# Team X

Requirements Document

Rebecca Young, Lineker Tomazeli, Alicia Bendz

November 4, 2011

COMP361 Software Engineering Project

School of Computer Science

McGill University

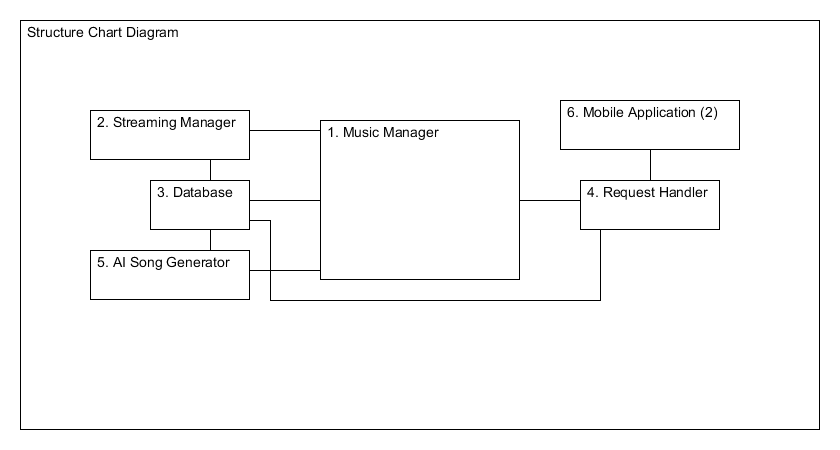
I. Introduction

II. Overall Description

**2.2 Product Functions**

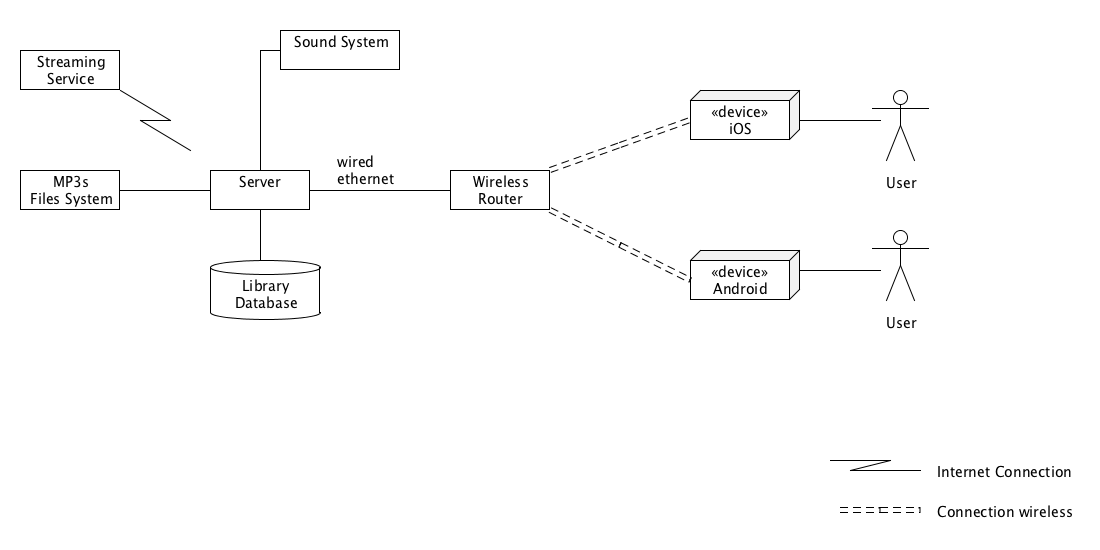
Structure Chart Diagram and Identification of Modules with Functions

The system consists of 6 distinct modules:



|  |  |  |
| --- | --- | --- |
|  | Module | Functions and responsibilities |
| 1 | Music Manager | The Music Manager is the hub of all the modules. It contains and monitors the Active Queue, Playlist, and Music Player, and appropriately processes all song requests sent from the Request Handler. It sends the corresponding success messages back to the Request Handler once the request has been processed.  This module also monitors the Active Queue, assuring that the number of songs there do not fall below the minimum requirement. When a certain number is reached, it requests another set from the Artificial Intelligence Song Generator and adds it to the Active Queue. The Active Queue which keeps track of its list of songs and their playing times is controlled directly by the Music Manager. |
| 2 | Streaming Manager | The Streaming Manager connects to the server from which music files are streamed. It permits searching among available files, oversees the streaming of selected files, and returns status and success messages at the request of a song stream. It is the direct and only link between the streaming server and the rest of the system. |
| 3 | Database | The Database contains all the user saved libraries, as well as copies of all the songs loaded by the user, whether they are in a Playlist or not. The source of each song may be local or streamed. |
| 4 | Request Handler | The Request Handler is the direct link between the client applications and the Music Manager. It receives all song requests and text feedback which have successfully been transmitted across the network. Feedback is compiled into a special file in the database, and song requests are verified to be valid requests and existing in the database, and then passed to the Music Manager. For each request, the Request Handler waits for the Music Manager to notify it of the request’s status, and returns a success or error message to the client application. |
| 5 | Artificial Intelligence Song Generator | This module contains all the intelligent algorithms which generate sets of songs when requested by the Music Manager. The generator has access to all the songs in the database and uses the song properties, time, and date (season) to determine current popular songs. |
| 6 | Mobile Application (2) | This module has two representations which are developed separately but have identical functions and responsibilities. Each application requires a working local network, and allows users to (1) view the current Active Queue at that site, (2) select a song from it and make a request, and (3) send text feedback to the organization. |

Environment diagram



The environment diagram above gives a representation of the physical environment, and the software application required for our application to be functional.

The minimum requirement for our application is to have a computer to act as a server. The server has to be connected to the background sound system (speakers) and have at least some music files (mp3) in the local computer or a internet connection so streaming can be activated.

By having access to the music files or streaming, we can start using the server to manage libraries and play music.

By importing songs and managing libraries the database will be populated, creating relationships between songs and different libraries that the administrator can create. This data will also be available for statistical purposes.

To extend the application for user interaction with the sound system, it is required to have a router wireless where users can connect their devices to the network and communicate with the Server. The server must be connected to the same router and network that the users' devices will be connected. Devices running iOS or Android OS will be supported. The user has to download an application which will be available in the Apple Store and Android Market Place.

The administrator of the server can make libraries available to users, so they can suggest songs to the current playing queue.

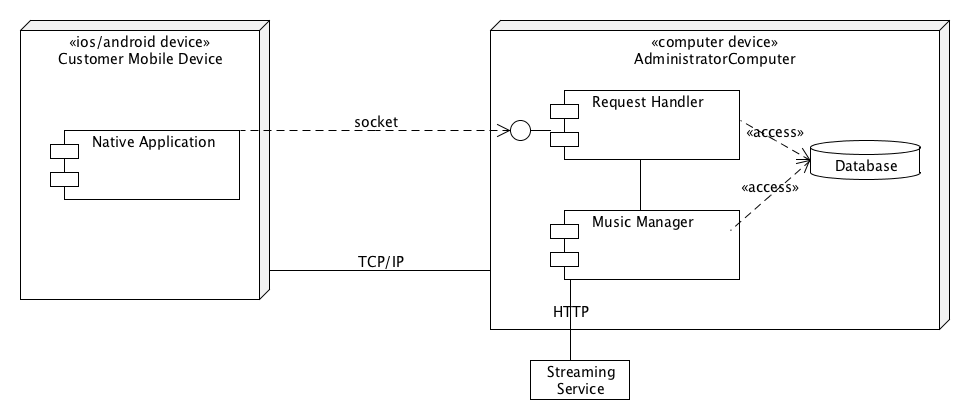
The user will also be able to send a feedback message to the administrator, increasing the interaction between users and administrator.

The Server can fully work without the internet, but the streaming feature will be deactivated, which forces the administrator to have their music files available on the file system.

At the moment, the only way of communication between clients and server is through the iOS and Android OS. No support is available to other platforms at the moment.

We deny any responsibility of setting up the local network and internet connection.

Installation diagram



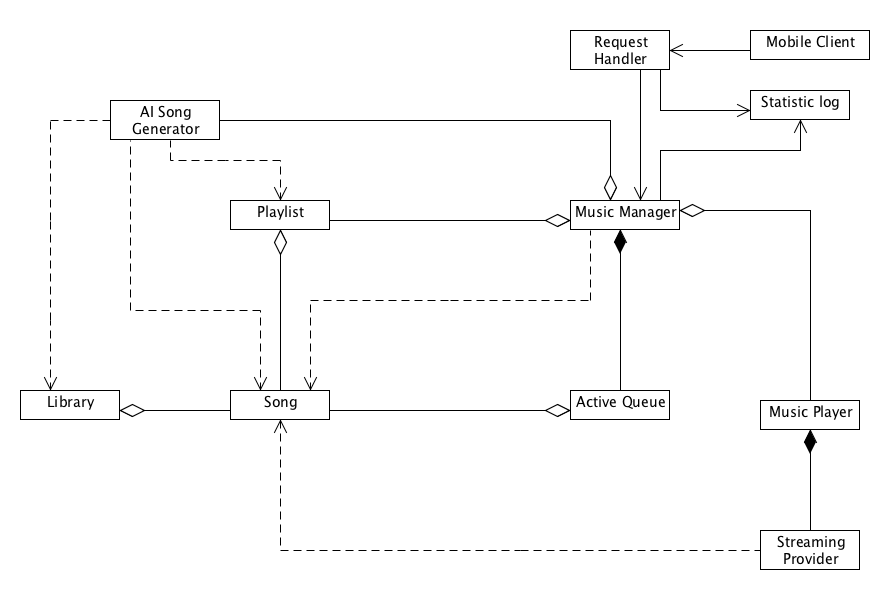
The installation diagram describes the communication between components in our system and external providers. The components include a Mobile Application, Request Handler, Music Manager, Database and a Streaming Service.

The Request Handler is responsible for handling connection and requests coming from the mobile application and record statistics in the database. The communication is done using sockets through TCP/IP.

Music Manager is responsible for managing playlists and playing songs requested by mobile devices or set by the administrator. The communication between the Request Handler and the Music Manager is done internally since they will be running in different threads. The Music Manager can also play songs from an external Streaming Service. The communication is done using HTTP.

Both Request Handler and Music Manager will come in the same installation package that will be available for download on our website. The Mobile Application will be available in their native application store (iOS – Apple Store, Android – Android Market Place).

Domain diagram



The domain diagram given describes the relationship between components in our system at a high-level. The components included in our system are the following: Song, Playlist, Statistic log, Library, Music Manager, Active Queue, Music Player, Streaming Provider, AI Song Generator, Request Handler, and Mobile Client.

A Song is representative of a song in our system. It contains information about the song including its location and whether it is a streamed song or local. Songs are used by several components of the system including the Library, Playlists, the Music Manager, the AI Song Generator, the Active Queue, and the Streaming Provider.

A Playlist is a set of Songs that can be generated by the AI Song Generator and read by the Music Manager. The Music Manager contains at least one Playlist at a time.

A Statistic log is a data object that is mean to contain logging information about Songs, such as when they are played and when they are requested, and about requests, such as how often they are made and who is making them. The Statistic logs are created by the Request Handler and the Music Manager.

The Library is the full collection of Songs in the system database. The database contains other forms of data than songs so the Library is meant to refer specifically to songs that exist in the database. The Library can be accessed by the AI Song Generator.

The Music Manager is a component that manages which music is currently playing in the Active Queue, which music will be played next according to a Playlist, and sends music to the Music Player. The Music Manager can take requests from the Request Handler for a particular Song to be played. It will then locate and retrieve the Songs that are to be played. When a Song is played, the Music Manager will log this activity with a Statistic log, which can later be viewed by the user. The Music Manager contains the Active Queue and the Playlist specifying which Songs are to be played. An AI Song Generator can also be used by the Music Manager to determine which songs to play. The Music Manager also contains and manages the Music Player.

The Active Queue is a small set of Songs that will be playing soon. Once a Song enters the Active Queue it will be played. The Active Queue is meant to ensure that there is always some knowledge of what is going to be played in the near future. The Music Manager manages the Active Queue.

The Music Player is responsible for playing the music. It will be managed in the Music Manger, which will give it the locations of Songs which it will either play directly if they are local or it will locate them via the Streaming Provider if not. If the music is to be streamed, the Music Player will create an instance of the Streaming Provider to search for and retrieve this required music.

The Streaming Provider provides a connection to an external music streaming service. Given a Song, the Streaming Provider instance will locate the stream for this song and give it to the Music Player to be played.

The AI Song Generator is a library of programs that is meant to return a song or set of songs that meet a specified set of constraints. This component is used by the Music Manger. The Music Manager will use an AI Song Generator in order to retrieve Songs to be queued in the current Playlist or to choose Songs from the current playlist to add to the Active Queue next. The AI Song Generator will be able to search through the Library of songs in order to find Songs that meet the requirements specified in the generation request.

The Request Handler is the component of the system which is listening for external Song requests. It will wait for a request from a Mobile Client, log the request with a Statistic log and then forward the request to the Music Manager.

Mobile Clients are the components of the system that are physically unattached to the rest of the system. They send requests for particular Songs to be played to the Request Handler via a network connection. The Request Handler can give the Mobile Clients information on what Songs are currently available through getting the current Playlist from the Music Manager.

**2.3 User Characteristics**

General Identification

The software will have two categories of end-users:

1. Administrators – the users at the service provider who will select the Active Playlist, select files for the database, and be able to view feedback and statistics. The main application will run on their server, an on-site computer.
2. Clients – the recipients of some service the administrators’ organization provides.

There are no stakeholders at this point.

User-specific goals, outcomes, and interface expectations

|  |  |  |  |
| --- | --- | --- | --- |
| End-Users | Goals | Outcomes | Interface Expectations |
| Administrators | The goal will be to increase client satisfaction at their sites, by allowing clients the opportunity to select their own background music. | The outcome should be that clients have an increased desire to come back to the service provider site, and depending on the particular service, that this feature will attract other clients. | The interface should be simple, easy to navigate, and fast (see *Preliminary Design Report, I iv. Success factors, p. 5*); it should add no additional stress to the administrator’s pre-existing tasks and responsibilities. |
| Clients | The goal is to be able to request a song(s) and have it played at the requested time, with a reasonable scheduling margin. | The outcome will be increased interaction between the client and their environment at the service provider site, as well as the unique experience of being able to choose background music. | The iOS and Android applications should be easy to navigate and control, simple, and visually attractive. |

User Validity

Administrator users must have the main Java application downloaded to their on-site computer, and they must provide or have access to a local network.

Client users must have either an Android or an iOS device, and at some point must have downloaded the application for their operating system.

User, Internal, and External Constraints

User constraints will be satisfied if the user-specific goals are met.

Internal constraints are imposed by module requirements or specific actors (see Section 3 for all Use Case Diagrams showing actors and their responsibilities):

1. Request Handler – the Request Handler is expecting textual feedback from the client applications and song requests. No other kind of input from the network will be accepted or processed. Invalid requests will be discarded and not saved.
2. Music Manager – the Music Manager can receive and process one request at a time from the Request Handler, and the same from the Active Queue. See Section 3 for detailed information on how these function and interact.
3. Active Queue – must contain at least 10 songs before the Play option may be selected. This is to ensure that clients have a reasonable list from which to choose. Playing without having this constraint met will display an error message. Streamed songs on the Queue also impose constraints; they must be streamed before played. The Queue anticipates this and asks to buffer songs in advance when possible.
4. Streaming Manager – receives streamed song references from the Queue (when trying to buffer in advance) and the Music Manager (when the user chooses songs to add to playlists) but is also limited by the website from which it streams.
5. AI Song Generator – imposes constraints on what options the user must specify in order to generate a list of songs. It also requires songs to exist in the database, whether referencing local files or streamed songs.

Externally, the website from which we will be streaming imposes constraints. We are limited to the songs in their selection, and no downloading can occur, only streaming. This means that each streamed song we refer to must be streamed every time it is played.

The network connection strength will determine the request and feedback processing speed, and the buffering of streaming songs, which will be done in advance when possible.

**2.4 Constraints**

General project constraints include the following:

Regulations: All streaming is carried out within legal requirements.

Network limitations: For full functionality to satisfy the user needs constraints, a local wireless network must be available on-site, and the connection must be consistent. The system’s Request Handler and Streaming Manager will need to deal with and adjust to slower networks; however the main application will adapt and still play background music even without all the complete functions.

The network security setting could also be an issue. If the setting is private and requires a password, the administrator must make the adjustments that allow, exceptionally, the client application to connect across the network.

Parallel operation: the main application on the server and many applications on smart devices must be able to execute at the same time. The main application’s Request Handler will manage receiving multiple requests simultaneously.

Reliability: Project release, even as a beta version, requires that the system be reliable within at least a 30% margin of the goals detailed previously. See *Preliminary Design Report, I iv. Success factors, p. 5.*

Time: the beta version of the project must be completed by the end of April 2012. See *Preliminary Design Report, IV, V Dated Activity Graph, p. 14.*

Money: Development should not cost the developing team anything other than time.

**2.5 Assumptions and Dependencies**

These are the elements of the project that, if changed, would affect the requirements. These could be things external to the project that could delay or restrict its development.

1. To be finished tomorrow….

**2.6 Apportioning of Requirements**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Title | Description | Delay Reason | When |
| 01 | Viewing Additional Information | The functionality that allows clients to view information about their service provider from their device application. | This is not necessary to the project, so we want to focus first on producing a good, working beta version before adding additional features. | After April 2012 |

**2.7 Risk Management**

2.6.1 Risk Assessment: Expounded ID Table

Risk location is identified by module or sub-modules (see Section 2.2 Modules).

**Priorities**

For identifying risk priorities, we have designated some terms. The priorities are determined based on three factors: (1) the seriousness of the problem it could cause the program based on the program’s output, (2) the impact to the client, and (3) our interest in the affected functionalities.

Low risk – 15%

Moderate risk – 50%

High risk – 75%

**Probabilities**

Likewise, since we cannot perform frequency experiments as this stage, we have assigned terms for probability values, with the given justification:

Very low – 5%. It is well within our abilities to prevent a failure from occurring, based on our pre-existing knowledge of the system and our available resources.

Low – 10%. With the research we have done to present, we have already done basic testing and prototypes to ensure that this functionality is feasible, and that a failure should not occur.

Medium – 30%. We possibly have done research and some basic testing, but there remain unknowns or assumptions we are making.

High – 50%. We have very little knowledge about this and consequently a failure is likely.

**Exposure**

Exposure is calculated as the product of the probability percentage and the loss. Since our project is not funded or paid for, our only loss is time; our loss estimates are made in working days.

ID nomenclature: N – network, D – development, S – scheduling, C – client.

|  |  |
| --- | --- |
| ID | 01N |
| Title | Network-related security |
| Description | Because our software is heavily dependent on a local network connection and sending messages across it, it is important that the database of songs and playlists on the local computer are secure, and that clients may only view the pre-selected playlist contents, without having access to the actual files. |
| Location | Database Module, Streaming Manager, Request Handler |
| Priority | High |
| Impact | A failure could result in legal problems, and the loss of client confidence in the software as a whole. |
| Probability | Very low |
| Exposure | 0.05\*10 = 0.5 |

|  |  |
| --- | --- |
| ID | 02D |
| Title | Development process risk: Streaming |
| Description | We are not very familiar with this area, and will need to choose some site(s) from which we can stream music, account for potential network problems and deal with them, and take care to respect all copyrights. The other possibility is that research reveals this to be impossible. |
| Location | Database Module, Streaming Manager, Request Handler |
| Priority | Moderate |
| Impact | The streaming functionality would be removed if we cannot resolve the above issues. |
| Probability | Medium |
| Exposure | 0.30\*8 = 2.4 |

|  |  |
| --- | --- |
| ID | 03D |
| Title | Development process risk: Graphic user interface |
| Description | Because we are using Java to ensure compatibility with multiple operating systems, our interface is limited to Java libraries. Creating an interface we are satisfied with, that is user-friendly and intuitive, could require research and/or more advanced designing. |
| Location | All modules with a user interface on the main application: Music Manager, Statistics, and Music Player. |
| Priority | Low |
| Impact | The interfaces would be less appealing than desirable. No functionality would be affected. |
| Probability | Low |
| Exposure | 0.10\*10 = 1.0 |

|  |  |
| --- | --- |
| ID | 04D |
| Title | Development process risk: Server to device communication |
| Description | Our smart device applications need to be able to locate and communicate with the server across the local network. Problems could include network strength, reliability, identification, and connection between device and server. |
| Location | Request Handler; but the whole system would be affected. |
| Priority | Moderate |
| Impact | Loss of a major functionality of our system – the ability to make requests across the network – and thus an unsuccessful project. |
| Probability | Low |
| Exposure | 0.10\*10 = 1.0 |

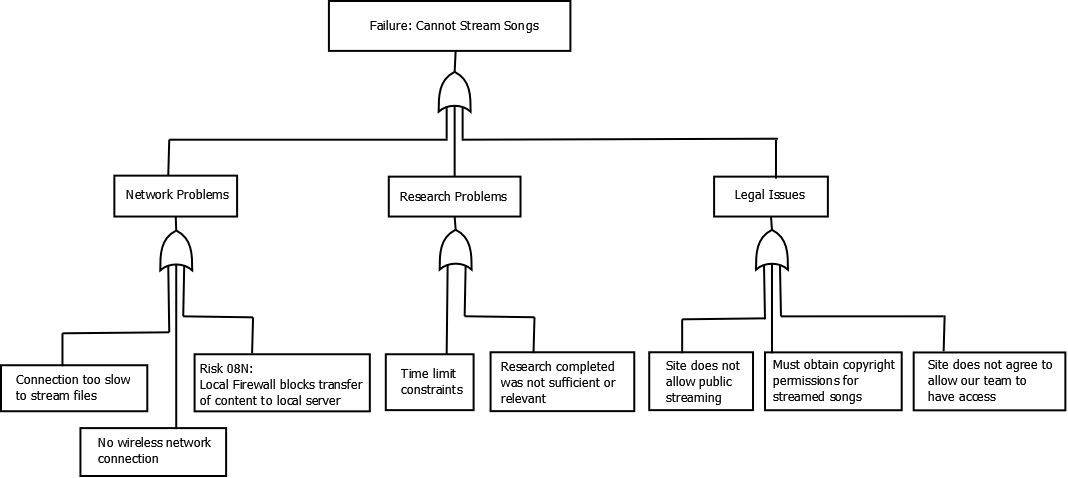
|  |  |
| --- | --- |
| ID | 05S |
| Title | Scheduling related issues; time constraints |
| Description | It is possible that our estimations of work and effort, and thus the corresponding portions of time allotted to work on phases were not feasible. We are working under several external deadlines, and multiple self-imposed internal ones. Getting too far behind could result in quality and functionality problems. |
| Location | Entire project. |
| Priority | Moderate |
| Impact | There would be increased stress on the developing team, and there could potentially be quality and functionality issues. |
| Probability | Medium |
| Exposure | 0.30\*10 = 3.0 |

|  |  |
| --- | --- |
| ID | 06D |
| Title | Development process risk: Library limitations |
| Description | Based on the libraries we choose, development could be inhibited later on in the process. We could also find that our assumptions about multi-platform compatibility for certain libraries are not correct. |
| Location | Music Manager, Music Player, Statistics Manager. |
| Priority | Moderate |
| Impact | We would need to re-evaluate our options and decide on new strategies or workarounds for problems which cause functionality issues. |
| Probability | Medium |
| Exposure | 0.3\*15 = 4.5 |

|  |  |
| --- | --- |
| ID | 07C |
| Title | Client feedback |
| Description | After our initial stage of testing with clients, feedback might indicate that we did not correctly gage user needs and opinions. Time permitting, this would require us to loop back to development stages and make adjustments. |
| Location | This could affect the entire project. |
| Priority | Low |
| Impact | This would affect the release date. |
| Probability | Low |
| Exposure | 0.1\*5 = 0.5 |

|  |  |
| --- | --- |
| ID | 08N |
| Title | Local Firewall |
| Description | The administrator’s on-site computer may have a local firewall which blocks external access to ports. This would prevent the client applications from connecting with the socket even if the network is functional. |
| Location | Request Handler; but the entire system would be affected. |
| Priority | High |
| Impact | This would not allow our system to have any of the functionalities involving network connection. |
| Probability | Medium |
| Exposure | 0.3\*5 = 1.5 |

2.6.2 Risk Analysis: Fault Tree for 02D Development process risk: Streaming



* + 1. Risk Solution Table: Action, Leverage, Control Summary

To calculate leverage, the exposure after the control risk management steps is calculated. As with the previous exposure calculations, exposure is the product of probability and loss in working days.

Leverage = (risk exposure – exposure after control reduction)/cost of reduction.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Title | Type of Action | Brief Control Summary | Cost of Reduction (working days) | Leverage |
| 01N | Network-related security | Assume | We will assume this risk since it is crucial to our functionality. We will focus on this risk while structuring our database, and run unit testing specifically to make sure it is secure. | 10 – extensive unit testing | 0.04 |
| 02D | Development process risk:  Streaming | Assume | The risk will be assumed; extra time has been designated before the third milestone research streaming API’s. We will also anticipate the possibility of needing to cut it out, and will prepare other options. | 3 – research period | 0.67 |
| 03D | Development process risk:  Graphic user interface | Assume | Special time has been designated for the fourth milestone to develop UIs. There is also time at the end during final integration and testing where interface polishing can be done. | 5 – final integration and testing period | 0.10 |
| 04D | Development process risk:  Server to device communication | Assume | This functionality is essential so we will assume this risk, and try to create simple prototypes during the research phase of our project to anticipate these types of problems. | 5 – research period | 0.59 |
| 05S | Scheduling related issues; time constraints | Assume | At this point, we can anticipate only by allowing buffer time for our milestones, which may potentially cut into time we designated for vacation, exam period, etc. We will learn from experience as the project progresses, and if needed will make changes to our activity graph accordingly. | 10 – total buffer time | 0.20 |
| 06D | Development process risk:  Library limitations | Assume | We will do our library-related research carefully before the third milestone, but will also need to use buffer time to make changes if large problems arise. | 5 – total buffer time | 0.40 |
| 07C | Client feedback | Assume | This risk must be assumed; we plan to complete our project as designed for the purpose of the course, and then make extensions to our timeline before releasing it to the public. | N – we are free to specify the release date | Very minimal – close to zero |
| 08N | Local Firewall | Transfer | The computer’s administrator must make changes to the local firewall to allow our software access to a port. | 0 | n/a – transferred risk |

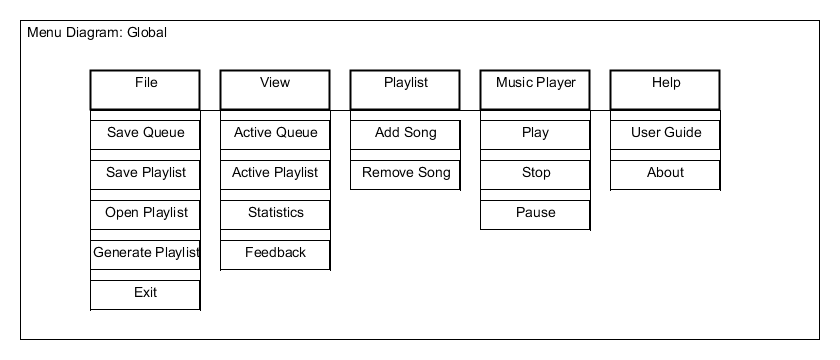
In summary, all risks will be assumed except for the Risk 08N for which we deny responsibility. For most risks, our control method involves allowing extra time to account for extra research, testing, and experimentation as necessary. While we do allow buffer time in our scheduling, scheduling itself is also a risk; minimizing this requires us to learn from experience and assume that our rough estimations are reasonable.

III. Specific Requirements

The purpose of this section is to give a global view of the program from two aspects of the user perspective. Firstly, the menu diagram presents the actual menu options visible to the user, with their submenus and options, and a brief description of the functionality of each option. Secondly, the global data-flow diagram demonstrates how data flows through the system. The three data flows of interest to users have been chosen: when a client makes a request and it is added to the Active Queue to be played, when a client has submitted some feedback, and when a streamed song is on the Active Queue.

**3.1 Global Program View**

Menu diagram and brief description of elements



**File**

Save Queue – Saves the current Active Queue.

Save Playlist – Saves the current Active Playlist.

Open Playlist – Displays a window allowing user to choose a new saved Playlist to open.

Once selected, the new Playlist replaces the current Active Playlist.

Generate Playlist – Displays a window allowing the user to specify options for generating

a new Playlist.

Exit – Exits the program.

**View**

Active Queue – Switches to the Queue View, where the Active Queue is displayed. See

Section 3.2.1.1

Active Playlist – Switches to the Playlist View, where the Active Playlist is displayed. See

Section 3.2.1.2.

Statistics – Displays a window allowing the user to select options and view the song

request statistics. See Section 3.2.7.

Feedback – Displays a window with the output of the compiled feedback file from the

database. See Section 3.2.7.

**Playlist**

Add Song – Displays a window allowing the user to browse locally and online (among the

songs able to be streamed) and choose a song or multiple to be added to the current Active Playlist.

Remove Song – Switches to the Playlist View and allows the user to choose which song to

be removed from that active list.

**Music Player**

Play – Plays the next song on the Active Queue; available when not playing.

Pause – Pauses the song being played from the Active Queue; available when not paused.

Stop – Stops the song being played from the Active Queue; available when not stopped.

**Help**

User Guide – Displays a window showing the basic instructions for operating the

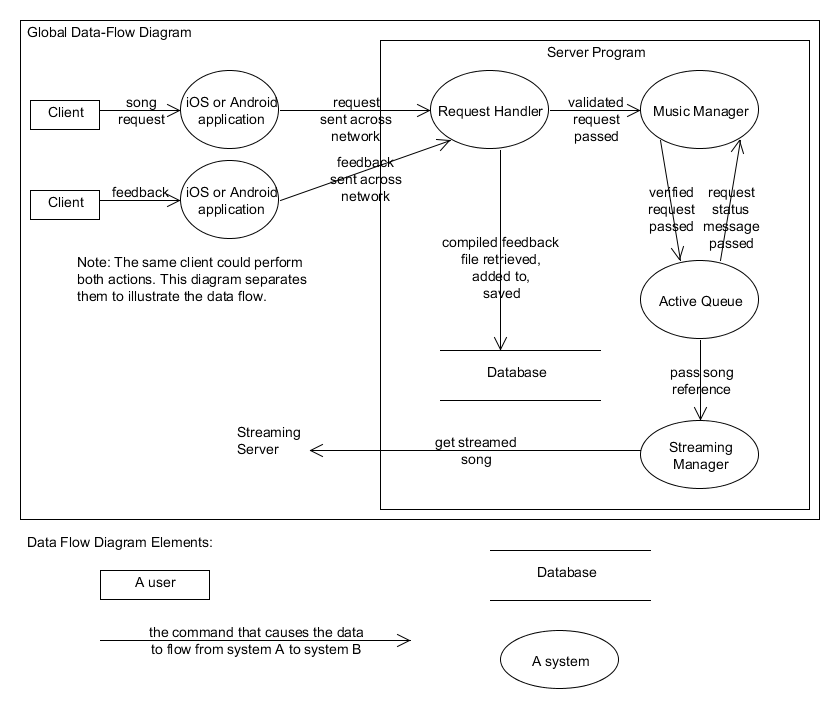
program. See Section 3.2.1.3.

About – Displays a window showing information about the software and development team.

See Section 3.2.1.3.

Global data-flow diagram

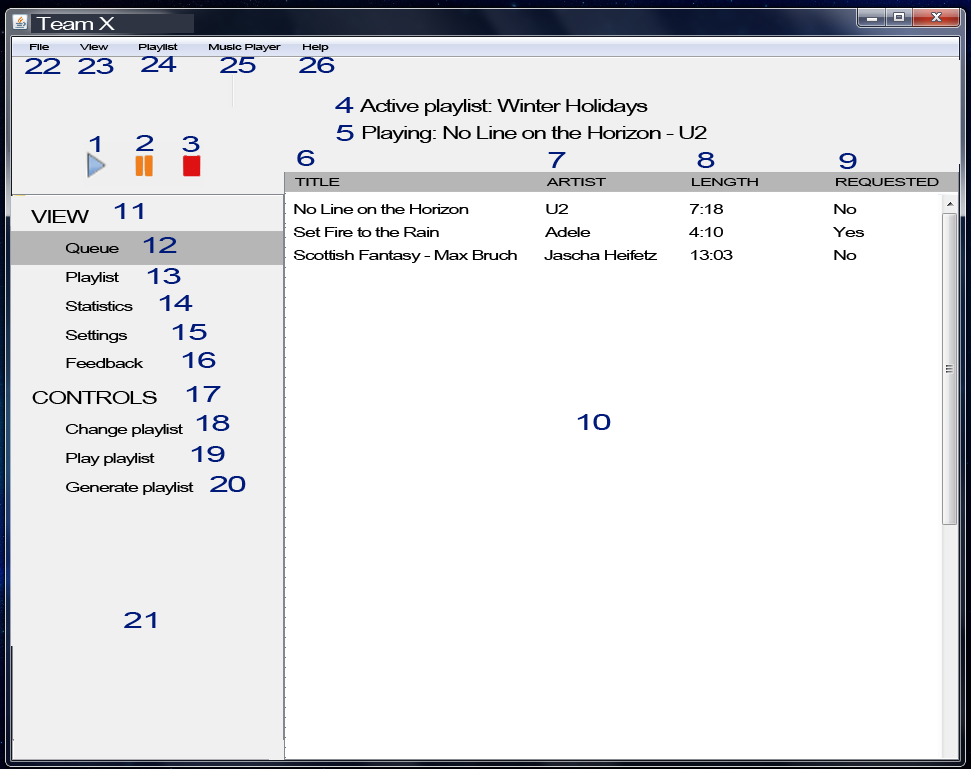
The data flow diagram shows the flow of data – song requests, feedback, and streamed song references – throughout the system.



**3.2 Logical Unified Software Breakdown**

3.2.1: Music Manager

**3.2.1.1 Screen Layouts: 1 – Queue View**



1: Play option – plays the first song on the Active Queue. See Use Case *Play song*, Section 3.2.2

2: Pause option – pauses the song that is currently playing. See Use Case *Pause song.* 3.2.2

3: Stop option – stops the song that is currently playing or paused. See Use Case *Stop song.* 3.2.2

4: Displays the active playlist.

5: Displays the title and artist of the song currently playing from the Active Queue.

6: The column for the title of each song on the Active Queue.

7: The column for the artist of each song on the Active Queue.

8: The column for the length in minutes and seconds for each song on the Active Queue.

9. The column displaying whether each song on the Active Queue has been client requested or not.

10: Main panel - the main panel of the UI which displays the Active Queue in the Queue View.

11: View - the first category of options on the side panel.

12: Queue option - when selected, this displays the Active Playlist in the main panel.

13: Playlist option - when selected, this displays the Active Playlist in the main panel. See Section – 3.2.1.2.

14: Statistics option – when selected, a pop-up window appears, displaying options for viewing song request statistics and generated reports. See Section 3.2.7.

15: Settings option – when selected, a pop-up window appears, displaying the Settings. See Section 3.2.1.

16: Feedback option – when selected, a pop-up window appears, displaying the contents of the compiled feedback file stored from the database. See Section 3.2.7.

17: Controls – the second category of options on the side panel.

18: Change playlist – when selected, displays a pop-up window which allows the user to change the playlist. See Use Case *Choose new playlist* Section 3.2.1.

19: Play playlist – when selected, this displays the Playlist View and begins playing the contents of the current Active Playlist. The Active Queue is saved but ignored, and no song requests are respected. A confirmation pop-up window appears before playing is begun, warning the user that song requests will not be received with that option.

20: Generate playlist – when selected, a pop-up window appears allowing the user to specify options to generate a new playlist. See Use Case *Generate a playlist* Section 3.2.1*.*

21: Side panel – contains the View and Controls option categories.

22: File drop-down menu – displays sub-menu options related to file management. See 3.1 Global View: Menu Diagram.

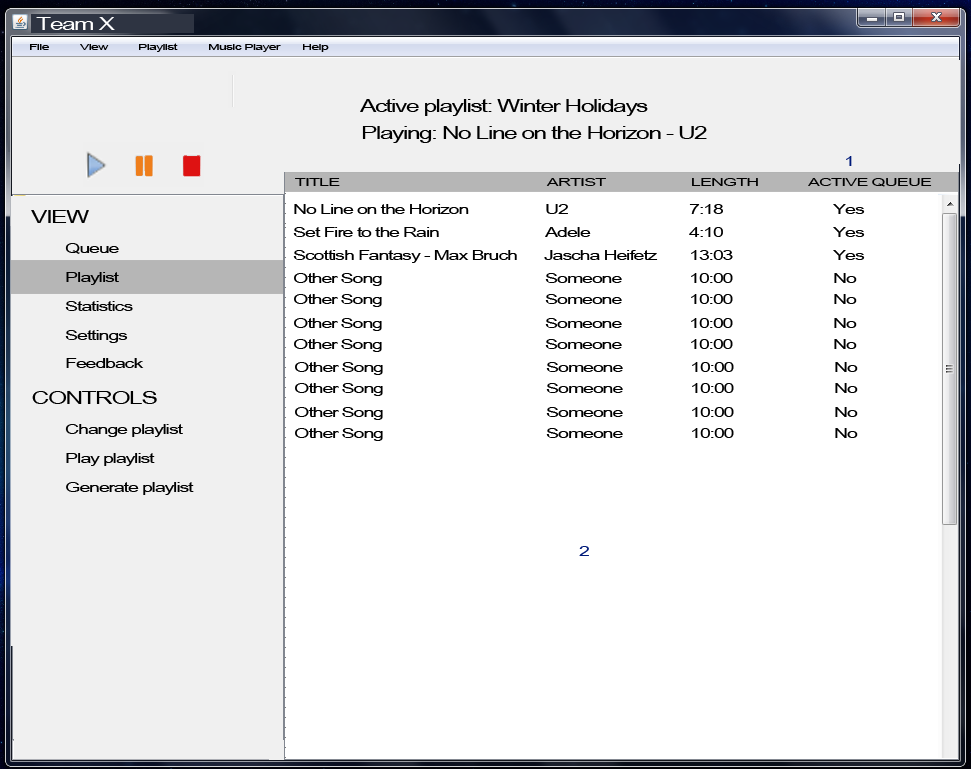
23: View drop-down menu – displays sub-menu options related to the view. See 3.1 Global View: Menu Diagram.

24: Playlist drop-down menu – displays sub-menu options related to the playlist. See 3.1 Global View: Menu Diagram.

25: Music Player drop-down menu – displays sub-menu options related to Music Player commands. See 3.1 Global View: Menu Diagram.

26: Help drop-down menu – displays sub-menu options User’s Guide and About. See 3.1 Global View: Menu Diagram.

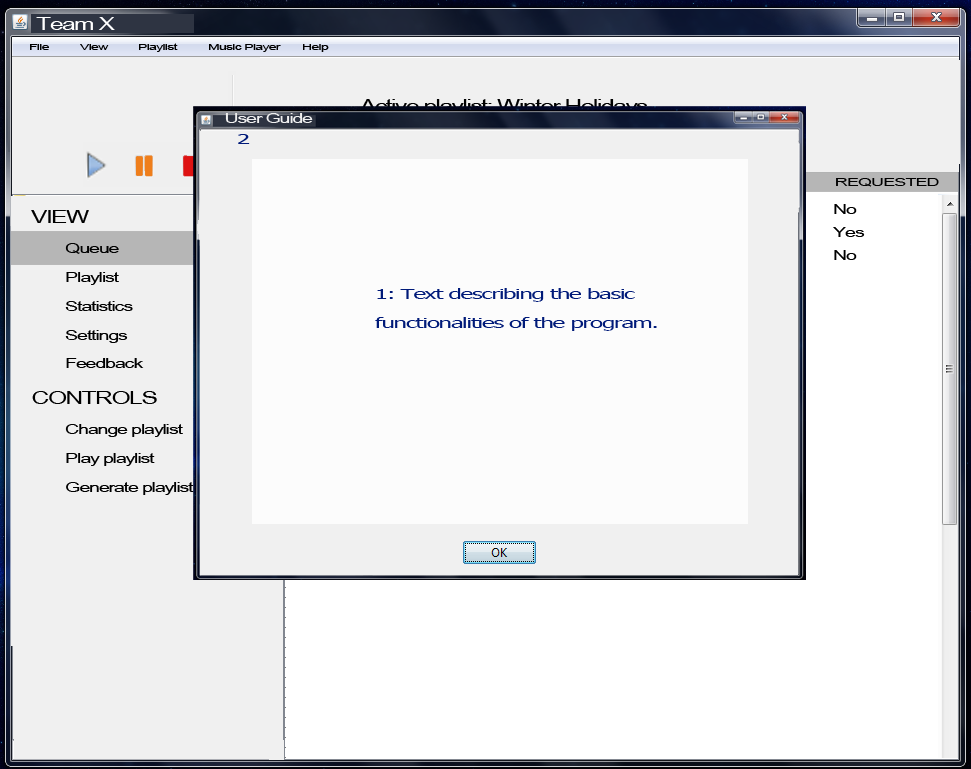
**3.2.1.2 Screen Layouts: 2 – Playlist View**

****

1: The column for displaying whether each song on the playlist is also on the Active Queue.

2: The main panel, which now displays the Active Playlist.

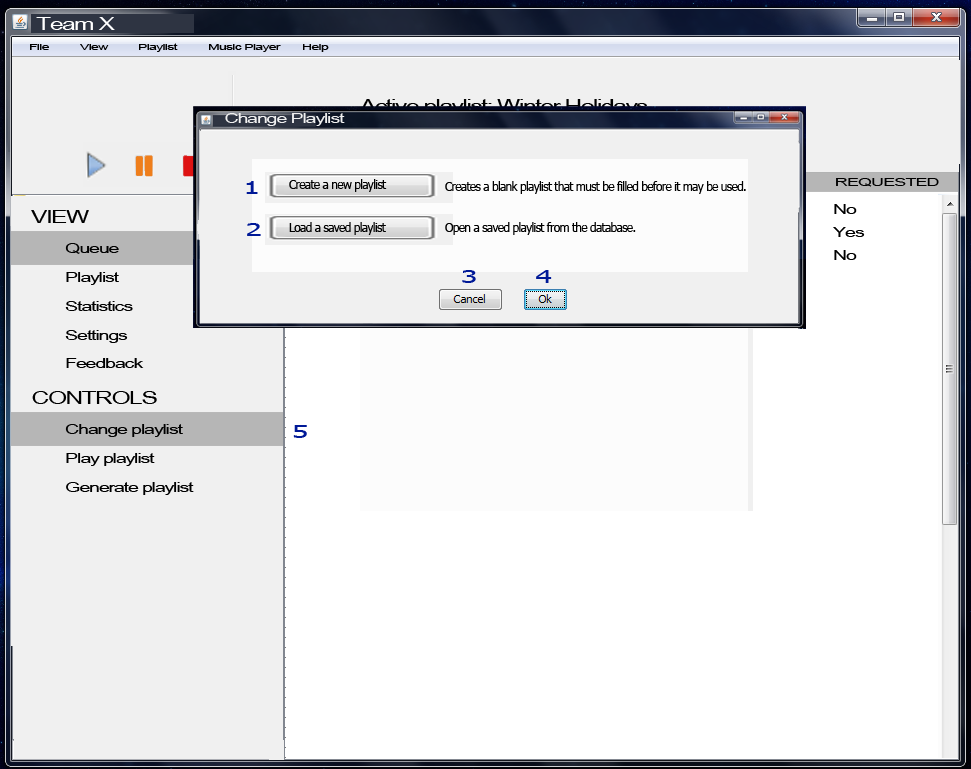
**3.2.1.3 Screen Layouts: 3 – Help Sub-options**

****

1: Text – for the second Help sub-option, About, this will have text describing the software briefly, the date of release, and brief team information.

2: User Guide - for the second Help sub-option, About, this will be replaced by the text “About”.

**3.2.1.4 Screen Layouts 4: Controls – Change Playlist option**

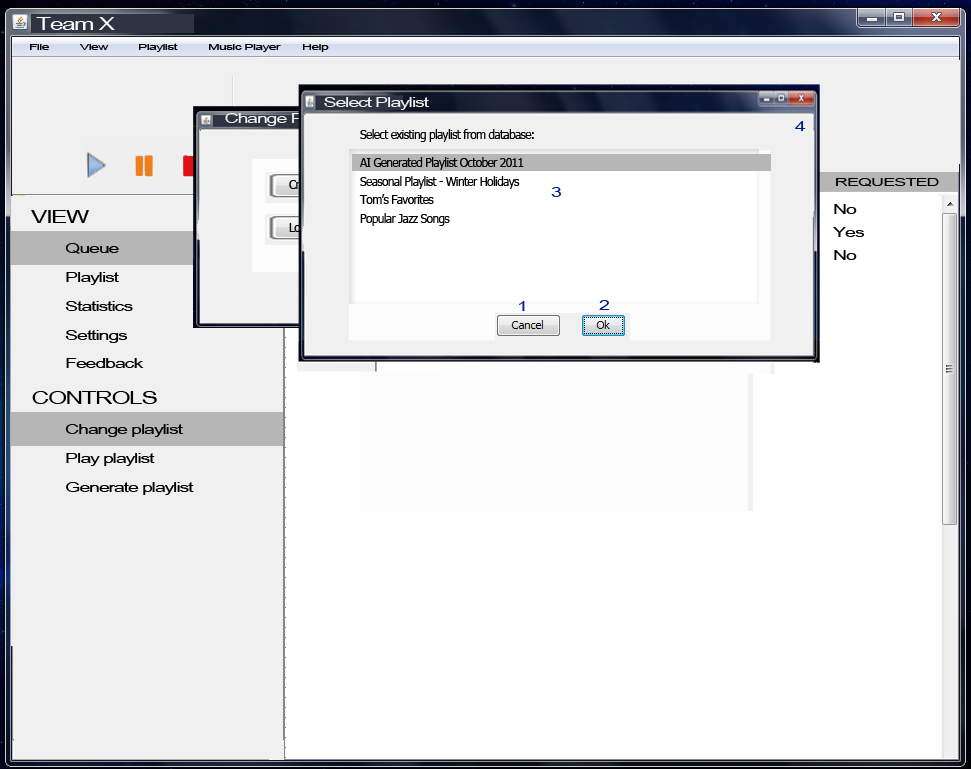


1: Create a new playlist – selecting this creates a blank playlist that must have at least 10 songs in it before it may be used. See the following screen layouts.

2: Load a saved playlist – this triggers a second pop-up window to appear, displaying the list of saved playlists in the database. See the following screen layouts.

3: Cancel – closes the window without making any changes.

4: Ok – If a saved playlist has been loaded, then this option replaces the current Active Playlist with the one selected. Otherwise, it does the same action as Cancel.

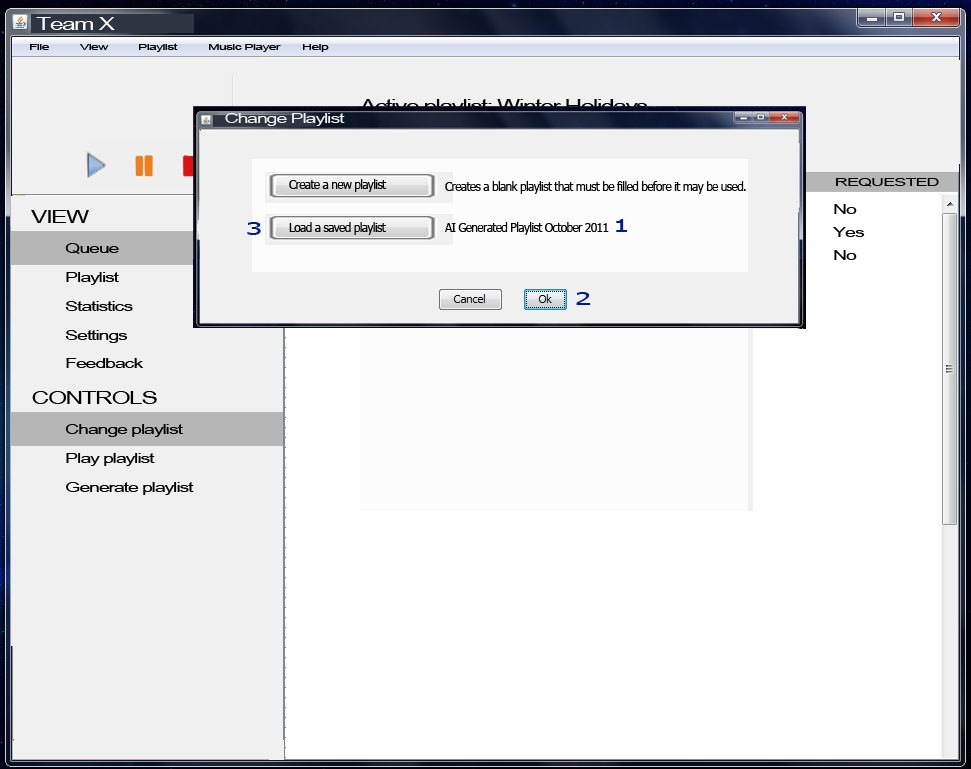


1: Cancel – this selection closes the window and returns the user to the Change Playlist window.

2: Ok – this selection chooses the highlighted playlist and returns to the Change Playlist window; see the following screen layout.

3: This area displays the list of the saved playlists in the database from which the user can choose.

4: Clicking Close returns the user to the Change Playlist window, identical to Cancel.

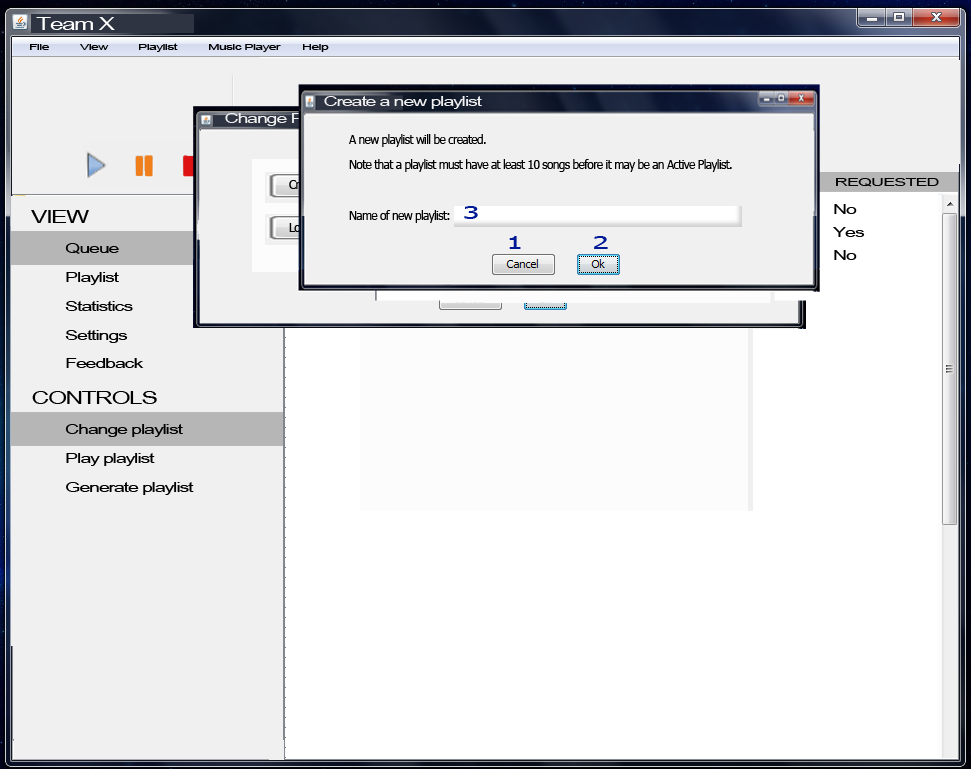


A saved playlist has been selected and the user is returned to the Change Playlist window.

1: This now displays the selected saved playlist.

2: Ok – selection loads the displayed playlist as the Active Playlist, replacing the current one.

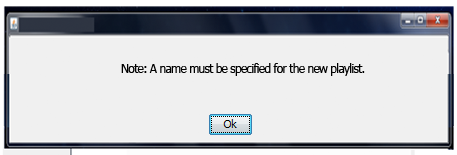
3: Load a saved playlist – this functionality remains the same as before.



1: Cancel – closes the window and returns the user to the Change Playlist window.

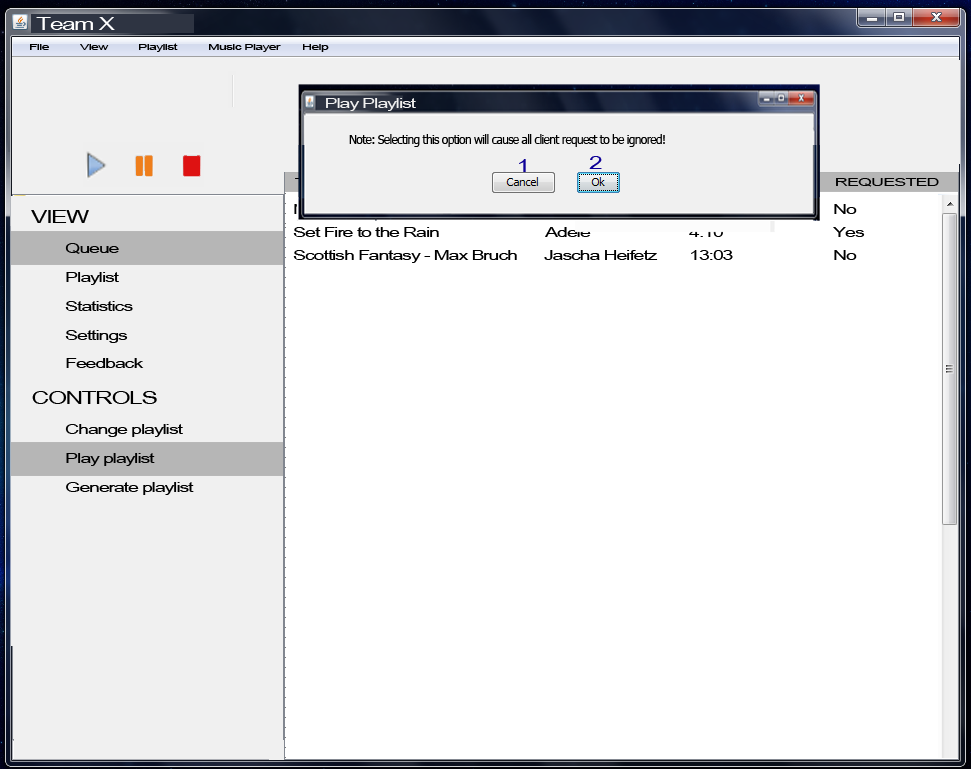
2: Ok – this closes both the sub-windows and displays the Playlist View of the main screen. After this, if there are fewer than 10 songs added to that list, an error message will show up if the Play option is selected.

3: Name field – here the user must enter the name of the new playlist. If there is no name, then another window is displayed:



Selecting Ok returns the user back to the window shown above for creating a new playlist.

**3.2.1.5 Screen Layouts 5: Controls – Play Playlist option**

****

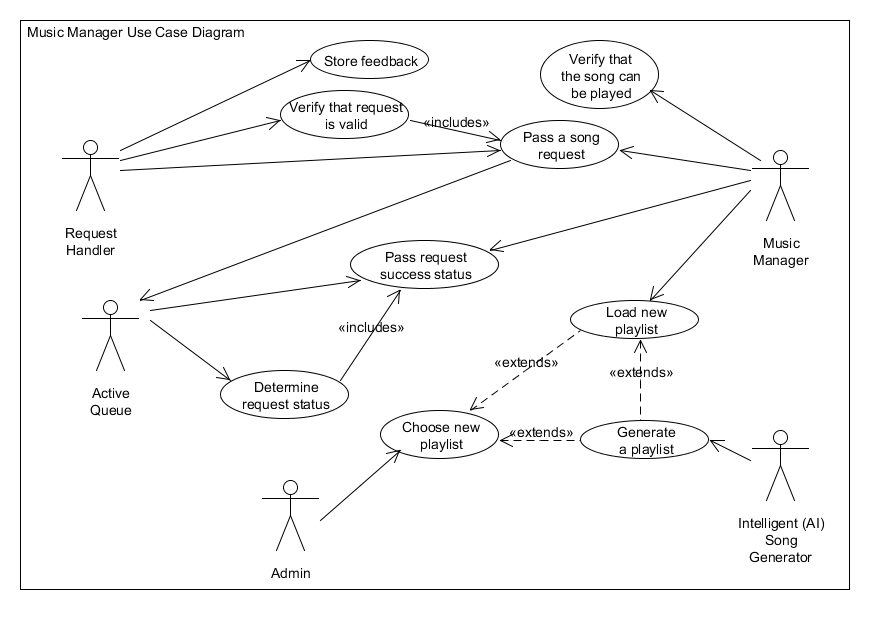
1: Cancel – closes window and returns user to the current main view. The Play Playlist option under Controls is un-highlighted.

2: Ok – plays the current Active Playlist.

**3.2.1.6 Screen Layouts 6: Settings Option**

The Settings option under VIEW will be highlighted, as in 3.2.1.1 except with Settings instead of Queue. The following pop-up window will be displayed:

**3.2.1.7 Screen Layouts 7: Music Player menu options**



**Module Use Case Templates**

|  |  |
| --- | --- |
| Use Case Name: | Pass a song request |
| Iteration: | Focused |
| Summary: | The client has made a request for a song, and this request is passed from the Request Handler to the Music Manager, which subsequently passes it to the Active Queue to see if it is valid. |
| Basic Course of Events: | 1. The use case begins when one actor – the Request Handler or the Music Manager – receives a song request message. 2. The receiver passes the request to the next recipient; either the Music Manager or the Active Queue. |
| Alternative Paths: | None |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | A client using the iOS or Android application has made a song request, which has successfully been transmitted across the network and received by the Request Handler, OR a song request has been received by the Music Manager from the Request Handler. |
| Assumptions: | The song requested is one that exists in the current active Playlist.  The song requested is playable. |
| Preconditions: | The assumptions hold true; the song exists in the current active Playlist and if the Request Handler is the receiver, the song has been verified playable. |
| Post conditions: | The success status of the request has been sent back to the Request Handler after being processed by the final recipient, the Active Queue. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 20, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Verify that request is valid |
| Iteration: | Focused |
| Summary: | The Request Handler has received a song request and needs to verify that it is valid before passing on the request. |
| Basic Course of Events: | 1. The use case begins when the Request Handler receives a song request message from an iOS or Android application. 2. The Request Handler checks that the song is valid by accessing the song reference in the database, and verifying that the song exists. 3. The Request Handler passes the request to the Music Manager. |
| Alternative Paths: | 3. If the song returns that it is not playable, then the Request Handler does not pass the request. Instead, it generates an error message and returns it to the original sending application. |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | The Request Handler has successfully received a song request message from an application. |
| Assumptions: | The song request corresponds to an actual client request from an application. |
| Preconditions: | The song request is in the expected format. |
| Post conditions: | The request has been dealt with; either sent on to the Music Manager or an error message returned. Consequently, a success status message has been sent back to the original client application – success or error. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 20, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Load new Playlist |
| Iteration: | Focused |
| Summary: | The Music Manager loads a new Playlist (specified by the user and requested and returned by the database or an empty one), replacing the current one. |
| Basic Course of Events: | 1. The user selects the New Playlist or Open Playlist option from the File menu, or the Change Playlist or Generate Playlist from the Controls sidebar. See Section 3.1 for the menu diagram. 2. The Music Manager is returned the new Playlist, either an empty one or one returned by the database. 3. The Music Manager saves the current Playlist to the database and sets its Active Playlist as the new Playlist returned. |
| Alternative Paths: | 2. If Generate Playlist is selected, the AI Song Generator is involved and must process the user’s selected options. See Use Case *Generate Playlist.* |
| Exception Paths: | None |
| Extension Paths: | *Generate Playlist* |
| Triggers: | User selects an option to either open a new Playlist or open a saved one. |
| Assumptions: | The Playlist returned to the Music Manager is a valid list of songs that exist in the Database. |
| Preconditions: | None |
| Post conditions: | The old Active Playlist has been saved into the database. The Active Playlist has been replaced by the user’s selection, with the exception of the case where the user has canceled the process or the new Playlist was the Active Playlist. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 20, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Generate Playlist |
| Iteration: | Focused |
| Summary: | The user has asked to load a new Playlist, and has chosen the Generate Playlist option. The specific options for generation have been sent to the AI Song Generator, where it is processed. The generated Playlist is then returned to the Music Manager. |
| Basic Course of Events: | 1. The user selects Generate Playlist from the Controls sidebar, Change Playlist, or New Playlist option. 2. Options are displayed to the user in a new window. These indicate the traits or criteria by which the AI Song Generator will produce the new list. 3. The user clicks “Generate” to generate the playlist. The selected options get sent to the AI Song Generator. 4. The AI Song Generator, using various artificial intelligence algorithms, creates the Playlist by processing and selecting songs from the database. 5. The current Active Playlist is saved. 6. The returned Playlist is set as the Active Playlist. |
| Alternative Paths: | 3. The user selects “Cancel” instead of “Generate”, canceling the process and returning the user to the main display.  3. The user has not selected any options. An error message appears and step 2 is repeated.  3. The user has not selected valid options, or not all required options are specified. An error message appears and step 2 is repeated.  4. There was an error generating the intelligent Playlist. An error message appears and the user is returned to the main display. |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | The user selects Generate Playlist from the Controls sidebar, Change Playlist, or New Playlist option. |
| Assumptions: | There are song references in the database from which the AI Song Generator can choose. These songs have the necessary fields which can be examined.  The AI Song Generator has generated correctly a playlist which corresponds to the user’s specified options. |
| Preconditions: | None (Alternative paths may occur if the Assumptions are not true.) |
| Post conditions: | At the end of the normal course of events, the original Active Playlist has been replaced by the generated one, which has been set as the new Active Playlist.  The original Active Playlist has been saved to the database. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 24, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Verify that the song can be played |
| Iteration: | Focused |
| Summary: | The Music Manager has been passed a request from the Request Handler, which checks that the song is playable. If so, it passes the request to the Active Queue to determine the status. See Use Case *Determine Request Status.* |
| Basic Course of Events: | 1. The Request Handler has passed a song request, already validated (see Use Case *Verify that request is valid)* to the Music Manager. 2. The Music Manager checks that the song is playable, by accessing the isPlayable Boolean variable in the Song object. This variable keeps track of whether the song is on the Active Playlist, and if it has exceeded the number of allowed plays for that day. 3. The Music Manager passes the request to the Active Queue and waits for the returned success status. |
| Alternative Paths: | 2. The isPlayable variable returns false; in this case, the Music Manager returns a failure message to the Request Handler and discards the request. |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | The Request Handler has received and validated a request, and sends it successfully to the Music Manager. |
| Assumptions: | The song request sent from the Request Handler is valid; the song exists in the database. See Use Case *Verify that request is valid.* |
| Preconditions: | None |
| Post conditions: | The Music Manager waits until it receives a success status message from the Active Queue. It then will pass this message back to the Request Handler. See Use Case *Pass request success status.* |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 24, 2011 – Focused |

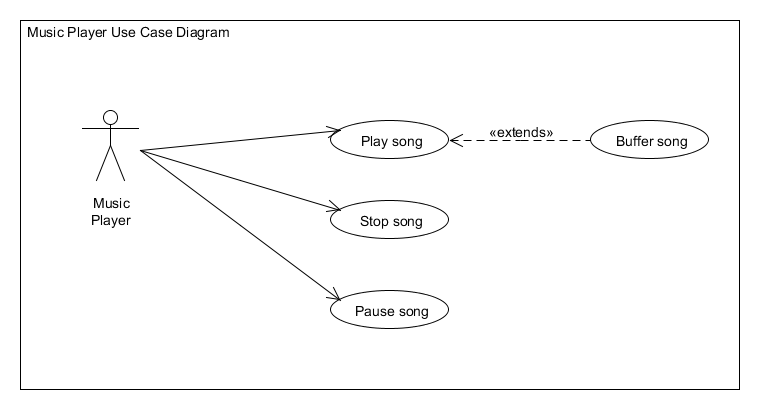
|  |  |
| --- | --- |
| Use Case Name: | Pass request success status |
| Iteration: | Focused |
| Summary: | The Active Queue or the Music Manager have a success status message, and they pass it on to the Music Manager or the Request Handler respectively. |
| Basic Course of Events: | 1. The Active Queue has either added the requested song or has determined that it cannot be added. 2. The Active Queue generates a success status message. 3. The Active Queue sends this message to the Music Manager. 4. The Music Manager sends this message to the Request Handler.   Note: Steps 3 and 4 are identical; this Use Case covers both. |
| Alternative Paths: | None |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | The Active Queue has determined the success status of a song request. |
| Assumptions: | The success status message that is determined is correct, and if the message is a “success” then the request has been added appropriately to the Active Queue. |
| Preconditions: | The success status has been determined. See Use Case *Determine request status.* |
| Post conditions: | The success status message will reach its destination – the client application from which the Request Handler received the request. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | October 28, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Determine success status |
| Iteration: | Focused |
| Summary: | The Active Queue has received a song request from the Music Manager. It processes it and determines whether or not in can be added. If it can, it adds it and returns *succeeded* status message to the Music Manager. Otherwise, it returns a *failed* status message back. |
| Basic Course of Events: | 1. The Active Queue receives a song request from the Music Manager. 2. The Active Queue checks the request time to verify whether it is a “play as soon as possible” or a specific time. 3. For a specific time request, the Active Queue checks the song scheduled to play at that time interval. 4. If there is no song scheduled then, it inserts the requested song there. 5. The success status for that request is set to *succeeded.* |
| Alternative Paths: | [3a. For a “play as soon as possible” request, the Active Queue inserts the song after the song following the one currently playing. (The song currently playing will terminate, then the next song in line will play, then the requested song will play.) Return to 5 of Basic Course of Events.  3b. If the end of the day’s playing period after the current playing song or the next one, then the success status for the request is set to *failed.*]  [4. If there is a song already scheduled then, it checks if it is a client requested song.  5. If the song is client requested, then it searches for a free interval (unreserved or filled with a non-client requested song) within 15 minutes of the requested time.  6a. If found, it inserts the song there. Return to 5 of Basic Course of Events.  6b. If not found, the success status for that request is set to *failed.*] |
| Exception Paths: | None |
| Extension Paths: | None |
| Triggers: | The Active Queue receives a song request from the Music Manager. |
| Assumptions: | The song request the Active Queue receives is valid and is playable; it exists in the Active Playlist and has not exceeded the maximum number of plays per day. |
| Preconditions: | The Active Queue has received a song request from the Music Manager. |
| Post conditions: | The Active Queue has correctly determined the success status of the request and will generate a message which will be passed to the Music Manager. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Choose new playlist |
| Iteration: | Focused |
| Summary: | The administrator (user of the main application) chooses a new playlist, either one generated by the AI Song Generator, a saved one from the database, or a new empty one. |
| Basic Course of Events: | 1. The user chooses the Open Playlist option from the File menu, or the Change Playlist option from the Controls side panel. 2. A window displays options. The user may either select to generate one, to open a saved one, or to open a new empty one. |
| Alternative Paths: | 2. The user selects “Cancel” to exit the process of choosing a new playlist. |
| Exception Paths: | None |
| Extension Paths: | 1. If the user selects an AI Song Generator, Use Case *Generate a playlist* is executed. This may lead to Use Case *Load new playlist.* 2. If the user selects a saved playlist in the database which is not the current Active Playlist, the Use Case *Load new playlist* is executed. |
| Triggers: | The user clicks an option which opens the window displaying options for new playlists. |
| Assumptions: | None |
| Preconditions: | None |
| Post conditions: | None |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Store feedback |
| Iteration: | Focused |
| Summary: | The Request Handler receives some feedback from a client application, and stores it in the compiled feedback file from the database. |
| Basic Course of Events: | 1. A client sends some textual feedback through the application. 2. The Request Handler receives the request. 3. The Request Handler retrieves the compiled feedback file from the database and opens it. |
| Alternative Paths: | None |
| Exception Paths: | 2.The feedback is sent but never received by the Request Handler.  3a. The compiled feedback file is not found in the database; it either has been deleted or damaged and is not retrievable.  3b. There is some input or output error with the file.  All exception paths result in the feedback not being stored in the file. |
| Extension Paths: | None |
| Triggers: | The Request Handler receives a feedback message sent from a client application. |
| Assumptions: | The feedback message sent comes from a client located on-site. |
| Preconditions: | A feedback message has been received by the Request Handler. |
| Post conditions: | The feedback message has been stored into the compiled feedback file in the database. The pre-existing contents of that file have not been altered. The user, upon selecting View Feedback, may view the most recent message sent. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

3.2.2: Music Player



|  |  |
| --- | --- |
| Use Case Name: | Play song |
| Iteration: | Focused |
| Summary: | The Music Player calls the methods which cause the first song on the Active Queue to be played. |
| Basic Course of Events: | 1. The user clicks the Play Song button from the main display, or from the Music Player menu option. 2. The Music Player identifies the first song on the Active Queue. 3. The Music Player calls the methods to play that song. |
| Alternative Paths: | None |
| Exception Paths: | If the song is already playing, clicking it again performs no action. |
| Extension Paths: | If the first song on the Active Playlist is a streamed song, it needs to be buffered and loaded before it can be played. |
| Triggers: | The user clicks an option asking for the Music Player to play. |
| Assumptions: | There is at least one song on the Active Queue. |
| Preconditions: | The song is not currently playing; it may be either stopped or paused. |
| Post conditions: | The first song on the Active Queue is playing. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

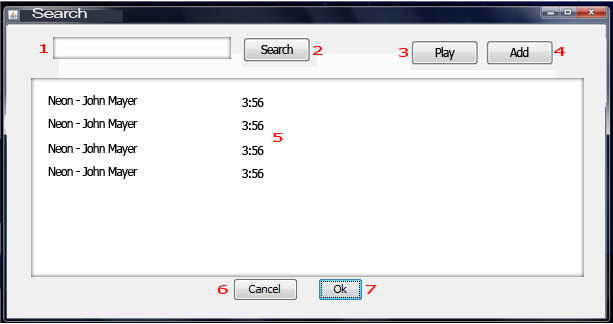
|  |  |
| --- | --- |
| Use Case Name: | Stop song |
| Iteration: | Focused |
| Summary: | The Music Player calls the methods which cause the first song on the Active Queue to be stopped (the song that is currently playing or paused). |
| Basic Course of Events: | 1. The user clicks the Stop Song button from the main display, or from the Music Player menu option. 2. The Music Player identifies the first song on the Active Queue. 3. The Music Player calls the methods to stop that song. |
| Alternative Paths: | None |
| Exception Paths: | If the song is already stopped, clicking again performs no action. |
| Extension Paths: | None |
| Triggers: | The user clicks an option asking for the Music Player to stop. |
| Assumptions: | There is at least one song on the Active Queue. |
| Preconditions: | The song is currently playing or paused. |
| Post conditions: | The song has been stopped. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Pause song |
| Iteration: | Focused |
| Summary: | The Music Player calls the methods which cause the first song on the Active Queue to be paused (the song that is currently playing). |
| Basic Course of Events: | 1. The user clicks the Pause Song button from the main display, or from the Music Player menu option. 2. The Music Player identifies the first song on the Active Queue. 3. The Music Player calls the methods to pause that song. |
| Alternative Paths: | None |
| Exception Paths: | If the song is already paused, clicking again performs no action. |
| Extension Paths: | None |
| Triggers: | The user clicks an option asking for the Music Player to pause the song. |
| Assumptions: | There is at least one song on the Active Queue. |
| Preconditions: | The song is currently playing. |
| Post conditions: | The song has been paused. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 – Focused |

|  |  |
| --- | --- |
| Use Case Name: | Buffer song |
| Iteration: | Focused |
| Summary: | A streamed song is on the Active Queue and needs to be buffered before it can be played. |
| Basic Course of Events: | 1. The Music Player detects a song in the first *n* (see Triggers) songs on the Active Queue which is a streamed song. 2. The Music Player, without disrupting other actions (playing the current song), sends the song’s reference to the Streaming Module. 3. The song is buffered. 4. The song is now ready to be played. |
| Alternative Paths: | None |
| Exception Paths: | The reference to the streamed song is invalid. In this case, the song cannot be streamed and is removed from the Active Queue. |
| Extension Paths: | None |
| Triggers: | Either the first song on the Active Queue is a streamed song, or some song in the first *n* songs on the list is a streamed song. (*n* will be determined later after testing is performed to find out the time required to buffer across a network of reasonable connection strength). |
| Assumptions: | None |
| Preconditions: | None |
| Post conditions: | The song has been buffered and is ready to be played upon command. |
| Related business rules: | None |
| Author: | Rebecca Young |
| Date: | November 4, 2011 |

3.2.3 Streaming Manager

**3.2.3.1 Screen Layouts: 1 – Search View**



1: Text box – User can type a song name, artist name or album

2: Search button – Search in the Streaming Service the parameter typed in the text box

3: Play button – Play selected song

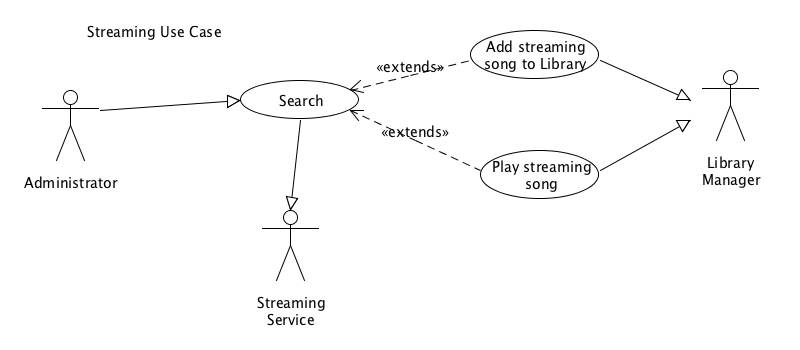
4: Add button – Add selected song to the current playlist

5: List of Songs – List of songs found

6: Cancel Button – Cancel operation

7: Ok Button – Conclude operation

**3.2.3.2 Use Case Diagram**



**3.2.3.3 Use Case Templates**

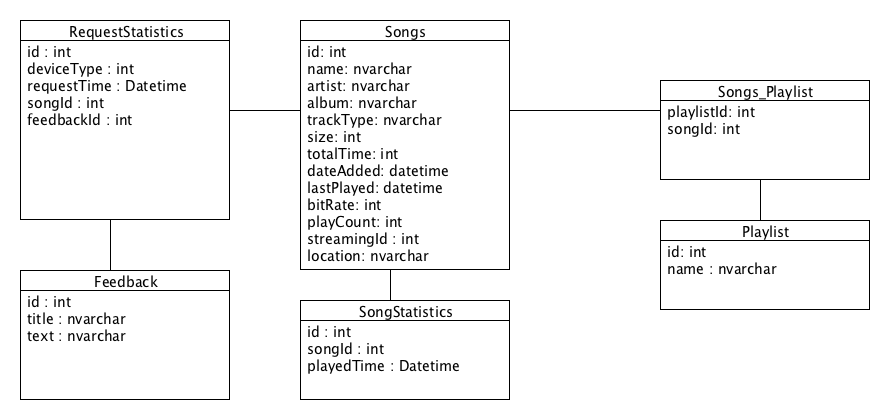
|  |  |
| --- | --- |
| Use Case Name: | Search |
| Iteration: |  |
| Summary: | The administrator wants to add songs from the streaming service to the library so first he has to search for it |
| Basic Course of Events: | 1. This use case begins when the administrator click on the search textbox 2. type name of artist, song or album that he is looking for and click search 3. the system respond with a list of songs found 4. See use case “Add Streaming song to Library” |
| Alternative Paths: | 4. Administrator can decide to play the song instead of adding to library. See use case “Play streaming song “ |
| Exception Paths: | If the system doesn’t find any songs it will return an empty list. |
| Extension Paths: | If the internet connection is interrupted it will not be possible to search for songs. I message will alert the administrator. |
| Triggers: | The administrator has decided to search for a song in the streaming service |
| Assumptions: | The computer running the application has access to the internet |
| Preconditions: | The administrator search for an artist, album or song name that exist. |
| Post conditions: | The administrator will receive a list of items found. |
| Related business rules: | none |
| Author: | Lineker Tomazeli |
| Date: | October 25, 2011 |

|  |  |
| --- | --- |
| Use Case Name: | Add Streaming song to Playlist |
| Iteration: |  |
| Summary: | The administrator wants to add songs from the streaming service |
| Basic Course of Events: | 1. This use case begins with the administrator searching for a song (see use case “Search”) 2. System respond with a list of songs found 3. Administrator select a song and clicks on “add” button 4. System will add song to current playlist and create references in the database 5. Song will be showed in the current playlist 6. Administrator can repeat steps 1-5 again if needed. |
| Alternative Paths: | 2. Administrator can double click the song he wants, this action replace the functionally of clicking in the “add” button |
| Exception Paths: | If the internet connection is interrupted it will not be possible to add the song. I message will alert the administrator. |
| Extension Paths: | none |
| Triggers: | The administrator has decided to add a streaming song to his library by selecting the song from the list found and clicking in “add” or double clicking the song. |
| Assumptions: | A search was made and records were found. Please see use case “Search” for details. |
| Preconditions: | The administrator search for an artist, album or song name and select one of the records found |
| Post conditions: | Song is added to the current library. |
| Related business rules: | none |
| Author: | Lineker Tomazeli |
| Date: | October 25, 2011 |

|  |  |
| --- | --- |
| Use Case Name: | Play Streaming song |
| Iteration: |  |
| Summary: | The administrator wants to play a song from the streaming service |
| Basic Course of Events: | 1. This use case begins with the administrator searching for a song (see use case “Search”) 2. System respond with a list of songs found 3. Administrator select a song and clicks on “play” button 4. System will start streaming (playing) the song, if there was a song playing before it will stop 5. Song will be showing in the current playing queue |
| Alternative Paths: | none |
| Exception Paths: | If the internet connection is interrupted it will not be possible to play the song. I message will alert the administrator. |
| Extension Paths: | none |
| Triggers: | The administrator has decided to play a streaming song by selecting the song from the list found and clicking in “play” |
| Assumptions: | A search was made and records were found. Please see use case “Search” for details. |
| Preconditions: | The administrator search for an artist, album or song name and select one of the records found |
| Post conditions: | Song starts to play and is added to the current playing queue. |
| Related business rules: | none |
| Author: | Lineker Tomazeli |
| Date: | October 25, 2011 |

3.2.5 Database

**3.2.5.1 Database Diagram**



**Songs table**Holds information of all the songs imported to the music manager.

id – Primary key  
name – Name of the song  
artist – Name of the artist  
album – Name of the album  
trackType – Type of the file, streaming or local.  
size – Size of the song file  
totalTime – Song length  
dateAdded – Date that was added to the Library  
lastPlayed – Date that was last played  
bitRate – Bit rate  
playCount – Counter for how many times song was played  
streamingId – Reference id to be used in the Streaming service  
location – Location in the local file system

**RequestStatistics table**Holds statistical information about request made by users

id – Primary key  
deviceType – Type of device (Android or iOS) that the request was originated.  
requestTime – Time that the request was made  
songId – Foreign key with Songs table, if it was a song request.  
feedbackId – Foreign key Feedback table, if it was a feedback request

**Feedback**Holds feedbacks sent by users

id – Primary key”  
title – title of the feedback sent by user  
text – text of the feedback sent by user

**SongStatistics table**Holds statistical information about songs that were played

id – Primary key  
playedTime – Time that the request was made  
songId – Foreign key with Songs table

**Playlist table**  
Holds information about playlists created by the administrator

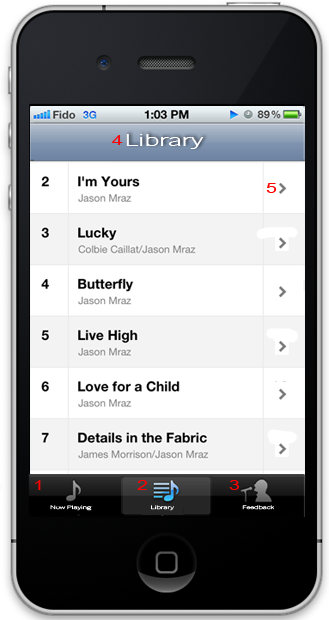
id – Primary key  
name – name of the playlist

**Songs\_Playlist**Pivot table between playlist and songs

songId – Foreign key with Songs table  
playlistId – Foreign key with Playlist table

3.2.9 iOS Application

**3.2.9.1 Screen Layouts: 1 – Playlist Available View**



1: Now Playing tab bar button – shows current playing queue.

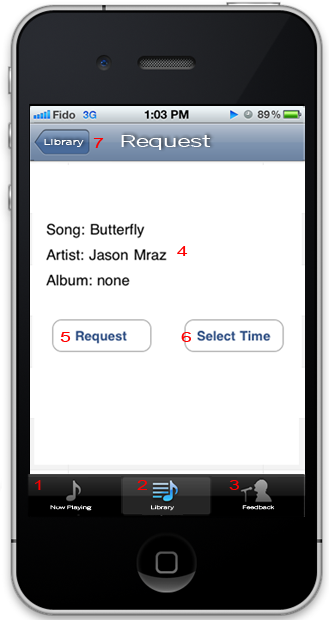
2: Library tab bar button – show playlist available for request

3: Feedback tab bar button – show form to submit a feedback

4: Title – Title for the current section

5: Arrow button – Arrow is showing that the current cell is clickable.

**3.2.9.2 Screen Layouts: 2 – Request Song View**

****

1: Now Playing tab bar button – shows current playing queue.

2: Library tab bar button – show playlist available for request

3: Feedback tab bar button – show form to submit a feedback

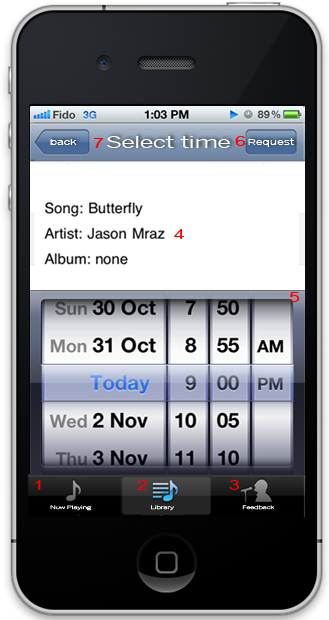
4: Information about the selected song

5: Request button – send a song request

6:Select Time button – show option to select what time use wants to play the requested song.

7: Library button – return to the Library tab. See layout section 3.2.9.1

**3.2.9.3 Screen Layouts: 3 – Select Time View**

****

1: Now Playing tab bar button – shows current playing queue.

2: Library tab bar button – show playlist available for request

3: Feedback tab bar button – show form to submit a feedback

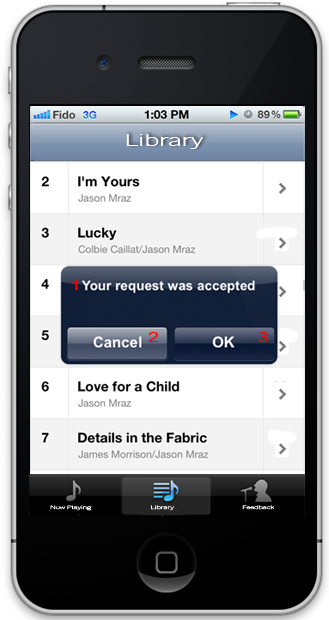
4: Information about the selected song

5: Date and time picker

6: Request button – send the song request

7: Back button – return to the request view. See section 3.2.9.2

**3.2.9.4 Screen Layouts: 4 – Alert View**

****

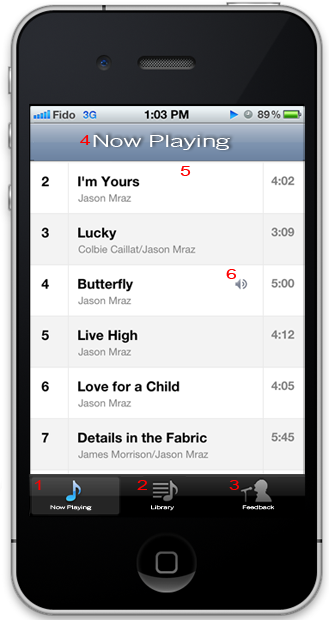
**\* Layout can vary between 1 to 5 buttons**

1: Alert Message

2: Ok button – Confirmation of action

3: Cancel button – Cancelation of action

**3.2.9.5 Screen Layouts: 5 – Playing Queue View**



1: Now Playing tab bar button – shows current playing queue.

2: Library tab bar button – show playlist available for request

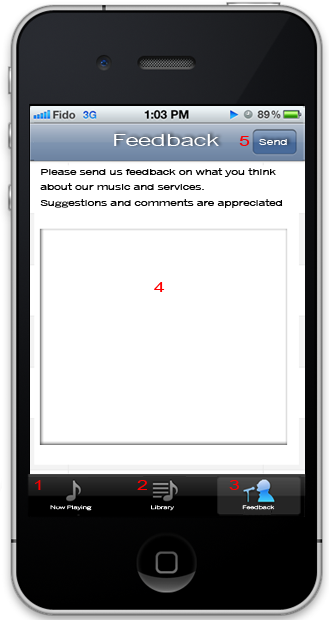
3: Feedback tab bar button – show form to submit a feedback

4: Title – title of this section

5: List of current playing queue

6: Indication of the current playing song

**3.2.9.6 Screen Layouts: 6 – Feedback View**



1: Now Playing tab bar button – shows current playing queue.

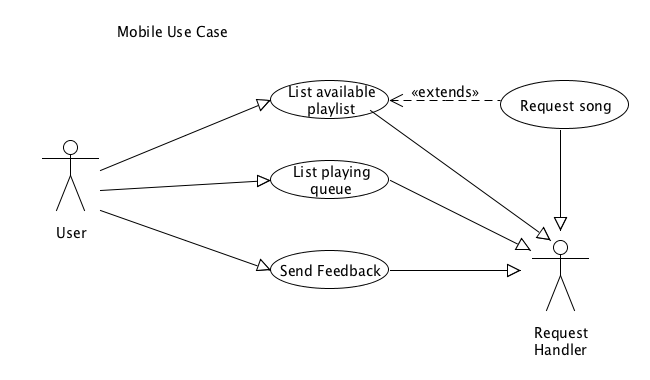
2: Library tab bar button – show playlist available for request

3: Feedback tab bar button – show form to submit a feedback

4: Text box – use can type its feedback message

5: Send button – Send feedback request.

**3.2.9.7 Use Case Diagram**



**3.2.9.8 Use Case Templates**

|  |  |
| --- | --- |
| Use Case Name: | List available playlist |
| Iteration: |  |
| Summary: | The user wants to view what songs are available to request. |
| Basic Course of Events: | 1. This use case begins with the user opening the mobile application (iOS) 2. Select tab “Library” 3. The system communicates with the server asynchronously. 4. The system responds with a list of available songs.  See section 3.2.9.1 Screen Layout 1 |
| Alternative Paths: | none |
| Exception Paths: | If the system doesn’t find the server it will return a empty list and alert the user. See section 3.2.9.4 for alert layout |
| Extension Paths: | User can also “pull down” the list to refresh. |
| Triggers: | The user has decided to view the list of songs available for request. |
| Assumptions: | There is a server running in the network. |
| Preconditions: | The user has an device with the application installed |
| Post conditions: | The user will receive a list of songs available. |
| Related business rules: | none |
| Author: | Lineker Tomazeli |
| Date: | October 26, 2011 |

|  |  |
| --- | --- |
| Use Case Name: | List playing queue |
| Iteration: |  |
| Summary: | The user wants to view what songs will be played. |
| Basic Course of Events: | 1. This use case begins with the user opening the mobile application (iOS) 2. Select tab “Now Playing” 3. The system communicates with the server asynchronously. 4. The system responds with a list of songs.  See section 3.2.9.5 Screen Layout 5 5. The system will highlight the current playing song and show the 5 last played and the 3 next songs |
| Alternative Paths: | none |
| Exception Paths: | If the device is connected to a network but the system doesn’t find the server, it will return a empty list and alert the user  The device is not connected to any network, it will alert the user. See section 3.2.9.4 for alert layout |
| Extension Paths: | User can also “pull down” the list to refresh. |
| Triggers: | The user has decided to view a list of songs that were played, the song that will be played and the songs that will be played. |
| Assumptions: | There is a server running in the network. |
| Preconditions: | The user has an device with the application installed |
| Post conditions: | The user will receive a list of songs. |

|  |  |
| --- | --- |
| Use Case Name: | Send Feedback |
| Iteration: |  |
| Summary: | The user wants to send a feedback to the administrator |
| Basic Course of Events: | 1. This use case begins with the user opening the mobile application (iOS) 2. Select tab “Feedback” 3. User types a feedback message and click “Send” See section 3.2.9.6 Screen Layout 6 4. The system sends the feedback message to the server asynchronously. 5. The system responds with an alert message saying that message was delivered. See section 3.2.9.4 for alert layout |
| Alternative Paths: | none |
| Exception Paths: | If the device is connected to a network but the system doesn’t find the server, it will return alert message to the user  The device is not connected to any network, it will alert the user. See section 3.2.9.4 for alert layout |
| Extension Paths: | none |
| Triggers: | The user has decided to send a feedback message to the administrator |
| Assumptions: | There is a server running in the network. |
| Preconditions: | User types a feedback message and click send |
| Post conditions: | Server save the message in the database and send a confirmation message to the user. |

|  |  |
| --- | --- |
| Use Case Name: | Request Song |
| Iteration: |  |
| Summary: | After user had listed the available playlist, he decided to request a song to play in the sound system. |
| Basic Course of Events: | 1. This use case begins with the user opening the mobile application (iOS) 2. Select tab “Library”. See section 3.2.9.8, use case “List Available Playlist” for details 3. Select a song. 4. The system will send the user request to the server and responds with a confirmation message saying that message was added to the playing queue. |
| Alternative Paths: | 3. After selecting a song, the user has the option of specifying what time he wants to play it. |
| Exception Paths: | If the device is connected to a network but the system doesn’t find the server, it will return alert message to the user  The device is not connected to any network, it will alert the user. |
| Extension Paths: | The server might return a message saying that the request could not be completed. |
| Triggers: | The user has decided to request a song |
| Assumptions: | There is a server running in the network. |
| Preconditions: | User chooses a song from the current playlist |
| Post conditions: | Song is added to the playing queue. |

IV. Supporting Information

(optional Appendices. Mention added risk from Prelim Design Doc)