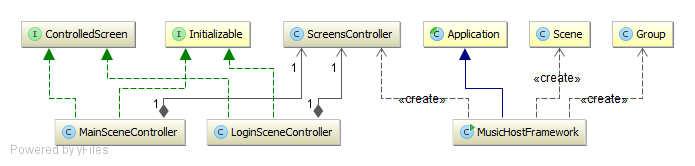


Figure: MusicHostFramework UML

### 5.5.2 ScreensController

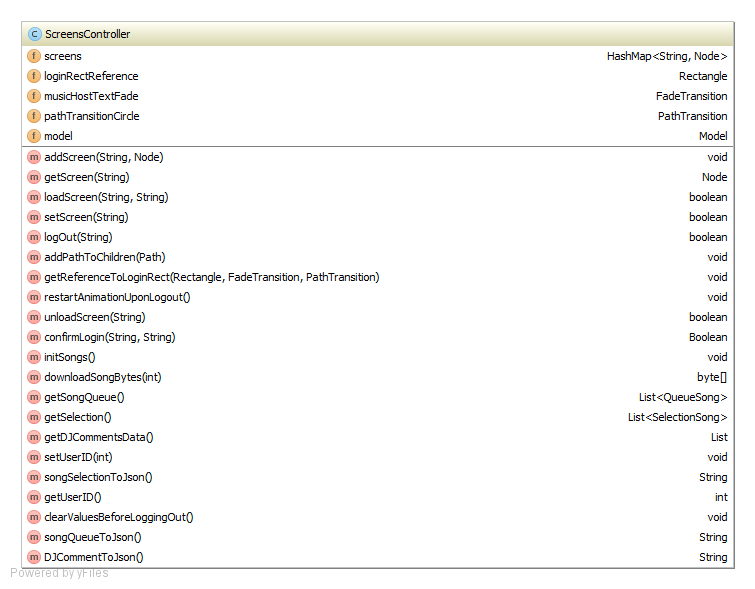


Figure: ScreenController UML

The *ScreensController* acts as the center hub for all business logic within the program.

The hashmap *screens* contains the nodes *LoginView.fxml* and *MainView.fxml*. This

Figure tbf shows the relationship that was previously described between all the classes in the application. tbf

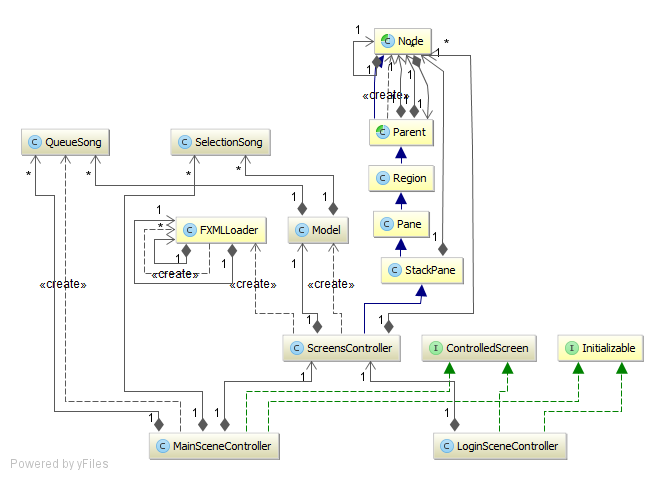


Figure: ScreensController UML

### 5.5.1 LoginScreenController

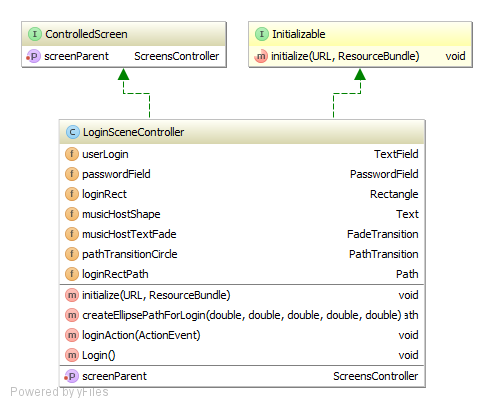


Figure: LoginSceneController Class

The LoginScreenController tbf

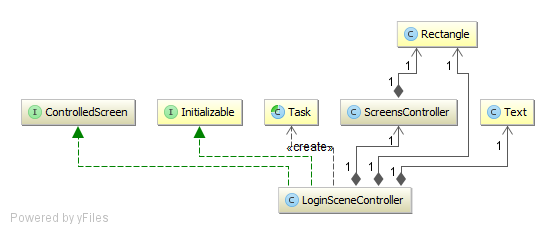


Figure: LoginSceneController UML

### 5.5.1 MainSceneController

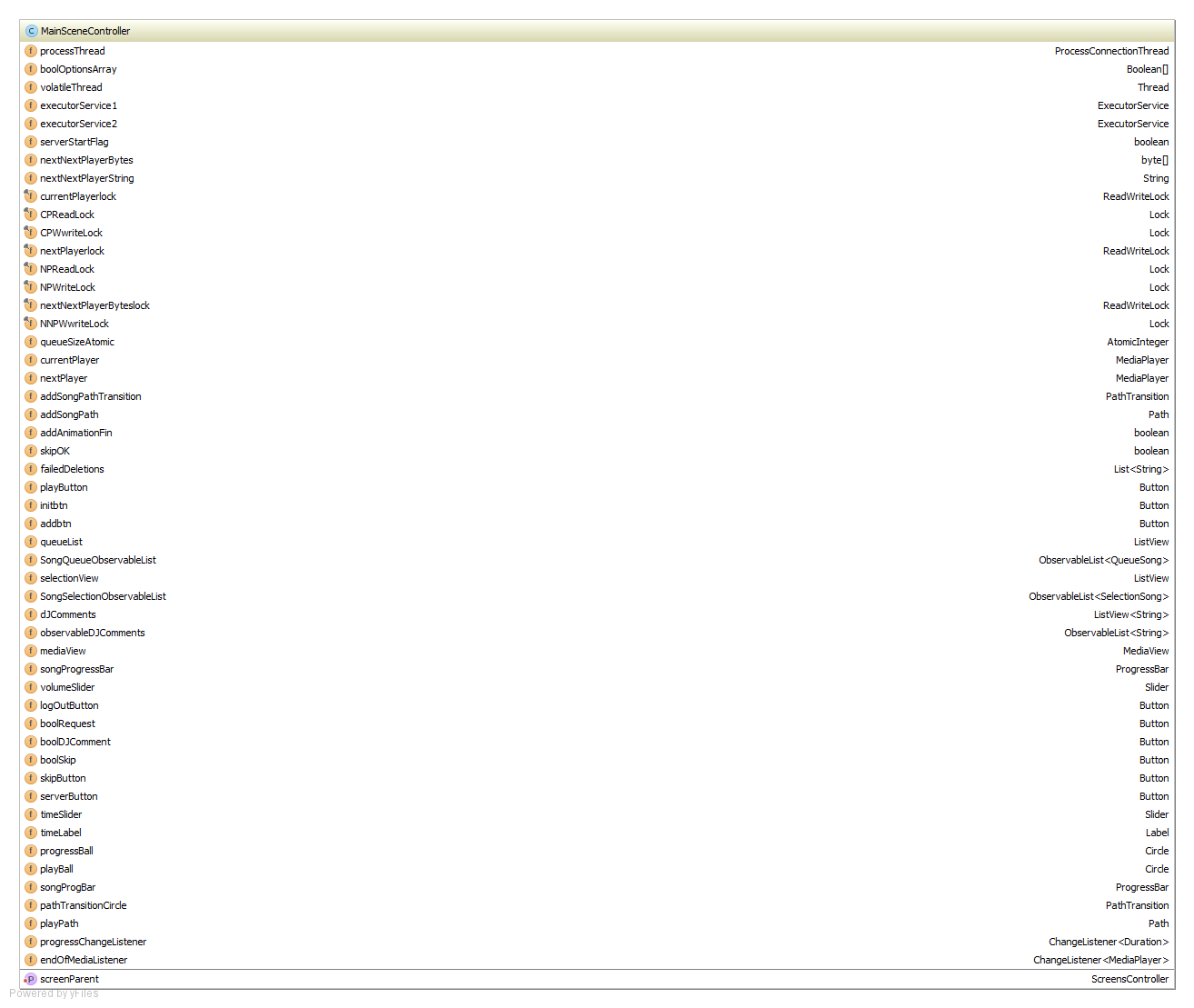


Figure: MainSceneController Fields

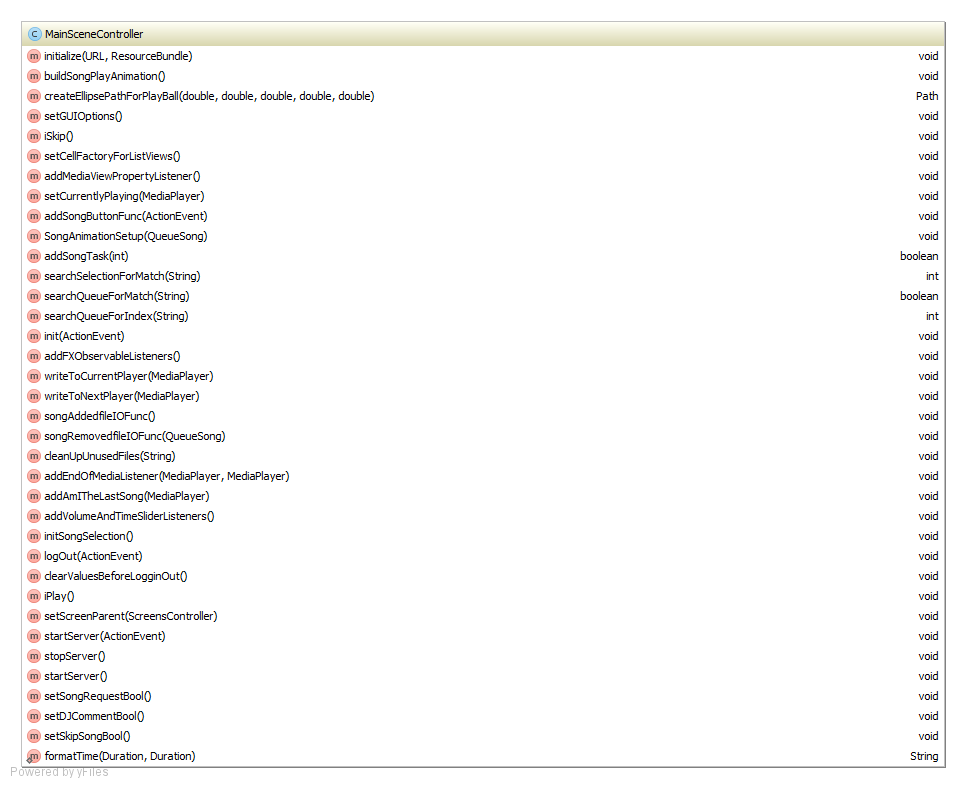
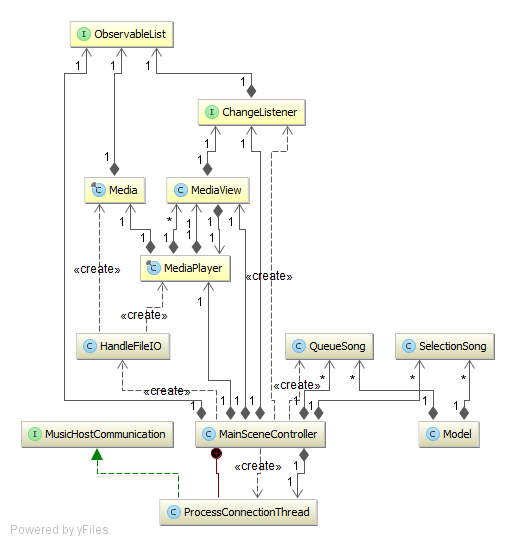
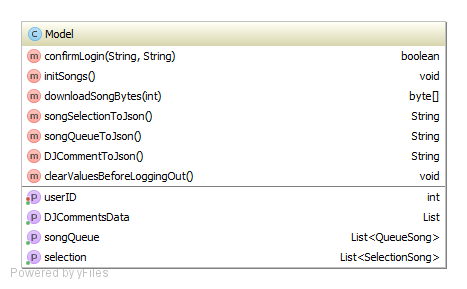
****

Figure: MainSceneController Methods

****

### 5.6.1 Model

The *Model* class holds the key properties for the *LoginController* and the *MainSceneController*.

tbf

tbf

tbf

sdfsdf

Figure :Model Class

dsf

sdf

sdf

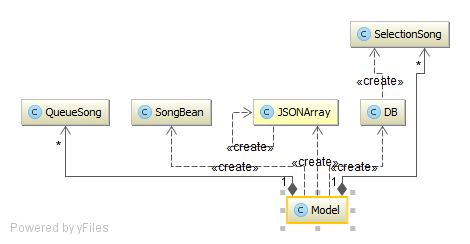
sdf  


Figure :Model UML

### 5.6.2 DB

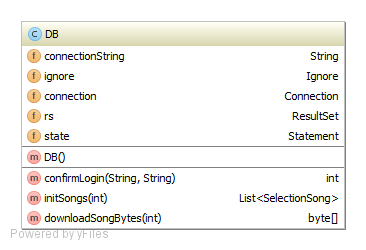
The DB class holds the connection to the remote database. It obtains it's connection string from the *Ignore* class *getCon()* method.

Figure : Model Class

### 5.6.2 DB

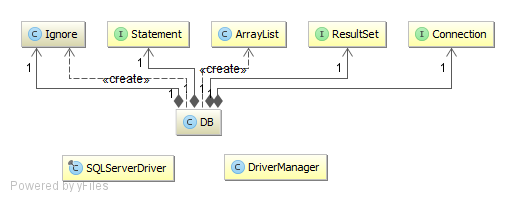
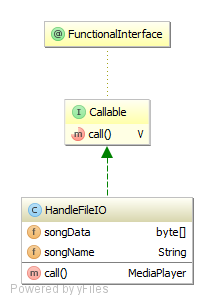


Figure : DB UML

### 5.6.2 HandleFileIO

HandleFileIO tbf

asd

asd

as

das

da

sd

asda

Figure : HandleFileIO class

### 5.6.1 ProcessConnectionThread

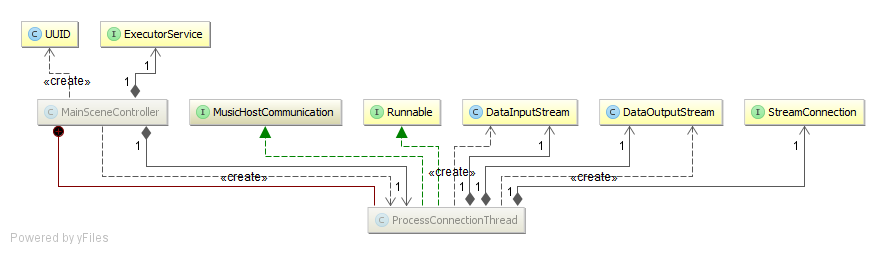


Figure : ProcessConnectionThread UML

## 5.7 Operation

This section describes how the application works at run time and how the various functional elements interact with each other.

### 5.7.1 Setup Sequence Diagram

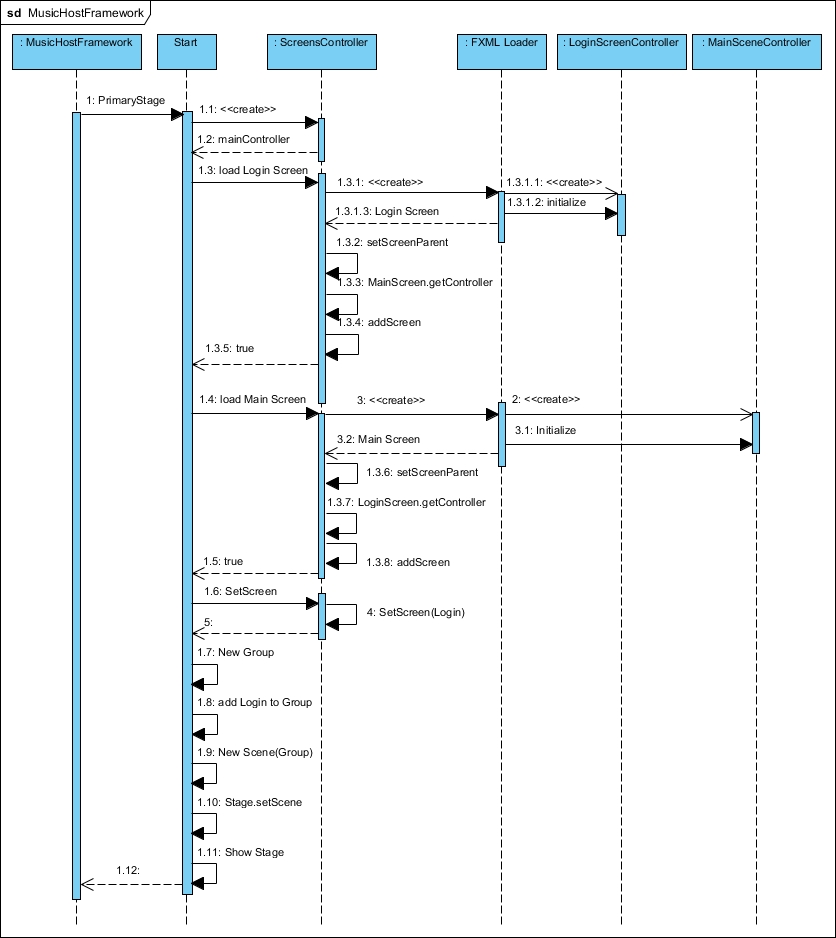


Figure : Setup Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.2 Login Sequence Diagram

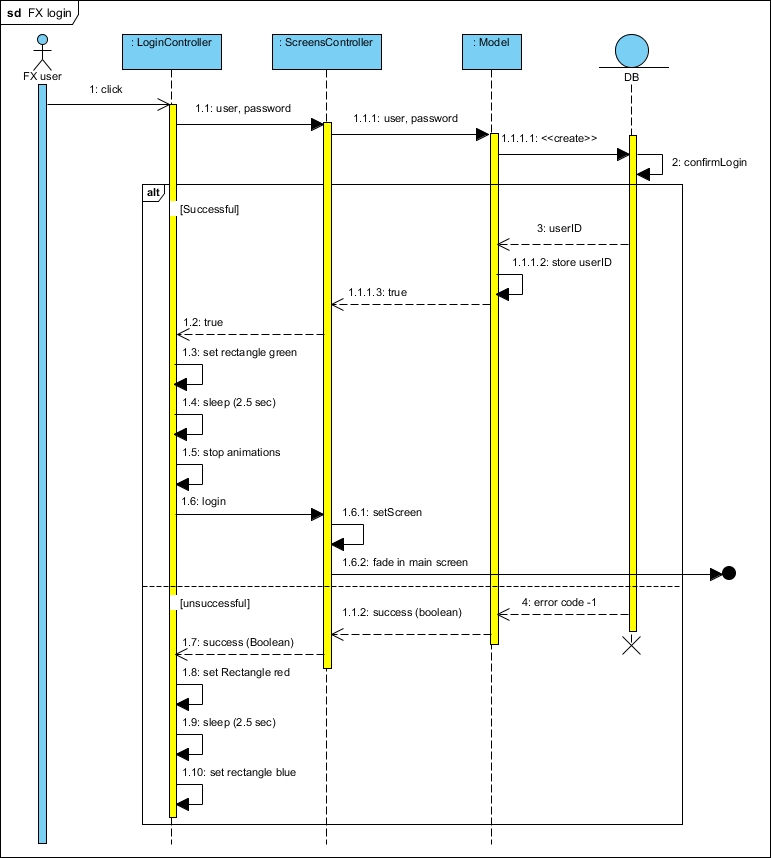


Figure : Login Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.3 Initialize Button Sequence Diagram

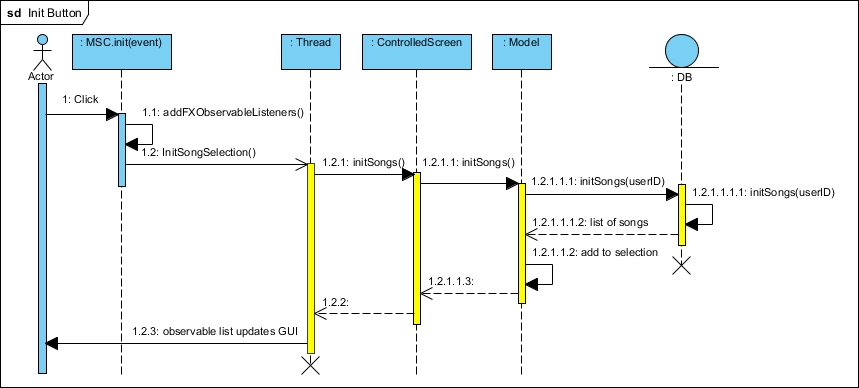


Figure : Initialize Button Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.4 Add Button Sequence Diagram

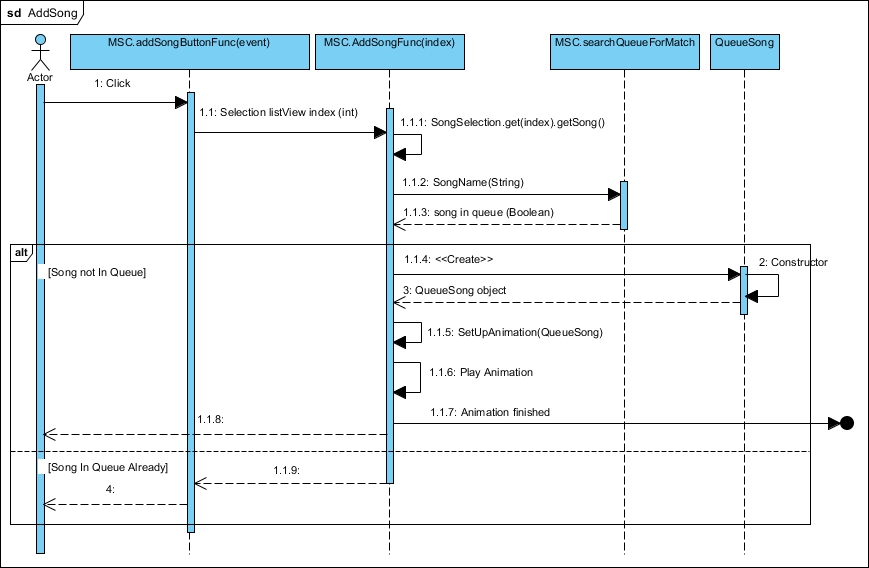


Figure : Initialize Button Sequence Diagram

### 5.7.5 Add Song Animation Ended Event Sequence Diagram

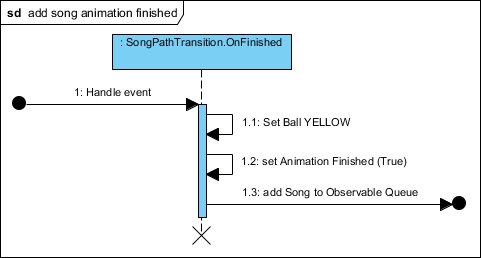


Figure : Initialize Button Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

### 5.7.6 Song Added Event Sequence Diagram

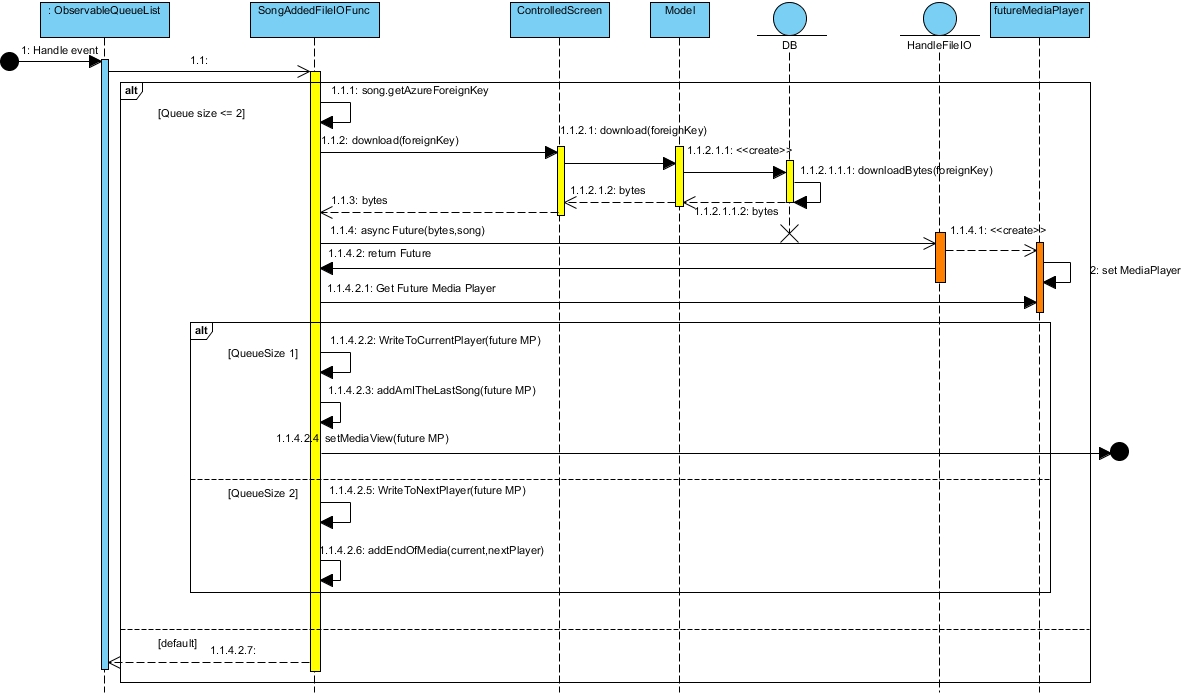


Figure : Song Added Event Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

Alt: [QueueSize == 1]

1.1.4.2.2 Assign the newly created MediaPlayer to the *currentPlayer* field.

1.1.4.2.3 Add an end of media listener to the MediaPlayer that just removes itself from the queue once it reaches the end of media.

1.1.4.2.4 Assign the new MediaPlayer to the MediaView triggering an event within the MediaView.

Alt: [QueueSize == 2]

1.1.4.2.5 Assign the newly created MediaPlayer to the *nextPlayer* field.

1.1.4.2.6 Add an end of media listener to the *currentPlayer* that will play the *nextPlayer* when it ends*.*

Alt: [default]

1.1.4.2.7 Return

### 5.7.6 Song Removed/Ended Event Sequence Diagram

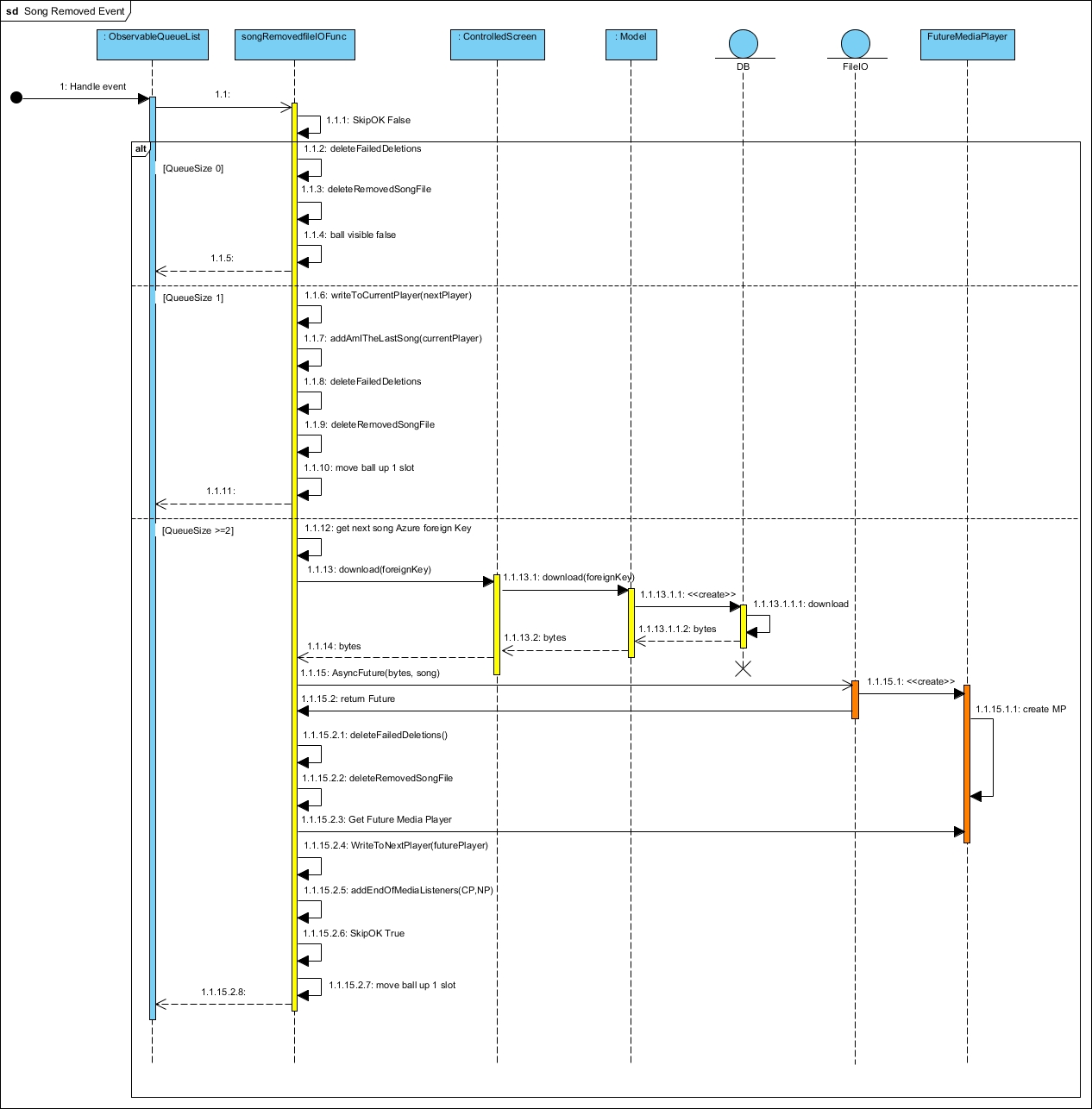


Figure : Song Removed/Ended Event Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

Alt: [QueueSize == 1]

1.1.4.2.2 Assign the newly created MediaPlayer to the *currentPlayer* field.

1.1.4.2.3 Add an end of media listener to the MediaPlayer that just removes itself from the queue once it reaches the end of media.

1.1.4.2.4 Assign the new MediaPlayer to the MediaView triggering an event within the MediaView.

Alt: [QueueSize == 2]

1.1.4.2.5 Assign the newly created MediaPlayer to the *nextPlayer* field.

1.1.4.2.6 Add an end of media listener to the *currentPlayer* that will play the *nextPlayer* when it ends*.*

Alt: [default]

1.1.4.2.7 Return

### 5.7.6 Logout Sequence Diagram

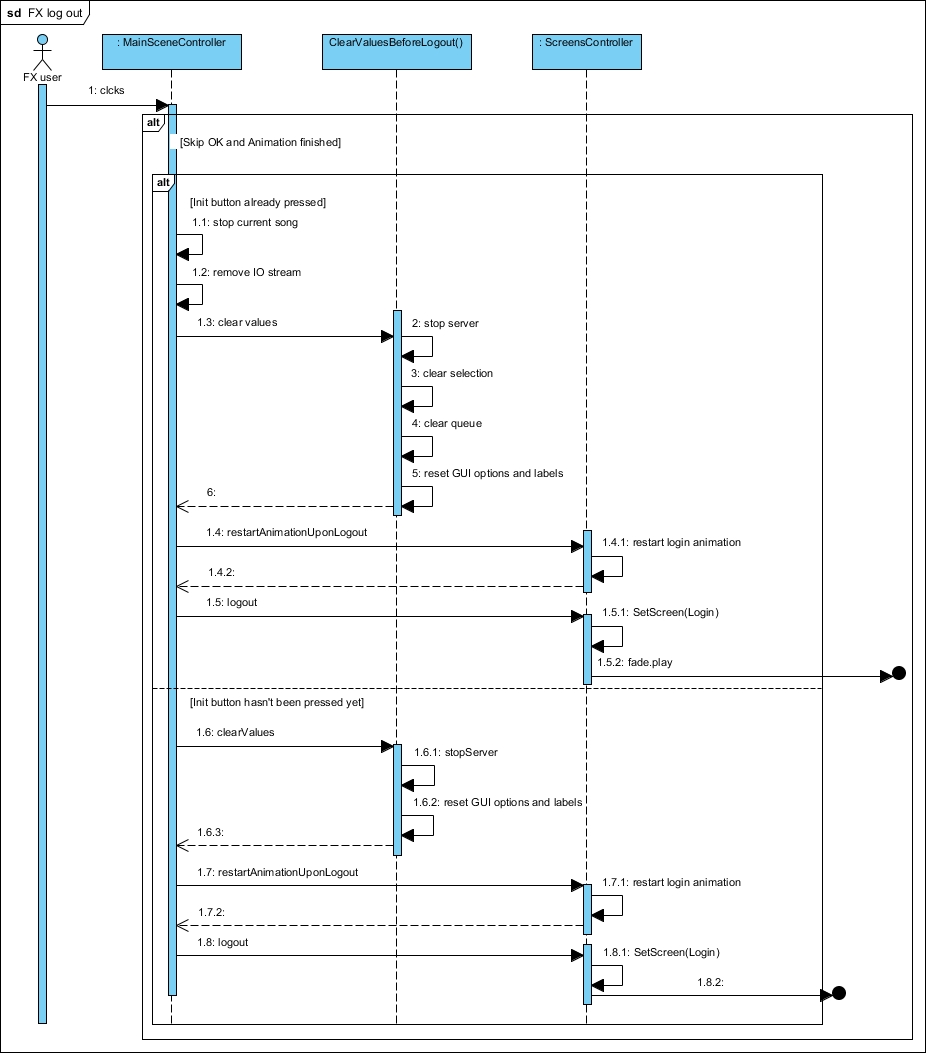


Figure : Logout Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.6 Server Button Sequence Diagram

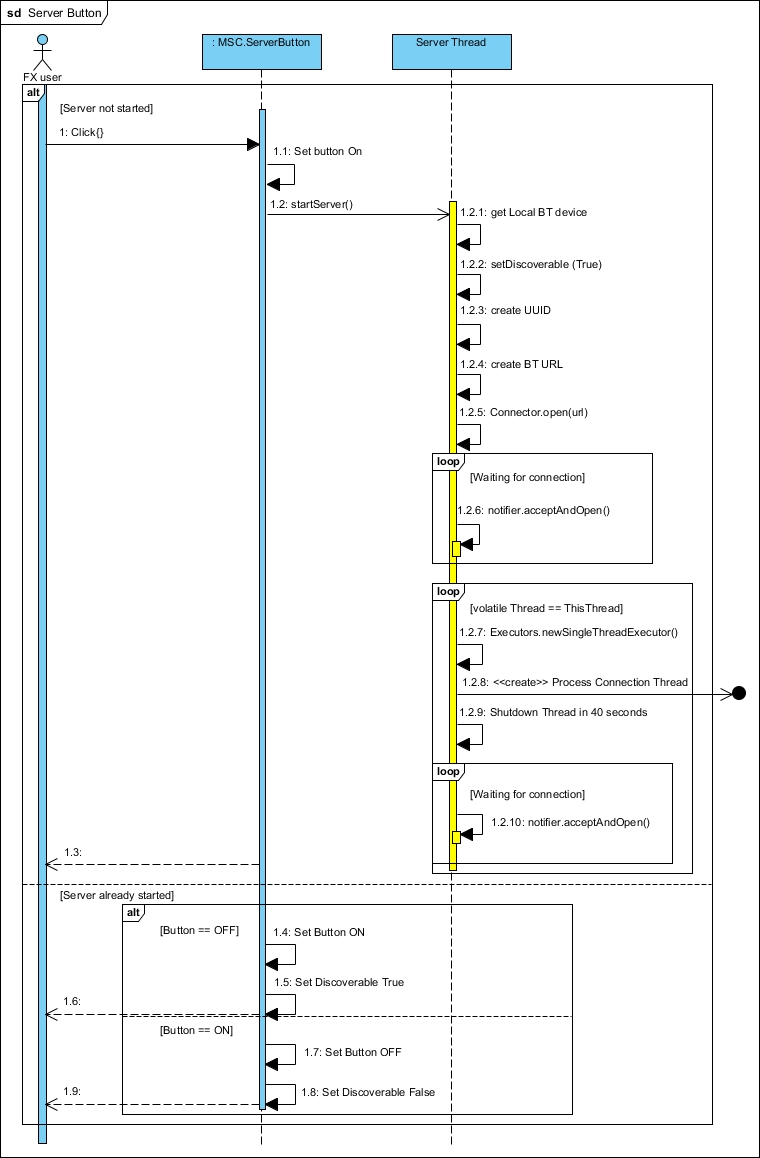


Figure : Logout Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.6 ProcessConnectionThread Sequence Diagram

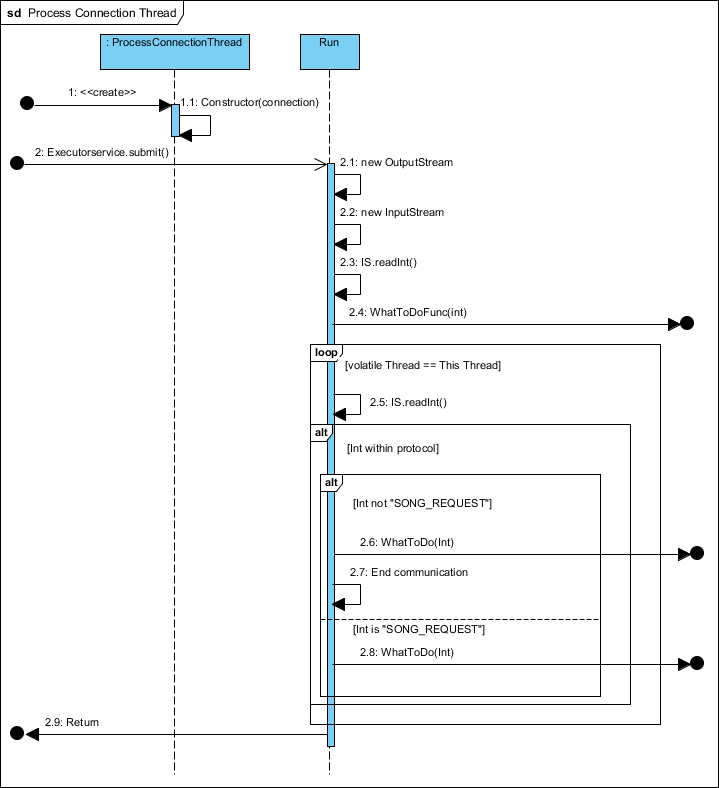


Figure : Logout Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.6 WhatToDoFunc Options, Song Request, Song Selected Sequence Diagram

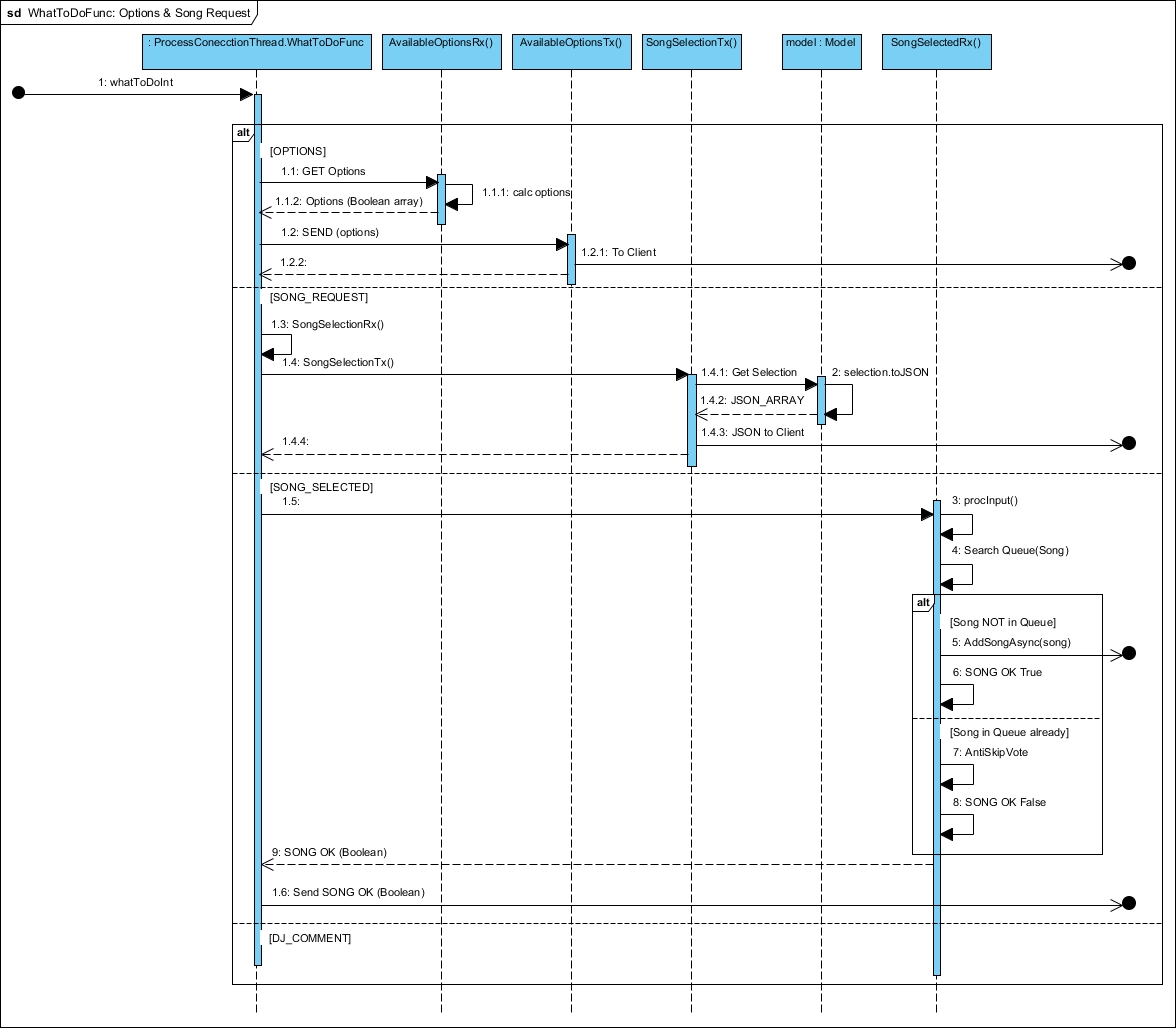


Figure : WhatToDoFunc, Options, Song Request, Song Selected Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.6 WhatToDoFunc DJ Comment Sequence Diagram

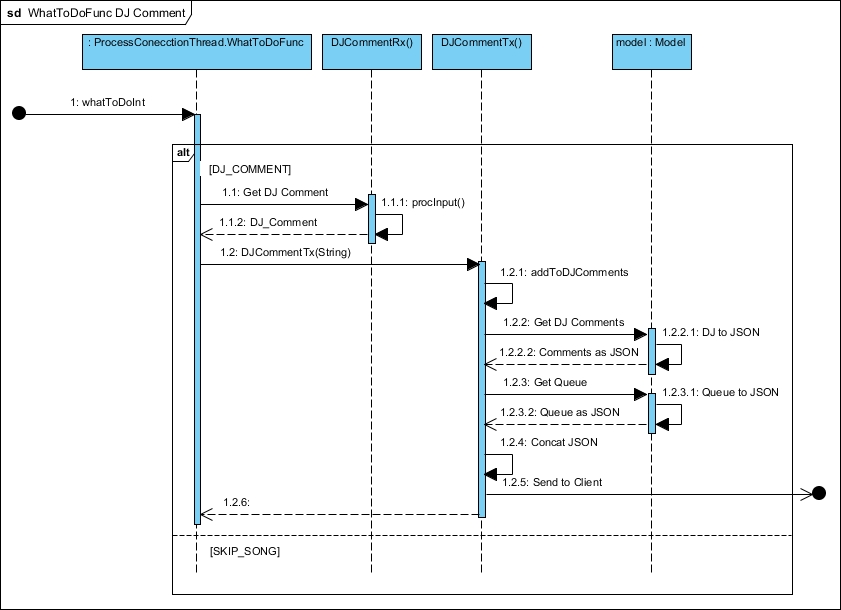


Figure : WhatToDoFunc DJ Comment Sequence Diagram

1. Handle Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

### 5.7.6 WhatToDoFunc Skip Song Sequence Diagram

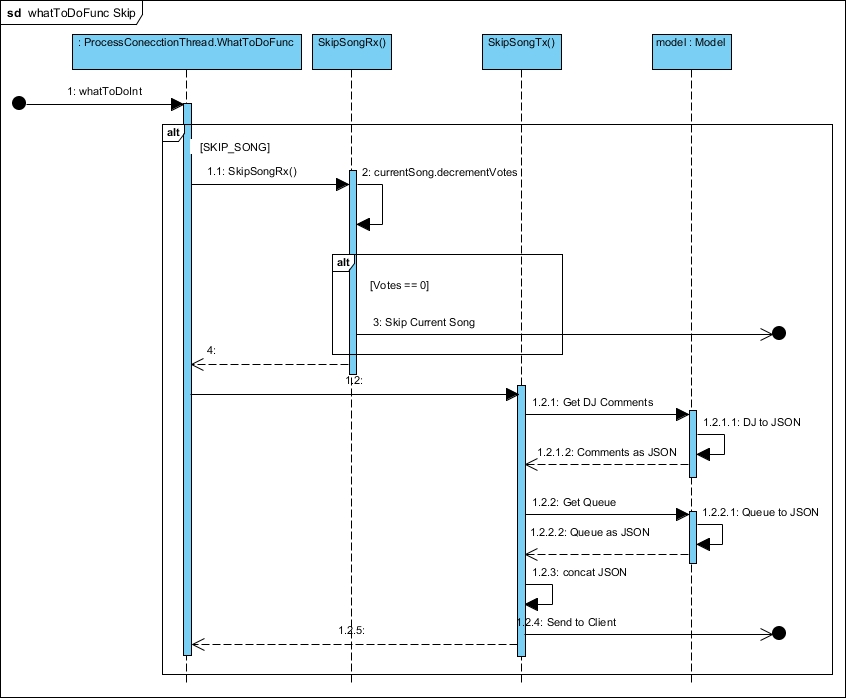


Figure : WhatToDoFunc Skip Song Sequence Diagram

Event

1.1 Start asynchronous SongFileIOFunc task

Alt: [QueueSize <= 2]

1.1.1 Acquire the foreign key of the added song.

1.1.2 Call download function

1.1.2.1 Call download function

1.1.2.1.1 Create DB object

1.1.2.1.1. Download the bytes of the added song.

1.1.3 Bytes returned after downloading.

1.1.4 Start Async Future HandleFileIO

1.1.4.1 Create an mp3 file from the downloaded bytes and then construct a MediaPlayer.

1.1.4.2.1. Get Future MediaPlayer

# 6 Android Music Host Client

The workflow for this project was relatively straightforward. Research and Investigation was handled through prototype development. These prototypes then segmented the project into core functionality features and ultimately decided the most efficient metho tbf

## 6.1 Research and Investigation

tbf

### 6.1.1 Graphical User Interface API

tbf

### 6.1.2 Cloud Services

Cloud services are tbf.

AWT and Google Cloud are tbf..

### 6.1.3 Bluetooth

Bluetooth has been around since tbf. It operates using frequency hopping spread dspectrum technique at 2.14 GHZ. tbf

## 6.2 Requirements

From the features defined, a number of requirements and dependencies were raised for the project to meet functionality requirements. These requirements were then segmented into 3 core feature elements. tbf

### 6.2.1 Graphical User Interface

Microsoft JDBC 4.0 is the minimum driver version required in order to make a connection to a Microsoft Azure SQL Server. tbf

### 6.2.2 Sound Engine

The sound engine should be capable of playing mp3 files. tbf

### 6.2.3 Database as a service

The remote database...tbf

### 6.2.4 Bluetooth Server

The Bluetooth library should provide the necessary API to run a server.

### 6.4 Foundation

The foundation for the GUI was built on tbf@github example. Taking advantage of the functionality already implemented in the scene switching feature. The example has the ability to add and remove Pane Nodes to the scene graph.

This provided the necessary functionality for the *LoginView.fxml* and *MainView.fxml* to be able to fade in and out. Business logic for logging in and out was built on this transition feature.

### 6.4.1 Foundational Architecture

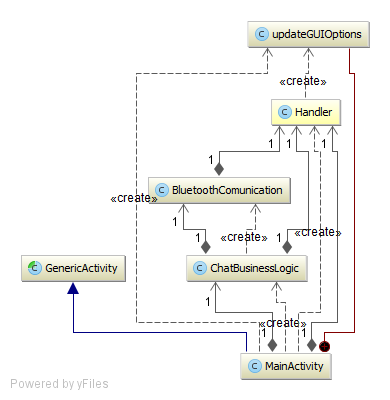
Figure 1 shows tbf

Figure : Foundational Architecture Design

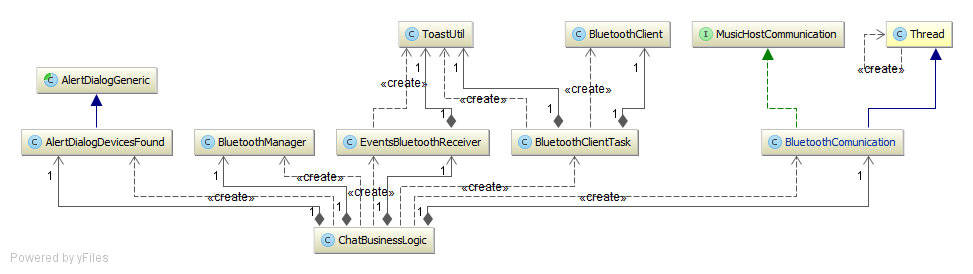
## 6.5 Realisation

With all of the tools selected and a basic architectural layout designed, the realisation process begun. The application went through much iteration before reaching the final stage. Features were implemented gradually in different versions[[2]](#endnote-3) starting with core functionality such as motion tracking and audio file playback.

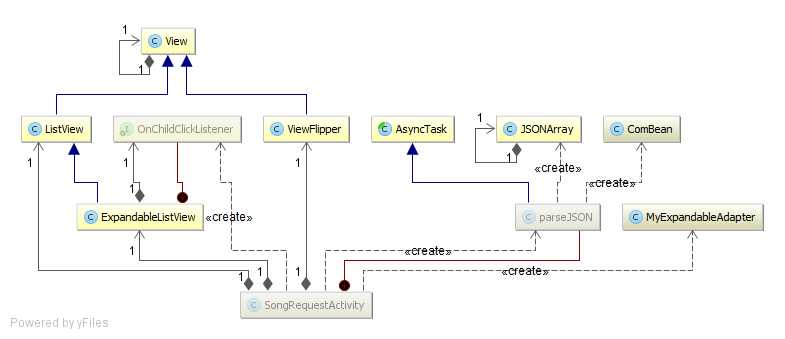
### 6.5.1 Music Host Client Architecture



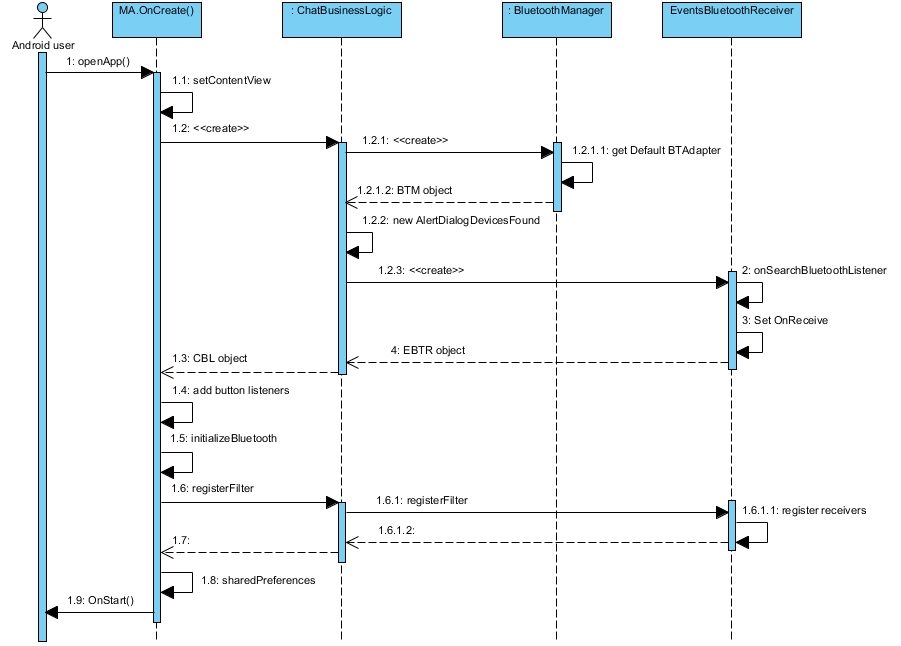
### 6.5.1 Music Host Client Architecture



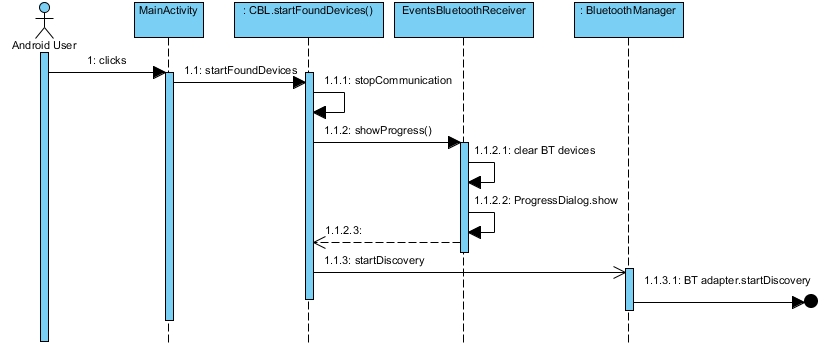
### 6.5.1 Music Host Client Architecture



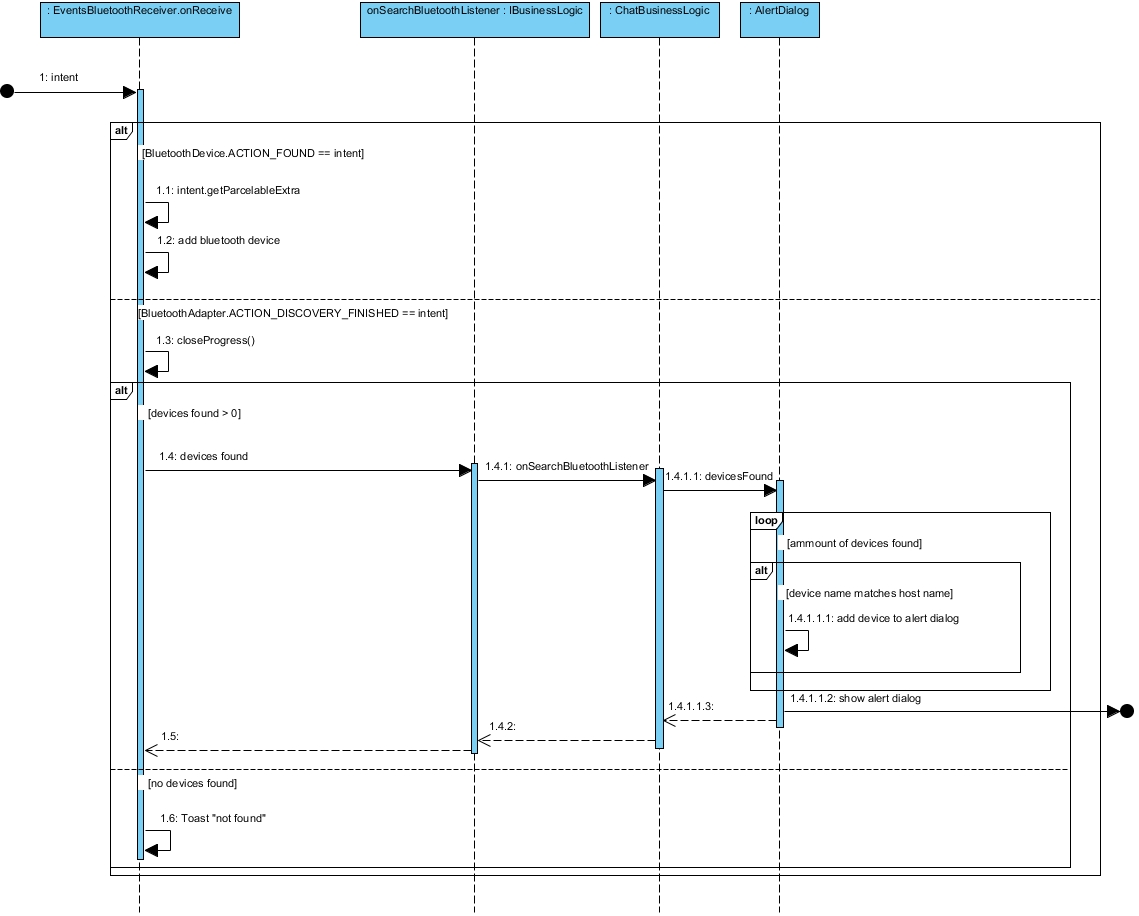
### 6.5.1 Music Host Client Architecture



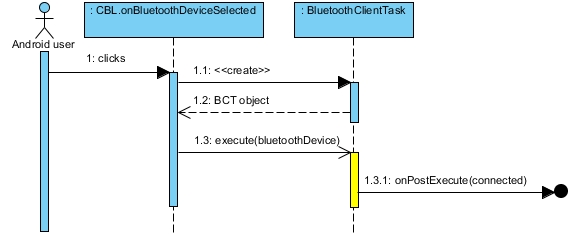
### 6.5.1 Music Host Client Architecture



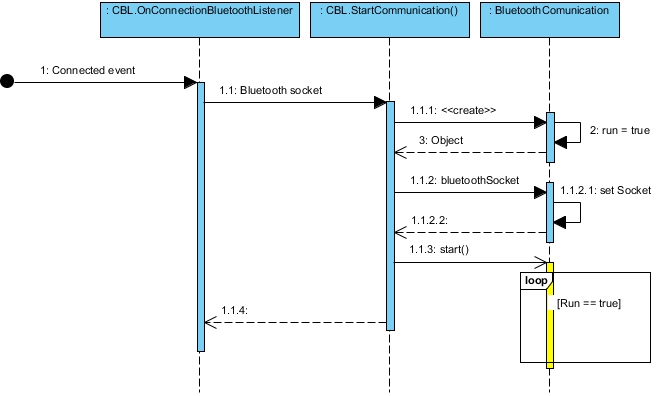
### 6.5.1 Music Host Client Architecture



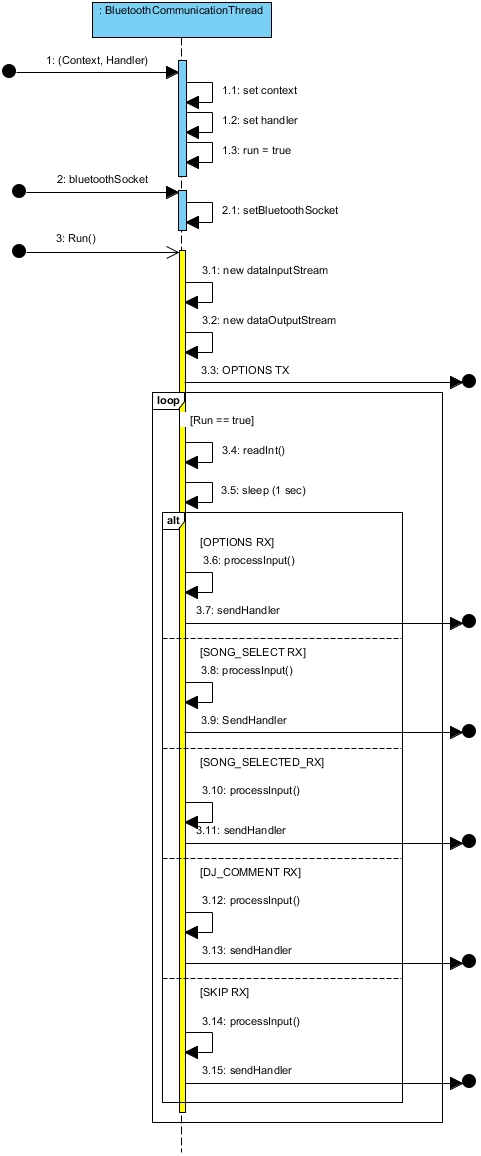
### 6.5.1 Music Host Client Architecture



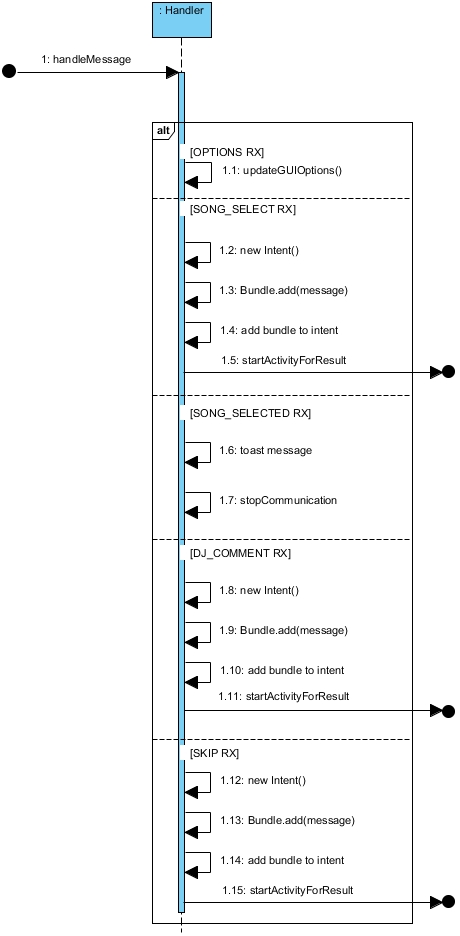
### 6.5.1 Music Host Client Architecture



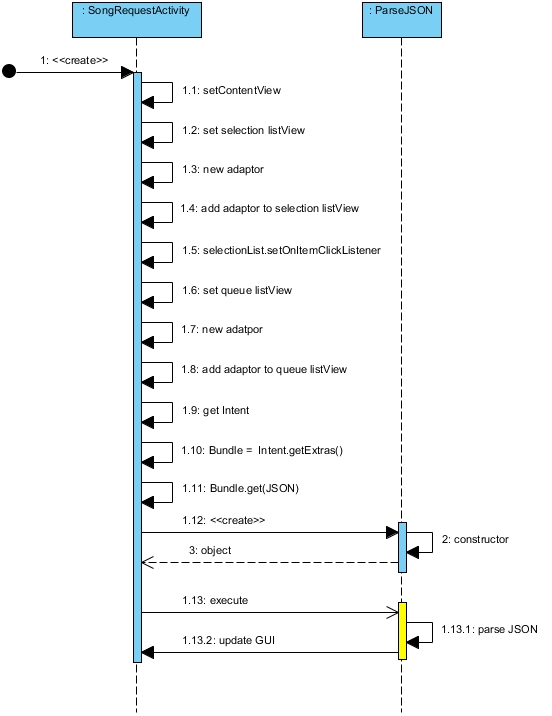
### 6.5.1 Music Host Client Architecture



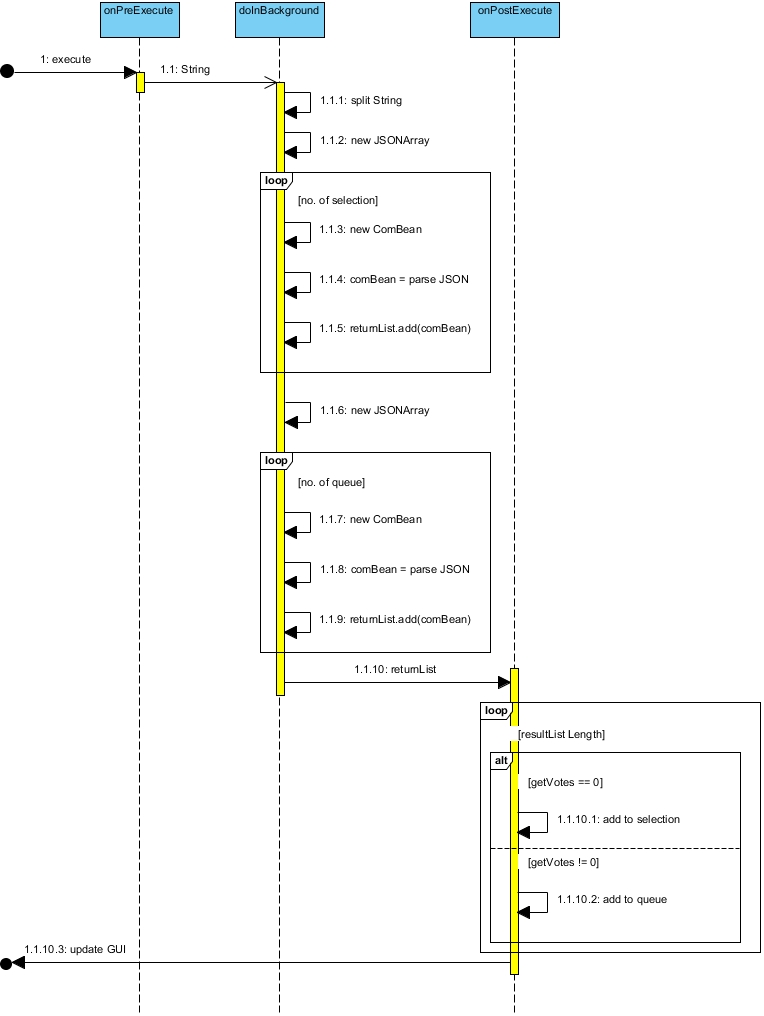
### 6.5.1 Music Host Client Architecture



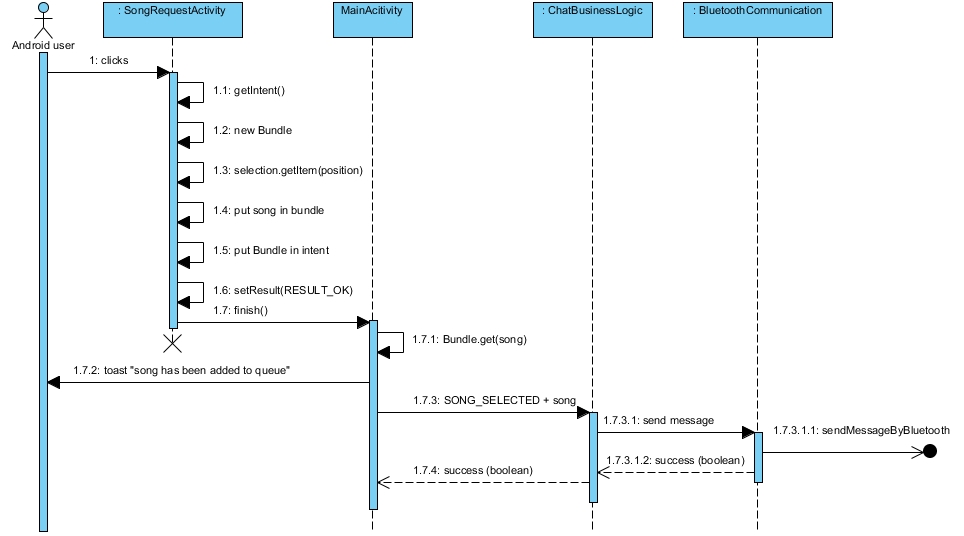
### 6.5.1 Music Host Client Architecture



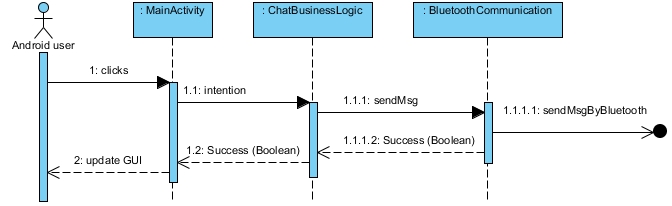
### 6.5.1 Music Host Client Architecture



### 6.5.1 Music Host Client Architecture



### 6.5.1 Music Host Client Architecture



### 6.5.1 Music Host Client Architecture

1. Github Version Commits: github.com/freshfunkee/KinectMusicController/commits/master [↑](#endnote-ref-2)
2. Github Version Commits: github.com/freshfunkee/KinectMusicController/commits/master [↑](#endnote-ref-3)