

6-5-2015

WeJ collaborative playlists

Jason Dougherty
Santa Clara University

Nicholas Fong
Santa Clara University

Alexander Hurst
Santa Clara University

Malia Lum
Santa Clara University

Follow this and additional works at: http://scholarcommons.scu.edu/cseng_senior



Part of the [Computer Engineering Commons](#)

Recommended Citation

Dougherty, Jason; Fong, Nicholas; Hurst, Alexander; and Lum, Malia, "WeJ collaborative playlists" (2015). *Computer Science and Engineering Senior Theses*. Paper 12.

This Thesis is brought to you for free and open access by the Student Scholarship at Scholar Commons. It has been accepted for inclusion in Computer Science and Engineering Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Santa Clara University

Department of Computer Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Jason Dougherty
Nicholas Fong
Alexander Hurst
Malia Lum

ENTITLED

WeJ Collaborative Playlists

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

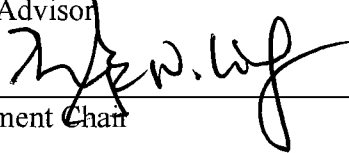
BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING



Thesis Advisor

June 5th 2015

Date



Department Chair

June 9, 2015

Date

WeJ Collaborative Playlists

by

Jason Dougherty
Nicholas Fong
Alexander Hurst
Malia Lum

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California

Spring 2015

WeJ Collaborative Playlists

by

Jason Dougherty
Nicholas Fong
Alexander Hurst
Malia Lum

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California

Spring 2015

WeJ Collaborative Playlists

Nicholas Fong
Jason Dougherty
Alexander Hurst
Malia Lum

Department of Computer Engineering
Santa Clara University
June 5, 2015

ABSTRACT

WeJ (pronounced we-jay) is a mobile web application that provides users with the ability to collaboratively create music playlists and listen to them with each other in real time. Users will be able to search for songs to be added to the playlist, select songs from pre-made libraries, and up and down vote songs to determine what will be played next. Our goal is to bring people together through the power of music.

Acknowledgments

We would like to thank many people for all of their support and encouragement throughout the entire Senior Design Project. First and foremost is Dr. Ahmed Amer, our senior design advisor, who has been very helpful from the beginning. We would also like to thank Professor Robin Everest for helping us prepare for the conference presentation and this entire document. Lastly, WeJ would be nothing without our alpha testing users who are also our friends, so thank you for your time and support.

Table of Contents

Signature Page.....	i
Title Page.....	ii
Abstract.....	iii
Acknowledgments.....	iv
Table of Contents.....	v
List of Figures.....	vii
List of Tables.....	viii
Chapter 1: Introduction.....	1
1.1 Problem Statement.....	1
1.2 Background.....	2
1.3 Objectives.....	3
Chapter 2: Use Cases.....	5
Chapter 3: Design and Implementation.....	6
3.1 Technologies Used.....	6
3.2 User Interface.....	7
3.3 Front End.....	8
3.4 Back End.....	8
3.5 Music API.....	9
Chapter 4: Testing and Documentation.....	11
4.1 Testing.....	11
4.2 Documentation.....	12
Chapter 5: Project Management.....	13
5.1 Project Schedule.....	13
5.2 Risk Management.....	13
Chapter 6: Societal Issues.....	14
6.1 Ethical.....	14
6.2 Social.....	14
6.3 Economic.....	14
6.4 Political, Health and Safety, and Manufacturability.....	15
6.5 Sustainability and Environmental Impact.....	15
6.6 Usability.....	15
6.7 Lifelong Learning.....	15
Chapter 7: Conclusion.....	17
7.1 Summary.....	17
7.2 Lessons Learned.....	17
7.3 Future Improvements.....	17

Appendix A: Project Management.....	A-1
A.1 Project Schedule.....	A-1
A.2 Risk Analysis.....	A-2
Appendix B: Conceptual Model.....	B-1
B.1 Playing Track View.....	B-1
B.2 Queued Tracks View.....	B-2
B.3 Libraries View.....	B-3
B.4 Search and Add Track View.....	B-4

List of Figures

2.1	Shared Link Use Case.....	5
3.1	AWS Architecture.....	7
A.1	Gantt Chart.....	A-1
B.1	Playing Track View.....	B-1
B.2	Queued Tracks View.....	B-2
B.3	Libraries View.....	B-3
B.4	Search and Add Track View.....	B-4

List of Tables

A.2 Risk Analysis..... A-2

Chapter 1: Introduction

1.1 Problem Statement

At social gatherings, music is one of the key factors that can contribute to the success or failure of an event. Music has the power to bring people together. Not only does it set the mood, the right song can generate excitement or fill awkward silences during a lull in conversation. When music can satisfy everyone at an event, it is a definite success.

Usually at these events, an appointed Disc Jockey (DJ) is given the power to control not only the music, but also the overall atmosphere of the party. This DJ must account for each attendee's music taste and satisfy requests with what little music library he or she has. Often times the DJ may fail to succeed at his or her job because he or she is not accommodating the music to the guests' needs. It is common to see conflicts arise between the DJ and a person who dislikes the music being played. It can be quite inconvenient to constantly switch those who are in charge of the music because it causes delay between songs. Clearly, the way music is chosen at parties and other group settings is not ideal. It is exclusive, limited, and inefficient. We believe everyone in attendance deserves a chance to choose what music is played at social gatherings.

We will be building a web-based and mobile friendly application that provides users with one place to collaboratively create playlists to be played and enjoyed in real-time.

Through our application, users will set up an account and have the ability to join groups specific to a genre, artist, or name of an event. Once they have joined a group, they can search for a song and add it to the playlist. Users can up-vote or down-vote songs based on their own musical taste. Our application will include free access to thousands of songs to satisfy any type of music preference. Users will be able to create, add to, and join as many playlists as they want and have access to numerous amounts of music with the simple click of a button. Our application seeks to eliminate the exclusivity of choosing music at a party and to expand the limited music library of one person. We will connect

aspiring song choosers to one simple place while simplifying the difficult role of being the “DJ.”

1.2 Background

Currently, a simple way to create collaborative music playlists with your friends does not exist. At social gatherings, an appointed DJ has full control of the music. Others must make song requests in hope that the DJ has the song they are looking for, or change who has access to the music controls. This process is inefficient and messy.

There have been three applications and products that have tried to eliminate this problem, but failed. iTunes DJ offered the first solution in March 2009 (arstechnica.com). This application allowed users to search and add songs to collaborative playlists with their friends using a remote application on the iPhone. Users needed to own an iPhone, use iTunes as their music source, and be connected to the same wireless network.

Additionally, the only music that could be added was limited to the user’s personal song library. This application was removed from iTunes in April 2013 because it lacked popularity (arstechnica.com).

Another attempted solution was Soundrop.fm. This application was founded in November 2011 and raised about 6.4 million dollars in funding (crunchbase.com). Soundrop.fm was available as an application within the music platform Spotify. Spotify is an online music streaming service that offers users access to millions of songs and the ability to make playlists that are accessible through a phone or computer (Spotify.com). Within Spotify, Soundrop.fm allowed users to create collaborative music playlists with friends or other random Spotify account holders. The problem with this solution was that it was limited to Spotify’s music library. In addition, Spotify recently changed its policy on applications for its desktop and mobile platform. Although there were millions of Soundrop.fm users, it had to shut down in December 2014 (show.co/blog/heavy-hearts).

The final existing solution to this problem is the use of specific bluetooth speakers. The Jabra SOLEMATE and Sol Republic DECK are speakers that allow multiple users to

connect to the speaker and add songs to a queued playlist. The problems associated with these speakers are the following: users are tied down to a specific speaker platform, there is a limited amount of users able to connect to the platform at once, and users are unable to see the queued songs added to the playlist.

1.3 Objectives

We will describe our objectives as requirements. Functional requirements define what the application will do. Non-functional requirements define the manner in which the functional requirements are achieved. Design constraints define how the application is implemented.

1.3.1 Functional Requirements

The application will allow users to:

1. Create collaborative playlists
2. Listen to playlists together in real time
3. Up and down vote songs to determine what will be played next
4. Search for songs to add to the playlist
5. Select songs from pre-made libraries to add to the playlist
6. Log into the application using their Facebook account

The application also will have an accompanying marketing website that explains what WeJ is and provides other relevant information.

1.3.2 Non-functional Requirements

The application will be:

1. Hybrid iOS Web Application (using Apple's WKWebView API)
2. Portable (will work on a variety of different device types)
3. User friendly (the design will be intuitive)

1.3.3 Design Constraints

The application:

1. Must be web-based
2. Will work on most modern platforms

Chapter 2: Use Cases

Our project revolves around a WeJ, or playlist, where users will go and listen to music. Therefore we have created our primary use case with this in mind.

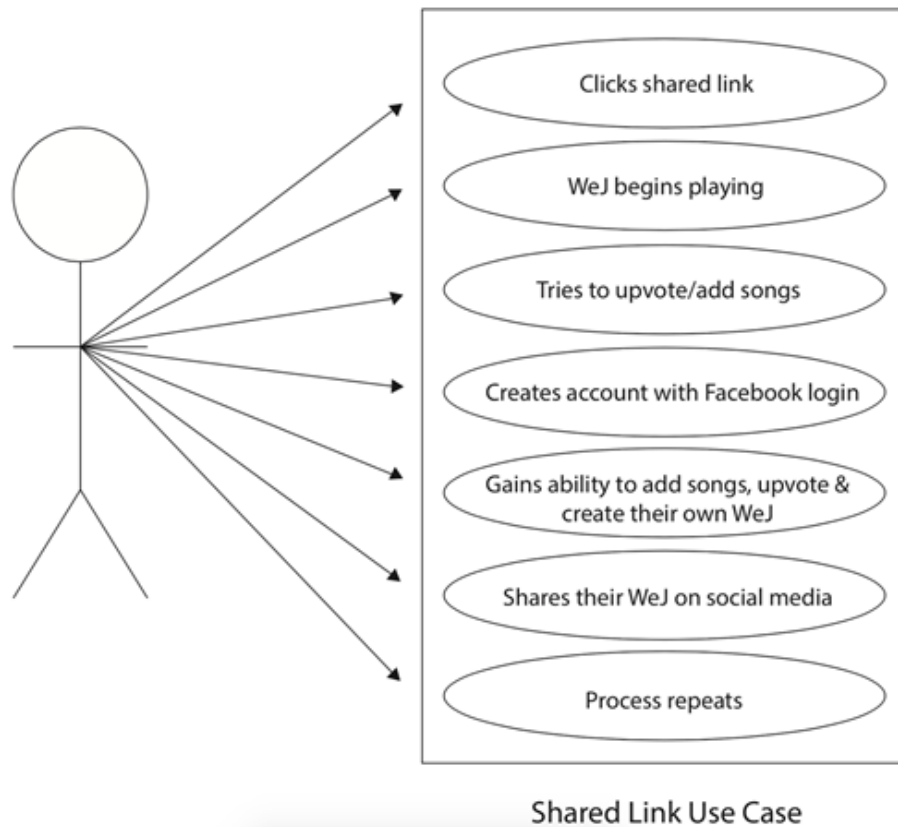


Figure 2.1: Shared Link Use Case

Chapter 3: Design and Implementation

3.1 Technologies Used

In this section we will briefly list and describe what technologies we used. In the sections that follow, we go further in depth as to why we chose these technologies.

Client Side Technologies:

- HTML
- Cascading Style Sheets (CSS)
- Bootstrap
- AngularJS
- SoundCloud JS API
- Apple iOS WKWebView – Native Web Wrapper
- WebSockets.

Back End Technologies:

- Ruby on Rails
- MySQL
- WebSockets

Hardware and Architecture:

- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Load Balancing (ELB)
- Amazon Relational Database Service (RDS)
- Amazon Route 53
- Amazon CloudFront

The manner in which our Hardware and Architecture comes together is shown below:

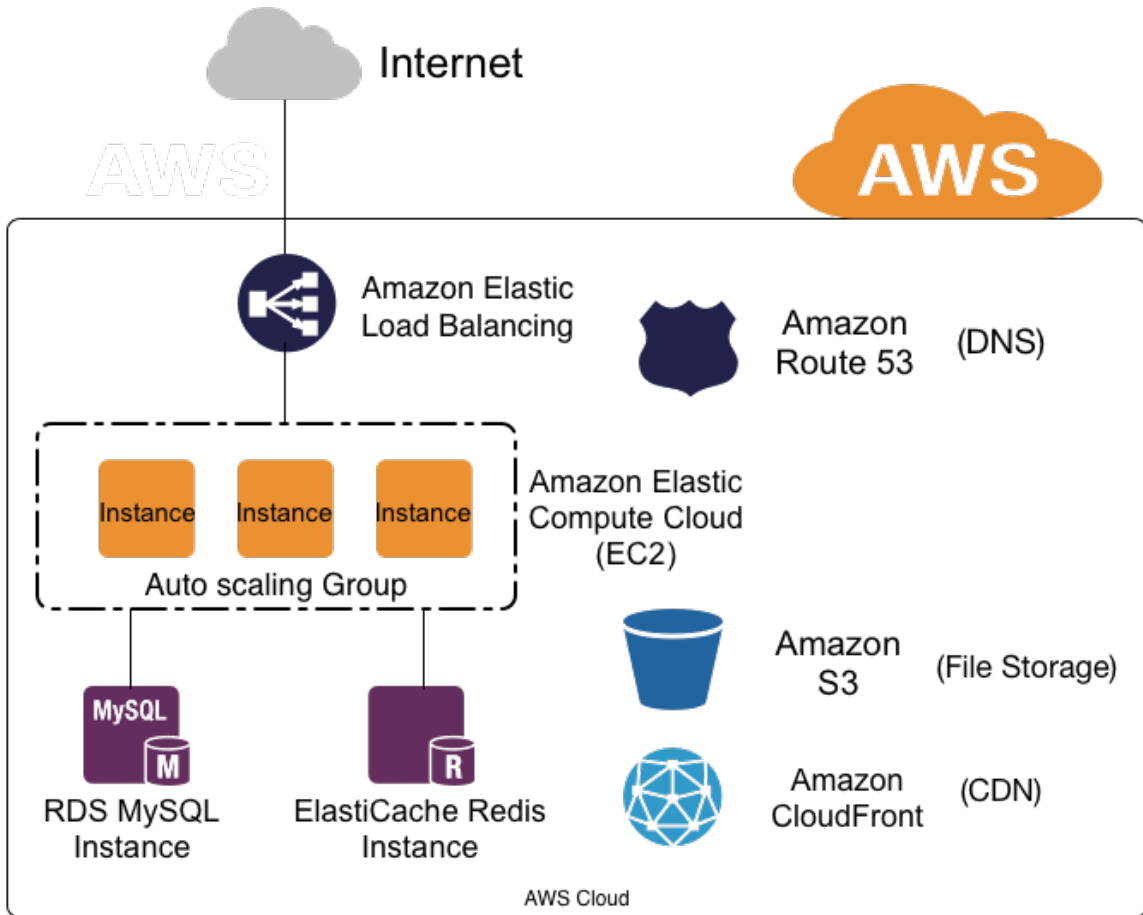


Figure 3.1: AWS Architecture

3.2 User Interface

The user interface presented in the conceptual model (located in Appendix B) has a simple feel, with easy navigation through the application. We want playing and adding playlists (WeJs) to be easy, responsive, and engaging. Users listening to a WeJ will ideally be in sync with all others listening to the same WeJ at the same time. This gives a sense of personality and community to each WeJ. This means that the songs in the WeJ will not be permanently saved or linked to it. However, users can save the songs they like in song libraries that are associated with the playlist in case they want add them in future listening sessions. The basic idea is that a playlist will develop its own general theme with its associated music libraries, but it won't ever be static or saved.

Users adding songs to the WeJ will be able to up-vote and down-vote each entry once. These votes will determine the order of the songs in the playlist. This method gives the songs the people want to hear the greatest priority. It also actively engages all users tuned into the playlist.

3.3 Client Side Technologies

We decided to use AngularJS as a front-end JavaScript framework for our application. Using AngularJS allows us to easily implement Model View Controller (MVC) in a standardized fashion, resulting in a structured front-end code base. AngularJS also has data binding, which means that we do not have to insert data into our view manually, eliminating many computationally expensive Document Object Module (DOM) manipulations. Ultimately AngularJS allows us to focus solely on the view without the distractions of developing a custom MVC system.

The primary reason for using Bootstrap Cascading Style Sheets (CSS) is its “mobile first” philosophy. Using the framework’s built-in grid system, scaling our website to any screen size will be simple.

We have decided to make a hybrid web application using Apple’s WKWebView for a number of reasons. Using this technology, we can create a web application with the look and feel of a native application. We also want our application to be comprised of one singular code base so that we can expand to other platforms with minimal effort.

3.4 Back End Technologies

Our backend is going to be written using the Ruby on Rails web application framework. Using the Rails framework will give us a template for server side development using a Model View Controller (MVC) design pattern. Using MVC, we can isolate our data from our presentation, reducing the software coupling of our application. Rails also allows for agile development with clean syntax reducing our overall backend complexity.

We will be using Oracle's MySQL for storing our persistent data. The nature of our data model is inherently relational consisting of a few one-to-many relationships and several many to many relationships. MySQL was designed for these types of relationships and has well defined guidelines and support for implementing them. Rails is also well integrated with MySQL, providing Object-Relational Mapping (ORM) through its Active Record Query Interface. Using an ORM creates a layer of abstraction over SQL, allowing us to generate queries using Rails. The end result is a more fluid and consistent code base that is easy to modify and build upon.

The ultimate goal of the back end is to build a robust and easily accessible JavaScript Object Notation (JSON) based API. We want to have the server generate as little HTML as possible for a greater application response time. When the application loads it sends a series of requests to our API endpoints. The requests will be processed through our server side APIs that will query MySQL, ultimately delivering a JSON response back to the client.

We utilized WebSockets in order to implement an event-based mechanism to manage the current playing track, queue, and voting. When a device connects to the server, it initiates and keeps a connection to the server open. Whenever an event occurs, the server pushes the change to all connected devices. WebSockets offers a great advantage over other solutions such as asynchronous JavaScript and XML (AJAX) long polling because it requires constant requests to the server in a given interval, potentially making requests that are superfluous and unnecessary.

3.5 Music API

After much research we narrowed our decision for music streaming down to three providers: SoundCloud, YouTube, and Grooveshark. We ultimately decided to use SoundCloud as the music provider for our system for a number of reasons. We discovered that SoundCloud has an impressively robust JavaScript API for searching and playing songs. The abundance of features and tools they provide allowed us to fully develop our entire application. YouTube was the best choice in terms of music selection,

but it is not a music-based service. A user can search for and add videos that are not music. Grooveshark also had a great selection of music, but lacked the API support we needed to fully implement our project. Additionally, Grooveshark shut down their service on April 30, 2015, stemming from ongoing legal issues. However, we designed the backend of our application to be flexible so that it can adapt to other music providers if needed.

Chapter 4: Testing and Documentation

4.1 Testing

To ensure our application was reliable and robust, we needed to have a test plan. As part of this plan we did do two types of testing: unit and end-to-end.

4.1.1 Unit Testing

- Involved testing individual units of code to ensure that they consistently do what is expected
- Used Ruby's built-in testing framework
- Written before implementing a function (test driven development)

4.1.2 Automated End-to-End Testing

- Ensured all parts (front and back end) of our application came together and worked from start to finish
- Automated using Selenium WebDriverJS, Mocha, and Chai
- Written upon feature completion

Our continuous integration server, Jenkins, ran both unit and end-to-end tests on every build. This ensured that as we added onto WeJ we were not breaking existing functions and features. Subsequently, if we change a feature the associated test must also be updated.

4.1.3 Real World Testing

We used WeJ at social gatherings where music was being played. We attached a set of speakers to a laptop or phone running the application and told friends to add tracks.

We had two key findings during this stage of testing:

1. SoundCloud, our music provider, has many DJ radio shows. One of our acquaintances added one of these shows and it dominated the social gathering since a show is an hour in length. As a result, we limited track length to eight minutes.
2. We encountered many SoundCloud errors related to the player. Some artists' uploads (example: Seven Lions) must be played using the Widget player rather than Music Manager.

4.2 Documentation

We used the practice of “documenting as we go” throughout the development of WeJ. This ensured that this entire document was up to date with the latest changes, remained accurate throughout the development process, and made our lives easier when finishing.

Chapter 5: Project Management

5.1 Project Schedule

To ensure we completed our project in a timely and efficient manner, we created a Gantt chart to help assign tasks to each team member and allowed them to know when each task was due. See Gantt Chart in Appendix A.1.

5.2 Risk Management

With our engineering project, specific risks were involved. In order to assess and account for these risks, we presumed what the risks are and how to plan for them if needed. For each risk we assigned a probability value between 0 and 1 and a severity level between 1 and 10. To calculate a risk's impact we multiplied those two numbers together. See the Risk Analysis table in Appendix A.2 that lists our risks sorted from highest to lowest impact.

Chapter 6: Societal Issues

6.1 Ethical

In today's digital age, privacy has become a very large ethical issue. In the beginning stages of WeJ, we wanted to have user accounts that included a user's real name, so when a track is added to the queue, everyone knew who added it. In our later designs we completely removed that feature and opted instead for less intrusive device identifiers. In other words, to add a song to a WeJ's queue, the user no longer needs to create an account. On the back end we can still monitor device identifiers for abuse, but we have no personally identifying information about our users (except their taste in music).

Another ethical issue that we encountered is the legality of music online. In the end, however, this was up to whoever's music API we used. For example, SoundCloud has their policies on who can upload music and how illegally uploaded music is taken down.

6.2 Social

We at WeJ like to say we are "bringing people together through the power of music." The development and creation of our system puts the power of choosing music into the hands of the masses. Gone are the days that a single individual has the power to control the music at a social gathering. Using WeJ allows anyone the ability to queue up a track.

6.3 Economic

Software engineering projects are unlike any other engineering projects because the cost related to development is minimal. The tools required, like APIs and basic hosting solutions, are generally available free to use. Providers do this to lure startups in and later hoping that later the developers continue to use them as they grow. The most important resource that we did use was time. Time, not money, was important in making WeJ a reality.

6.4 Political, Health and Safety, Manufacturability, and Compassion

WeJ does not have any direct effects on public policy because it is a pure software solution. We did not need to take into consideration the will of the general public because people are not forced to use our application. Additionally, health and safety concerns are mostly irrelevant except the issue of music volume, which was out of our control. And since this is a privately hosted web application, manufacturability for the general public is out of their hands and purely up to us. Lastly our project has no compassion component.

6.5 Sustainability and Environmental Impact

Our cloud solutions provider is Amazon Web Services (AWS) and our environmental impact is directly tied to their sustainability practices. According to Amazon, as of April 2015, 25% of the power consumed in their global infrastructure comes from renewable energy sources. Also, some of their locations run on 100% carbon-neutral energy.

Additionally by going with a cloud-based solution, rather than dedicated servers, we eliminate energy waste. Dedicated servers are computers that are always on and “ready-to-go”. By definition cloud based solutions are virtual; they share resources with other customers and can be started and stopped based on demand, rather than being always on.

6.6 Usability

Web applications that are not user-friendly usually tend to fail. When building WeJ we always had the user in mind. Starting from the conception phase we ensured that our application’s design was straightforward by going through mock-ups and user testing. Additionally our application is designed like other music players, ensuring that our users have a familiar and intuitive user interface.

6.7 Lifelong Learning

The saying “learning never stops” is especially true in the software engineering world. New technologies in the form of frameworks, libraries, and more are always springing

up. Therefore we as engineers must always be on the lookout of interesting technologies that may help us (and our employers) do a good and efficient job. Throughout the development of WeJ we learned of new ways of doing things by researching different technologies that we could use to implement our project. A great example of this was using WebSockets over AJAX long polling to keep clients in sync.

Chapter 7: Conclusion

7.1 Summary

WeJ is a fun, easy to use, real-time, and collaborative playlist mobile application. It is not tied to a specific platform so the source of music can be easily changed. It is compatible with all speaker systems. WeJ is also fully web based meaning that it can be used on any device with a web browser. WeJ in its early stage has disrupted the way music is chosen at social gatherings and has brought people together.

7.2 Lessons Learned

Throughout the entire Senior Design project we have learned many lessons about taking an idea and transforming it into an actual product.

- Real world testing, actually using the application with users other than the developers, is useful for finding bugs and gaining user feedback about features.
- Having a team with good chemistry is vital for the overall success of the project. We noticed that many start-ups in Silicon Valley always are looking for a good “fit” and now we understand why they do so.
- The scope of our project defined in the fall was too large and impossible to tackle in its original magnitude. We learned that it is better to start small, with a minimum feature set, and build from there.

7.3 Future Improvements

Given more time, the following are improvements and features that we would have like to have implemented in WeJ.

Improved syncing: Currently players update the current track time every five seconds (with time never updating backwards). At any given time multiple listeners may be up to five seconds apart in the track.

Multiple music sources: Our current music source is solely SoundCloud but our backend is designed to handle multiple music APIs. The Search and Add Track front-end code would need to be modified for this feature.

Chat room: Chat rooms would enable anybody listening or simply viewing a WeJ the ability to chat with each other, in case they are not in the same location. This feature would probably take advantage of WebSockets and not actually store messages in the database.

Visual effects: A rather superfluous but awesome feature to add is a visualizer art on the playing screen. Of course the user will have the ability to turn this feature on and off.

Appendix A: Project Management

A.1 Project Schedule

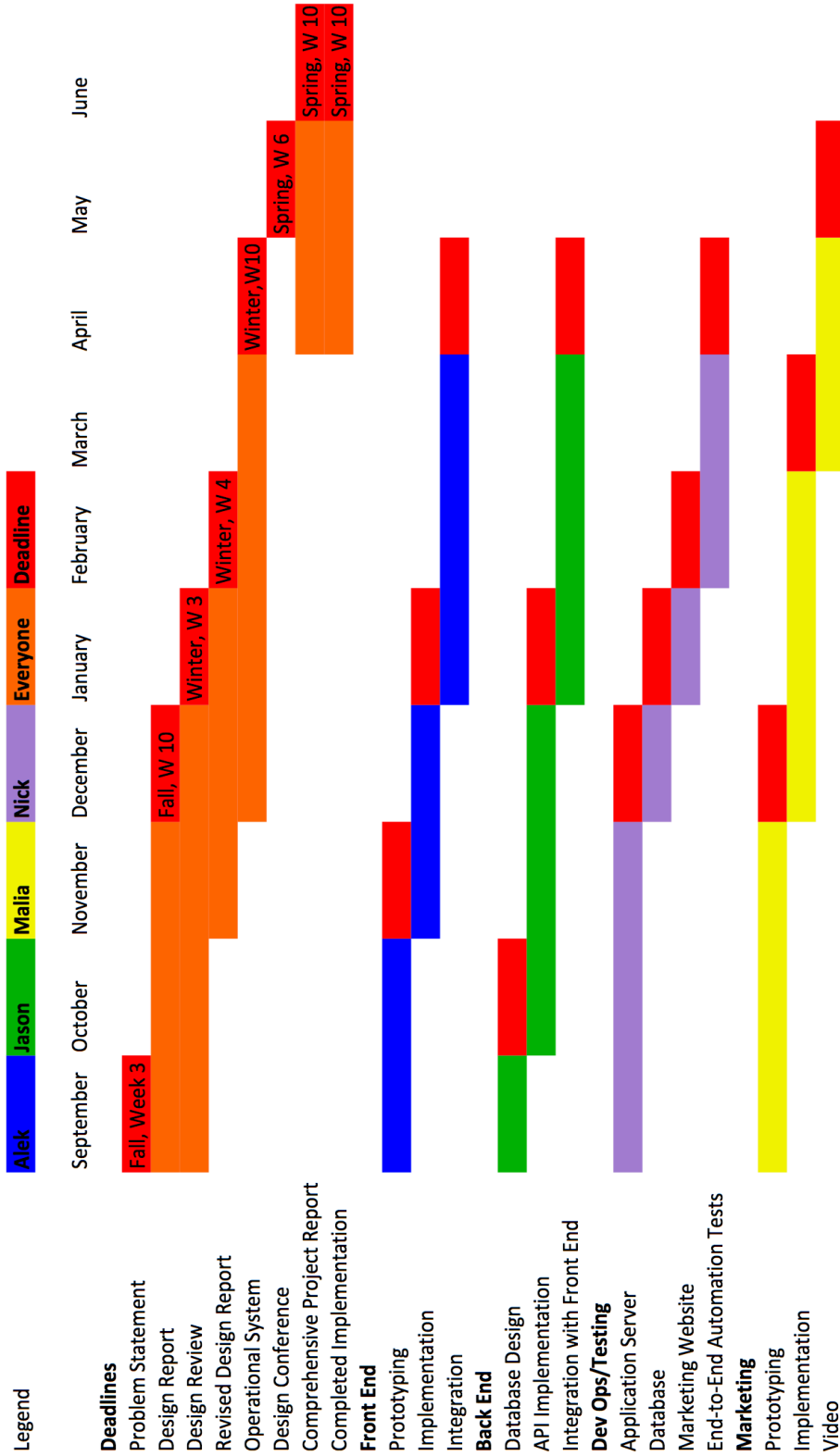


Figure A.1: Gantt Chart

A.2 Risk Analysis

Table A.2: Risk Analysis

Risk	Consequence	Probability (P)	Severity (S)	Impact (P x S)	Mitigation Strategy
Bugs	<p>Restrict our application from performing smoothly and correctly.</p> <p>Our application outputs inaccurate information (e.g. songs, playlists, music information, etc.)</p>	1.0	6.0	6.0	<p>Ensure our test plan has complete coverage with multiple test cases to test our system against.</p> <p>Start early to ensure the application is bug free.</p>
Integration Issues	The application does not work in specified browsers.	1.0	5.0	5.0	<p>Test our design after the HTML/CSS mockup is made.</p> <p>Use strategies and libraries that are cross-platform compatible.</p>
Time	The project is not completed by the deadline.	0.5	9.0	4.5	<p>Follow the Gantt chart for excellent use of time management.</p> <p>Divide work of application into small sections.</p> <p>Consistent group meetings to make sure checkpoints are reached on time.</p>
Data Loss	<p>Work is lost in the process of development.</p> <p>Need to recover lost work.</p>	0.5	6.0	3.0	Maintain consistent backup of code, GitHub.

	Loss of time.				
Personnel	Members of the project are not equally doing their part. Members of the project fall ill and behind on work.	0.2	6.0	1.2	Ensure that each team member is 100% committed to the project. Confirm each member of the group has clear responsibilities and tasks. Speak to underperforming team members.
User Acceptability	Users are unsatisfied with layout, design, color scheme, etc.	0.2	6.0	1.2	Conduct user surveys and acceptance testing to ensure design concept is pleasing.

Appendix B: Conceptual Model

B.1 Playing Track View



Figure B.1: Playing Track View

This view contains the following: the name of the WeJ the user is currently in, album artwork, and track information.

B.2 Queued Tracks View

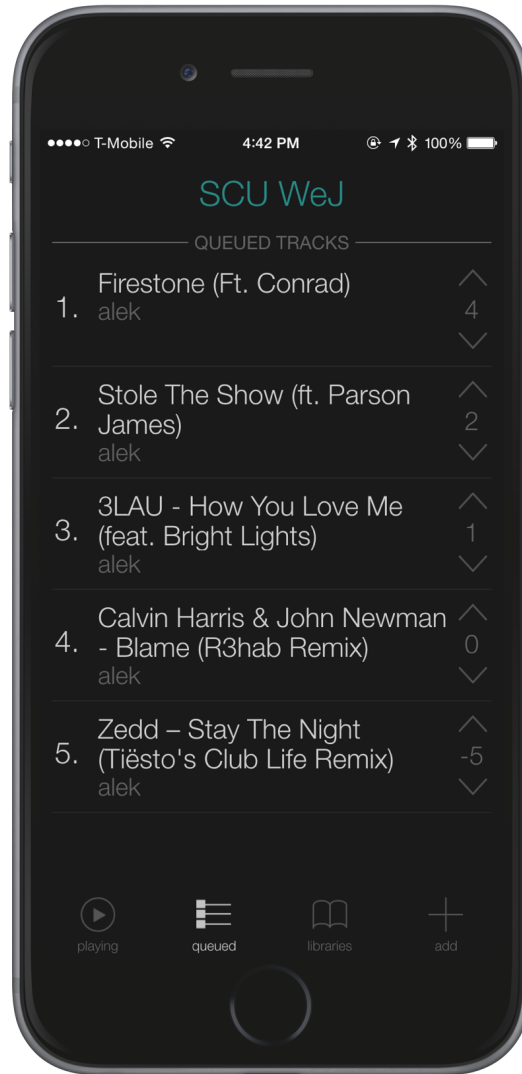


Figure B.2: Queued Tracks View

This view contains a list of the tracks that are in the queue and the ability to up and down vote tracks.

B.3 Libraries View

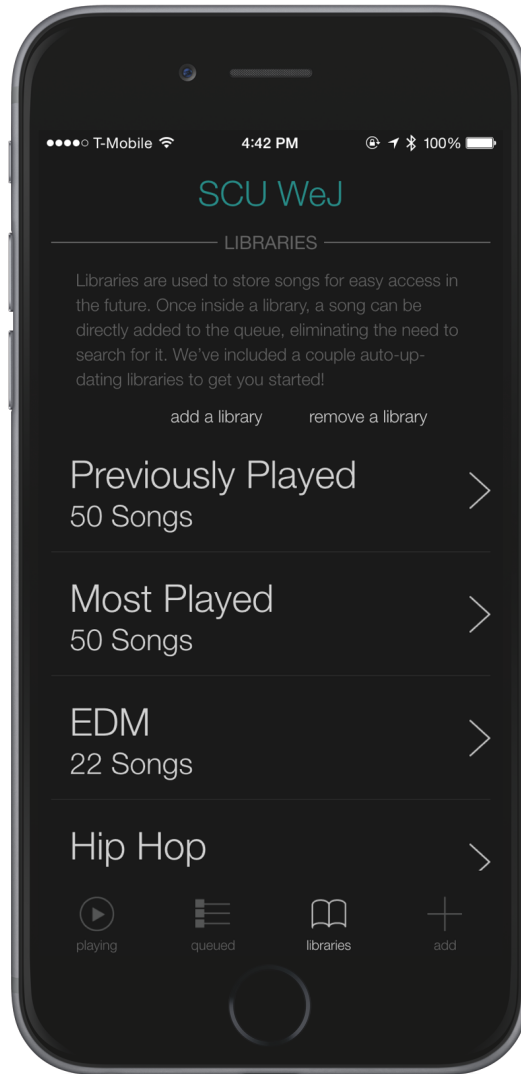


Figure B.3: Libraries View

This view contains a list of libraries that users can go into and select songs from.

B.4 Search and Add Track View

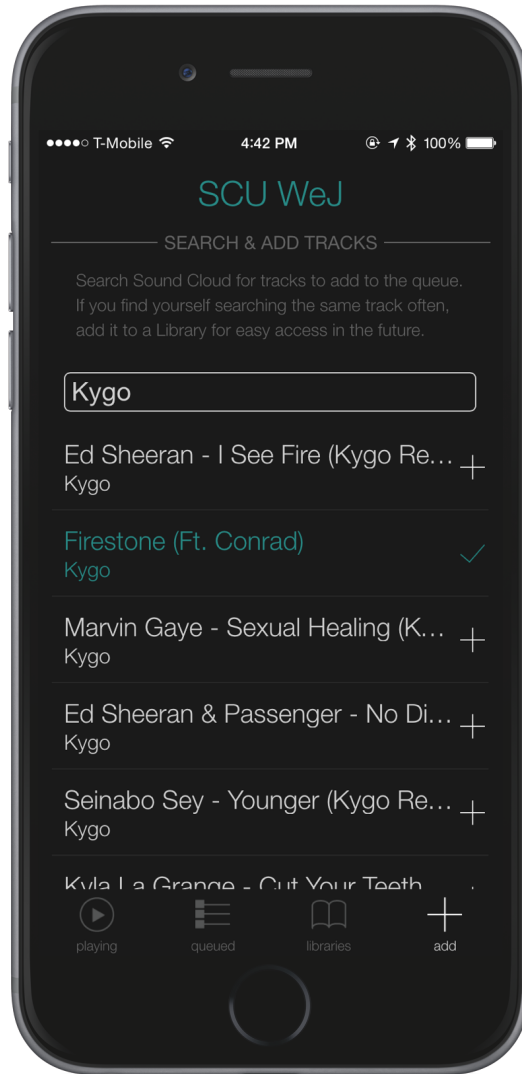


Figure B.4: Search and Add Track View

This view contains a search box allowing the user to input a query, then displays the results below. A “+” next to each track adds the track to the queue and turns into a checkmark afterwards.

The manner in which our Hardware and Architecture comes together is shown below:

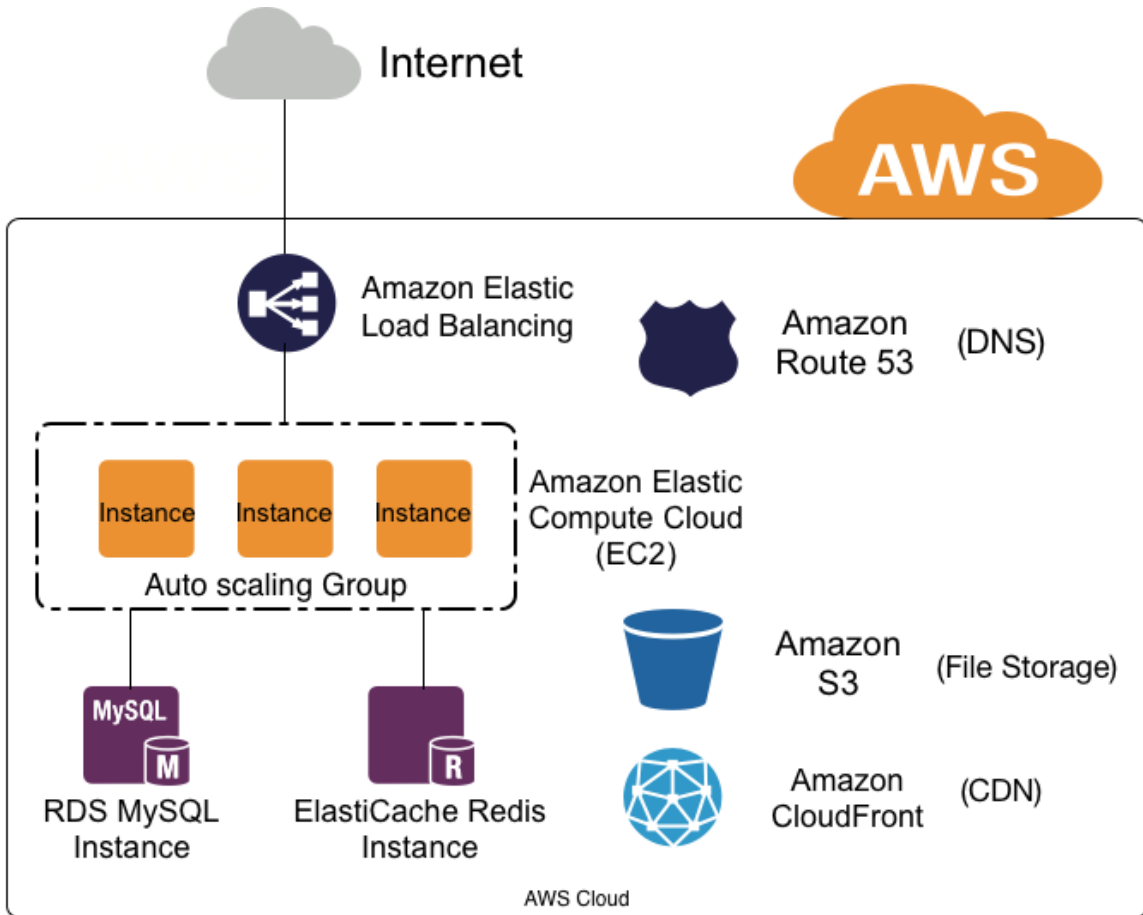


Figure 3.1: AWS Architecture

3.2 User Interface

The user interface presented in the conceptual model (located in Appendix B) has a simple feel, with easy navigation through the application. We want playing and adding playlists (WeJs) to be easy, responsive, and engaging. Users listening to a WeJ will ideally be in sync with all others listening to the same WeJ at the same time. This gives a sense of personality and community to each WeJ. This means that the songs in the WeJ will not be permanently saved or linked to it. However, users can save the songs they like in song libraries that are associated with the playlist in case they want add them in future listening sessions. The basic idea is that a playlist will develop its own general theme with its associated music libraries, but it won't ever be static or saved.

Appendix A: Project Management

A.1 Project Schedule

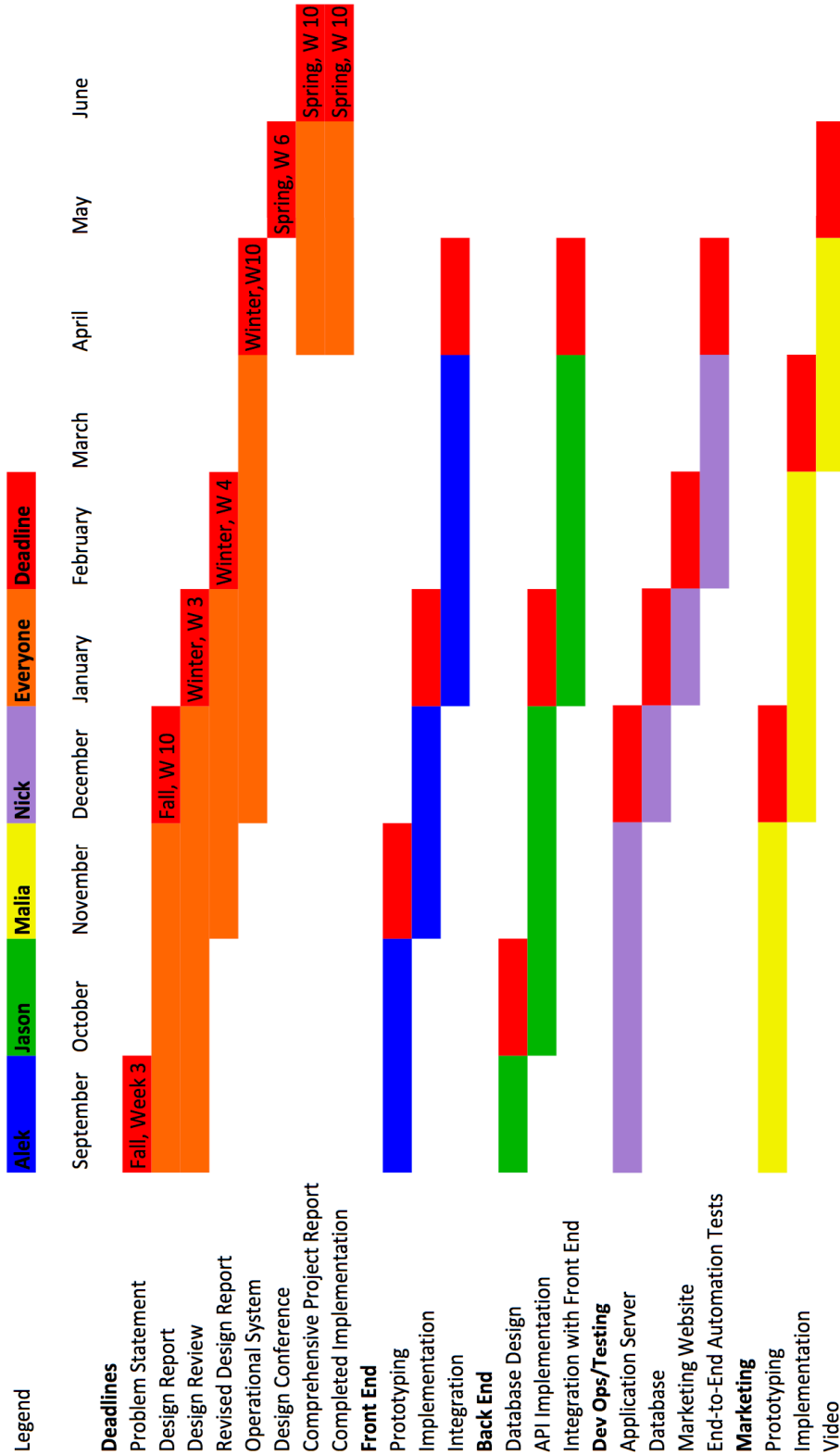


Figure A.1: Gantt Chart

Appendix B: Conceptual Model

B.1 Playing Track View



Figure B.1: Playing Track View

This view contains the following: the name of the WeJ the user is currently in, album artwork, and track information.

B.2 Queued Tracks View

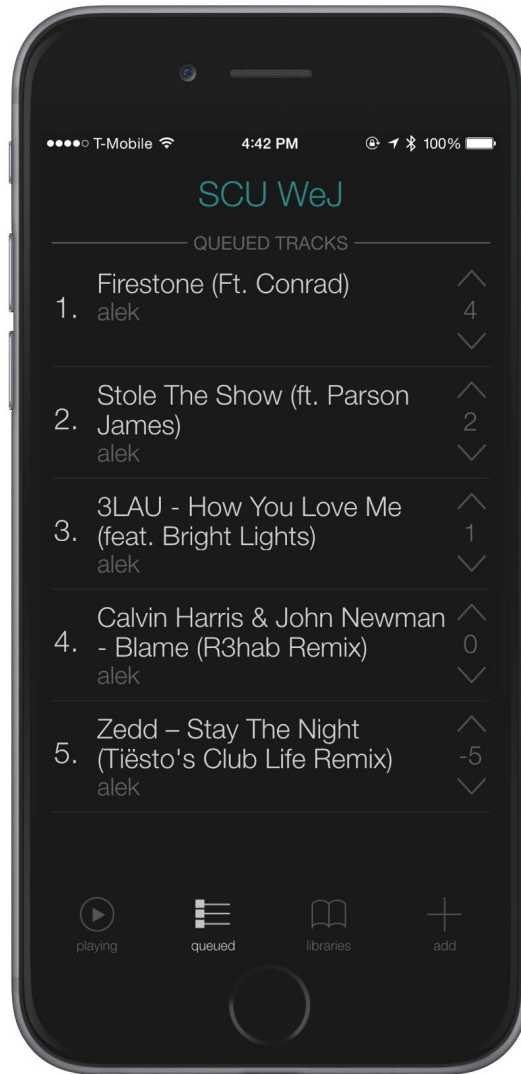


Figure B.2: Queued Tracks View

This view contains a list of the tracks that are in the queue and the ability to up and down vote tracks.

B.3 Libraries View

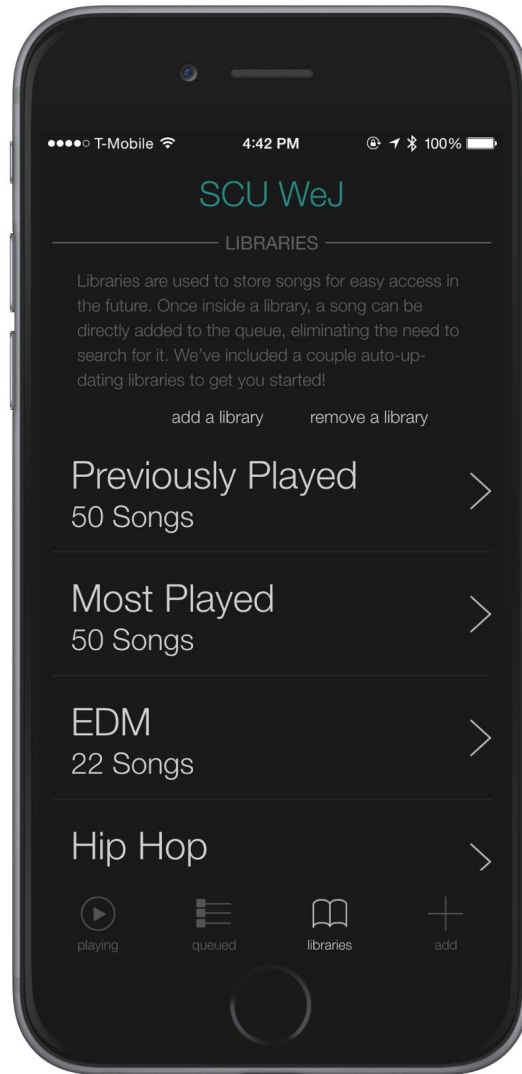


Figure B.3: Libraries View

This view contains a list of libraries that users can go into and select songs from.

B.4 Search and Add Track View

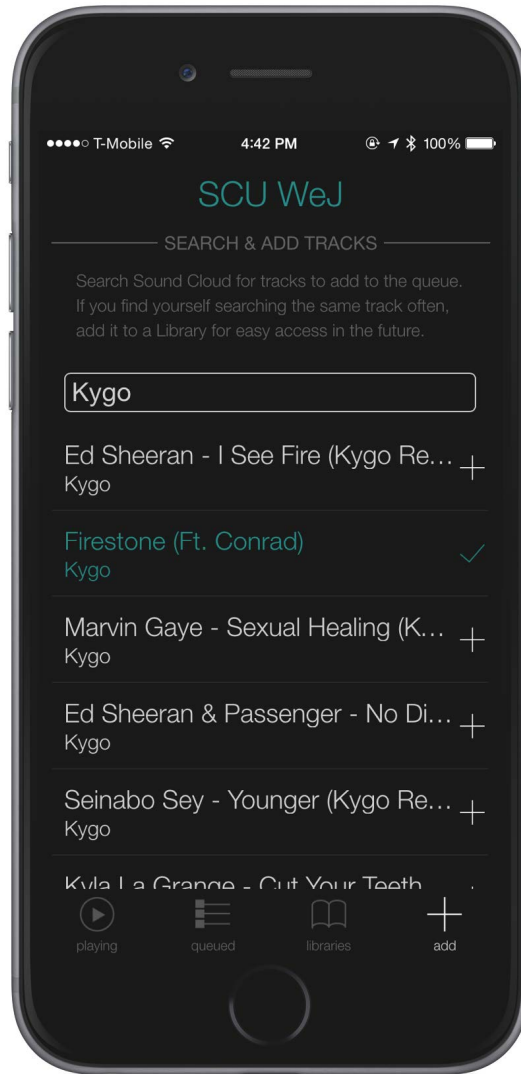


Figure B.4: Search and Add Track View

This view contains a search box allowing the user to input a query, then displays the results below. A “+” next to each track adds the track to the queue and turns into a checkmark afterwards.