SE 3XA3: Software Requirements Specification
DJ NODE

Team 12, DJS
Amandeep Panesar - panesas2
Victor Velenchovsky - velech
Taha Mian - miantm

November 13, 2016

Contents

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Context . . . 1
- 1.3 Design Principle . . . 1
- 1.4 Structure . . . 1

2 Anticipated and Unlikely Changes **2**
- 2.1 Anticipated Changes . . . 2
- 2.2 Unlikely Changes . . . 2

3 Module Hierarchy **3**

4 Connection Between Requirements and Design **3**

5 Module Decomposition **4**
- 5.1 Hardware Hiding Modules (M1) . . . 4
- 5.2 Behaviour-Hiding Module . . . 4
  - 5.2.1 Create User ID (M3) . . . 4
  - 5.2.2 Displays Options(M4) . . . 5
  - 5.2.3 Input Vote(M5) . . . 5
  - 5.2.4 Display Total Votes(M6) . . . 5
  - 5.2.5 Play Music(M8) . . . 5
- 5.3 Software Decision Module . . . 5
  - 5.3.1 Get Library(M2) . . . 5
  - 5.3.2 Tally Votes(M2) . . . 6

6 Traceability Matrix **6**

7 Use Hierarchy Between Modules **7**

List of Tables

1 **Revision History** . . . ii
2 Module Hierarchy . . . 3
3 Trace Between Requirements and Modules . . . 6
4 Trace Between Anticipated Changes and Modules . . . 6

List of Figures

1 Use Hierarchy Between Modules . . . 7

Table 1: **Revision History**

Date	**Version**	**Notes**
9/11/16	1.0	Started Sections 1,3,7
11/11/16	1.1	Finsihed Sections 1,3,7 Starting Sections 2,5,6
12/11/16	1.2	Finished All Sections
13/11/16	1.3	Review

1 Introduction

1.1 Overview

The world in today's day and age is evolving at an exponential pace. Both the automotive and retail industry have adapted and are becoming more and more technologically advanced. However, the entertainment sector and more specifically the playlist sub-sector of the entertainment industry has been struggling to keep up with technology. The DJS dynamic playlist system was designed to create a more modern approach to the music and entertainment industry. The DJS application allows the users to vote for specific songs through a fair and democratic voting system.

1.2 Context

This Module Guide (MG) document will provide a modular decomposition of the system and show the structure of the application. This document will also provide how the system meets functional and nonfunctional requirements described in the Software Requirements Specifications (SRS) document. In addition, the Module Interface Specification (MIS) will explain semantics of the system. The separate document follows the MG document and provides syntax,variables, and functions for each module.

1.3 Design Principle

The design principles employed in DJS are Information Hiding and Encapsulation. The use of both these development guidelines help each module hide secrets from the entire system. Encapsulation also allows data and functions to be accessed by certain modules and keeps the data and implementation safe from misuse or interference.

1.4 Structure

This MG document is organized in the following manner:

- Section 2 Anticipated and Unlikely Changes
- Section 3 Module Hierarchy
- Section 4 Connection Between Requirements and Design
- Section 5 Module Decomposition
- Section 6 Traceability Matrix
- Section 7 Use Hierarchy Between Modules

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

The items listed below are the anticipated changes to occur in the design of DJS. The modularized design of DJS however allows only certain aspects of the implementation to be altered.

AC1: The specific hardware on which the server is running.

AC2: The function which tallies votes for each individual song is hidden.

AC3: The function which lists all songs in the music folder is hidden.

AC4: The votes functions which increments the votes.

AC5: The server function which hosts the home website should be hidden.

AC6: The socket.io emit function that allows the server and home page to communicate will be hidden.

AC7: A user's casted vote on a particular song should be hidden.

2.2 Unlikely Changes

The following items are unlikely to change even after implementing the anticipated changes.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a static source for music.

UC3: The format of the media will stay consistent.

UC4: The location where the server is hosted.

UC5: The number of songs that are allowed for voting (5 songs can be shown as options for the user to vote for).

3 Module Hierarchy

The following are modules implemented in DJS. The modules are placed into a hierarchy based on decomposed secrets as displayed in the table below. The leaves listed below make up the hierarchy tree and are implemented in the system.

M1: Hardware-Hiding Module

M2: Get Library

M3: Create User ID

M4: Displays Options

M5: Input Vote

M6: Tally Vote

M7: Display Total Votes

M8: Play Music

Level 1	**Level 2**
Hardware-Hiding Module	M1
Behaviour-Hiding Module	M3 M4 M5 M7 M8
Software Decision Module	M6 M2

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The system was built to fulfill the requirements created during the design process. The server class in the DJS system calls and connects all components of the program which are then served to the end user. The requirements pertaining to appearance and usability are fulfilled by the home class which deals with creating the outline for the client side. The Get Library Module however does the background task of creating a list of five options the user can pick from. The result is then parsed into the Display Options Module which formats and

places the five options as specified in the home class. In addition, while the options are being generated the user is assigned a unique identifier which is then saved on the users browser using cookies and is implemented in the Create User ID Module. The user will then cast a vote for their favourite song which is sent to the server using the Input Vote Module. This input module allows the server class to then initiate the Tally Votes Module which totals all votes for every distinct option. After tallying the votes the software then decides on what song was voted the most and instantiates the Play Music Module. The server class allows the Play Music Module to execute if there is no song currently being played which in turn plays the option with the most votes. While those modules are running in the background the Display Total Votes Module is updating the counter in the home class to provide the user feedback on which song is the most voted and will most likely be played. The client side will also have a user help page which can provide instructions for the user if necessary. However, since the options are listed and are buttons the operation of the system should be intuitive and will only have static input (vote for song 1,2,3,4,or 5 no user text input). In addition, in order to satisfy the requirement of a song should always be played the project has set some defaults. The defaults are required when all options have equal or no votes since no clear majority exists. The default when this situation occurs is to pick a random song instead. Another requirement that is satisfied is keeping the graphical interface clean and simple which allows for intuitive control. The result of this allows only song titles to be shown which also helps avoid contradiction in cultural and political requirements.

5 Module Decomposition

5.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

5.2.1 Create User ID (M3)

Secrets: The creation and assigning of a unique identifier for any given connected user.

Services: Creates a cookie which is stored on the connected users browser to uniquely identify the user quickly.

Implemented By: socket.io

5.2.2 Displays Options(M4)

Secrets: The song options which are displayed to the user.

Services: Uses an array passed from the library module to display the five options on the client side.

Implemented By: DJS

5.2.3 Input Vote(M5)

Secrets: The option picked by a connected user.

Services: Using the cookies set by another module the user is identified and the vote is recorded for other modules to manipulate.

Implemented By: DJS

5.2.4 Display Total Votes(M6)

Secrets: The total votes for each option is displayed.

Services: Gets the total from tally votes and displays them to the client side aspect of the system.

Implemented By: DJS

5.2.5 Play Music(M8)

Secrets: Plays the music from input.

Services: Uses the input parameter and plays the associated music with the default os sound player.

Implemented By: play-sound

5.3 Software Decision Module

5.3.1 Get Library(M2)

Secrets: Sets the five song options for the user.

Services: The module reads from the music directory and picks five songs which are then placed into an array for other modules to manipulate.

Implemented By: DJS

5.3.2 Tally Votes(M2)

Secrets: Tally votes for all five options and play the most voted song else pick a random song.

Services: The module takes in user data and counts the votes for each song. Afterwards, the votes are compared with one another and the highest voted option is played. Otherwise if all songs have the same amount of votes (no votes or tied votes) it will pick a random song.

Implemented By: DJS

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M2, M4, M7, M5
R2	M2, M4
R3	M4,M7,M6
R4	M5,M3,M2, M4
R5	M2, M4
R6	M3
R7	M2, M4, M7, M6, M5
R8	M2, M4, M7
R9	M4, M6, M5, M7
R10	M8, M2, M6
R11	M3, M5, M6, M4

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M6
AC3	M2
AC4	M6
AC5	M1
AC6	M7,M4
AC7	M5

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

The DJS system was design to adhere to the principle of all Use Hierarchies having directed acyclic graphs (DAGS) which avoids cyclic dependencies. In addition, since all modules are connected to the Hardware-Hiding Module which interfaces with system hardware it has been shown separately.

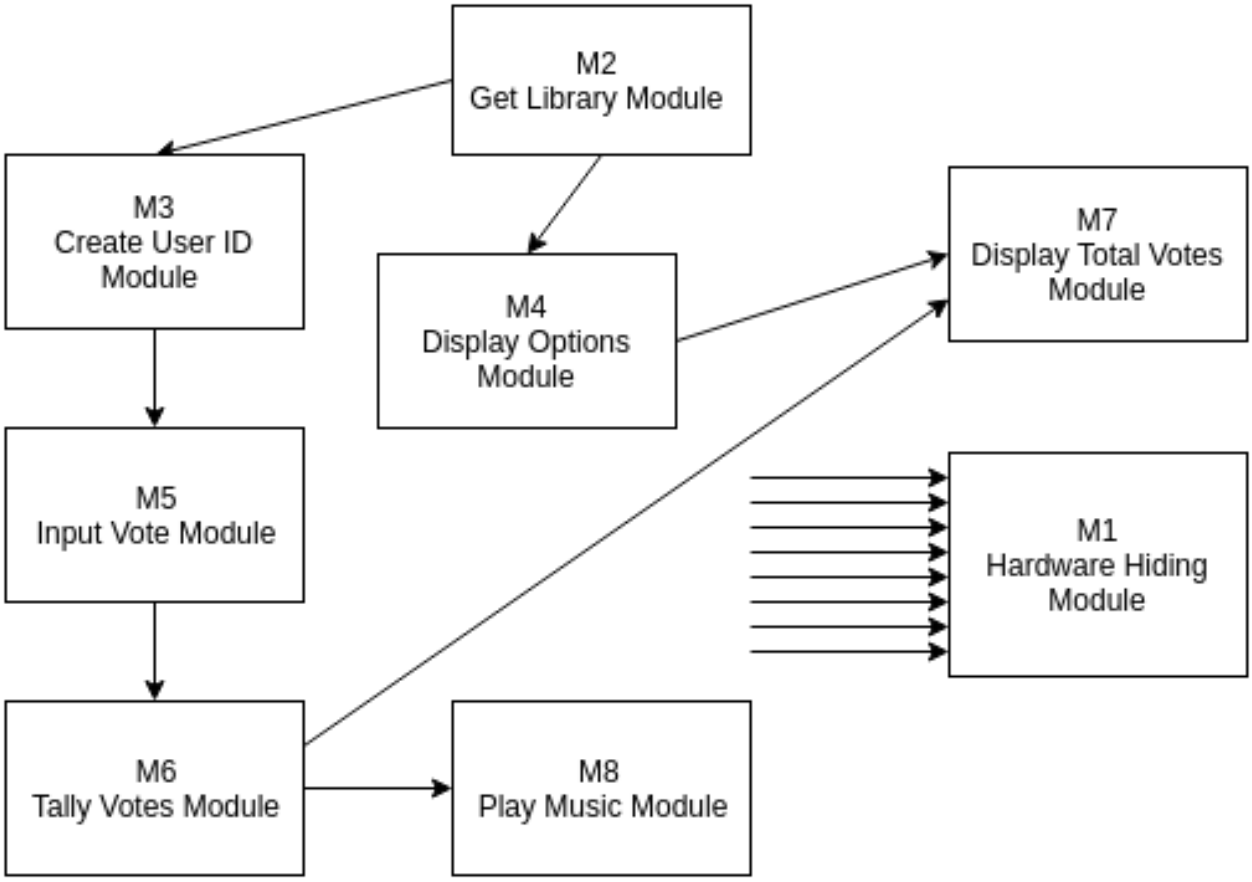

Figure 1: Use Hierarchy Between Modules