6-12-2017

PowerPlaylist: A Collaborative Web Application That Aims to Give Everyone a Voice

Sara Cassella
Santa Clara University, scassella@scu.edu

Alexander Polatnick
Santa Clara University, apolatnick@scu.edu

Kristen Ronhovde
Santa Clara University, kronhovde@scu.edu

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: June 11, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Sara Cassella
Alexander Polatnick
Kristen Ronhovde

ENTITLED

PowerPlaylist: A Collaborative Web Application That Aims to Give Everyone a Voice

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN WEB DESIGN AND ENGINEERING

Thesis Advisor

Department Chair

PowerPlaylist: A Collaborative Web Application That Aims to Give Everyone a Voice

by

Sara Cassella
Alexander Polatnick
Kristen Ronhovde

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

Santa Clara, California
June 12, 2017

PowerPlaylist: A Collaborative Web Application That Aims to Give Everyone a Voice

Sara Cassella
Alexander Polatnick
Kristen Ronhovde

Department of Computer Engineering
Santa Clara University
June 12, 2017

ABSTRACT

PowerPlaylist is a client-server based online collaborative playlist that allows guests of a party to have private access to the hosts playlist. Guests can access the queue of songs, request songs, or up-vote or down-vote songs already requested by other guests. Before PowerPlaylist, there was no musical platform that allowed guests at an event to express his or her opinions on what music should be played without distracting the host or DJ. This web application platform solves that problem while requiring little authentication, and not requiring any software downloads in order to be used. PowerPlaylist aims to give every guest a voice.

Acknowledgements

We would like to thank our advisor Professor Atkinson for his guidance, time, and advice.
We would also like to thank Santa Clara University for fostering a learning environment that teaches us to be engineers with competence, conscious, and compassion.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many social and professional events involve music where the DJ ultimately decides which songs to play. However, we believe that every attendee, not just the DJ, should have an opportunity to choose the songs they hear. Currently guests cannot efficiently express his or her music choices, or for the DJ to get an accurate feel for the music opinions of the group as a whole. If attendees want to suggest a particular song, they can only do so by approaching the DJ directly, which is not only inconvenient for the attendees themselves but it can also distract the DJ. Further, often times the venue itself might not allow attendees to speak directly to the DJ. As a result, DJs often lack feedback on their music. Consequently, they cannot cater to the musical taste of their particular audiences. Currently, no software platform on the market allows an audience to quickly and easily give their opinions about what songs they want to listen to at a particular event. Although popular web applications such as Spotify [1], Pandora [2], and iTunes [3] have implemented algorithms that form playlists to cater to a history of past song choices, they do not take a live feed of data from an audience during an event. Because of this, our market is lacking an efficient way for the DJ to tap into the mood or vibe of his or her audience in real time.

Our client-server [4], web-based solution allows an audience to suggest specific songs for the DJ to play, and allows audience members to give feedback on songs suggested by other users. An attendee can sign into the event, recommend songs to the DJ, and upvote or downvote songs found on a live feed. DJs have the ultimate authority over what music is played, but they may find it helpful to have a list of suggested songs from their audience. Instead of speaking with multiple event attendees and noting down their song requests, a method that can get chaotic, a DJ can be more efficient with his or her time and quickly refer to a list to get song ideas that people want to hear, along with feedback to the songs they he or she has selected. The DJ and audience can pick from a variety of songs, which can be found on Spotify or in his or her iTunes library. Therefore, the DJ can cater to

his or her specific audience in real time. PowerPlaylist gives guests more power to be heard at an event while the ultimate power remains with the host.

Chapter 2

List of Requirements

Below are the functional requirements, non-functional requirements, and design constraints of our platform. These are actions that our platform must complete, or specific ways that our platform must be built.

2.1 Functional Requirements

- Allows a host to play and pause music.
- Allows a host to create a playlist by searching for songs and selecting them.
- Allows host to create a party and share a pin code with guests so they can access the playlist.
- Allows a host to authenticate with a music provider in order to access music.
- Allows a host to approve and reject songs suggested by guests.
- Allows guests to search for and suggest songs.
- Allows guests to upvote or downvote songs in the playlist.
- User interface displays up next playlist and suggested songs playlist to host and guests.
- The song that is currently playing is visible to host and guests.

2.2 Non-Functional Requirements

- The user interface is intuitive and easy to use.
- Users can quickly find songs they are looking for.
- There is a seamless integration of music from the music providers to our platform using data abstraction.

- System can handle multiple users processing requests at one time.
- Quickly and easily authenticate guests with a pin code.

2.3 Design Constraints

- System is a web application.
- System must be compatible with commonly used browers.
- System must be mobile responsive.

Chapter 3

Use Cases

As you can see in Figure 3.1, our use case diagram features the most common actions taken by the hosts and guests of a party while using PowerPlaylist. A host creates a party with a pin code, searches for songs, requests songs, upvotes or downvotes songs, approves or rejects songs suggested by guests, and may stop, pause, or play the music at an event. A guest enters the party with a pin code provided by the host of an event, searches for songs, requests songs, and may upvote or downvote songs. Below we study these actions in more detail.

1. **Create a Party With a Pin Code**
    - **Actor:** Host.
    - **Pre-Conditions:** Event does not exist and cannot be accessed.
    - **Post-Conditions:** A playlist is created and made public for guests. A pin number is generated by the system that the host gives to the guests so that they may enter the party.

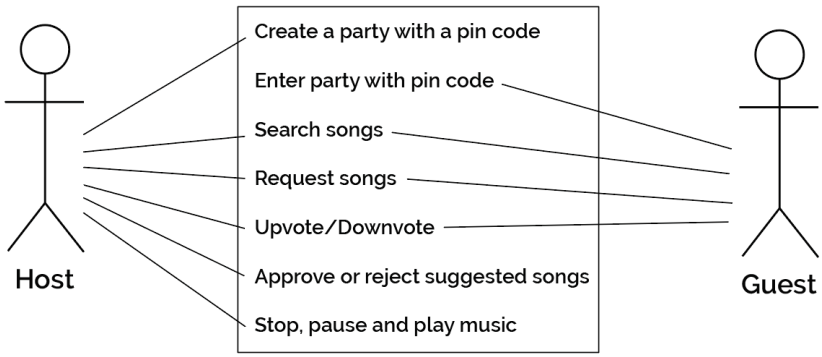

Figure 3.1: Use Case Diagram

2. **Enter Party With a Pin Code**

   - **Actor:** Guest.
   - **Pre-Conditions:** Event must have been created by host and the system generated a pin code. The host must give the pin code to the guest so that he or she may access the playlist.
   - **Post-Conditions:** The guest now has access to the full playlist and can see songs coming up next and songs suggested by other guests.

3. **Search Songs**

   - **Actor:** Host and Guest.
   - **Pre-Conditions:** The host or guest would like to add a particular song to the playlist.
   - **Post-Conditions:** When a song name or artist name is searched in the search bar, a list of songs appear that match that criteria.

4. **Request Songs**

   - **Actor:** Host and Guest.
   - **Pre-Conditions:** Song must have been searched, and is now listed in a drop down menu.
   - **Post-Conditions:** When the song is selected from the dropdown list it is added to a queue of songs that need to be approved or rejected by the host. If a song is approved, it is added to the main playlist.

5. **Upvote or Downvote Songs**

   - **Actor:** Guest.
   - **Pre-Conditions:** Songs must be available in the main Up Next playlist.
   - **Post-Conditions:** A counter next to the song is increased if the guest upvoted the song, or decreased if the guest downvoted the song.

6. **Approve or Reject Suggested Songs**

   - **Actor:** Host.
   - **Pre-Conditions:** Songs must have been searched and requested by guests of an event.
   - **Post-Conditions:** If a song is approved, it is added to the main playlist. If it is rejected, it is deleted and will not be added to the playlist.

7. **Stop, Pause, and Play Music**

   - **Actor:** Host.
   - **Pre-Conditions:** The system is doing an action with the song that the host does not want to continue. For example, a song may be paused and they would like to play the song.
   - **Post-Conditions:** If the host presses stop, the music stops. If the host presses play, the music plays. If the host presses pause, the music pauses.

Chapter 4

Conceptual Model

For our conceptual model we want to keep our user interface very simple. Because searching and requesting songs both rely on a search bar, we have placed the search bar front and center on the page and made it white to improve contrast. We have also alternated shades of grey in between songs on the playlist to include contrast and readability. There are numbers next to each song on the playlist so that it is clear which song is first on the list and which one is last. We have selected a very dark interface because it is more aesthetically pleasing in low light situations, such as parties or concerts. Figure 4.2 through Figure 4.5 are the mobile version of our web application, and Figure 4.6 through Figure 4.9 are the desktop version. We want to have two different viewports depending on what device the user is using to access the application. We kept the pin authentication page very concise by only including an area to input the pin number to enter the system. Once a pin is entered, the main tasks users perform include searching for songs, requesting songs, upvoting and downvoting them, then adding them to the queue. Lastly, there is a section on the hosts interface which allows them to either approve songs suggested by guests and add them to the main playlist, or reject songs suggested by guests which deletes the song.

Figure 4.1: Desktop and Mobile View.

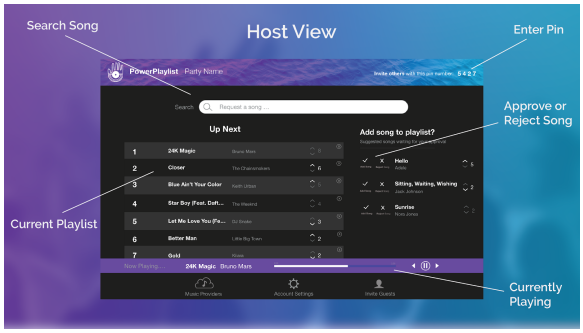

Figure 4.2: Host View With Key Features Highlighted.

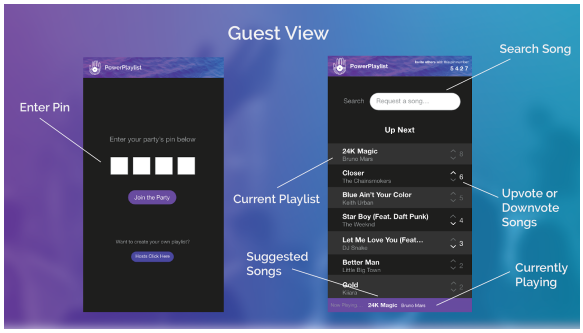

Figure 4.3: Guest View With Key Features Highlighted.

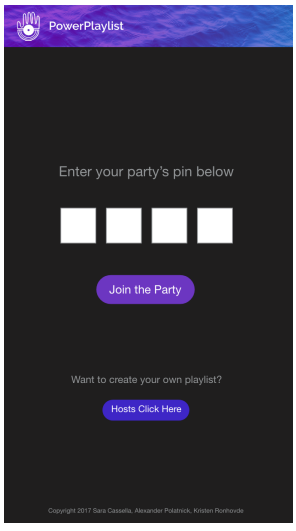

Figure 4.4: Pin Authentication Page On a Mobile View.

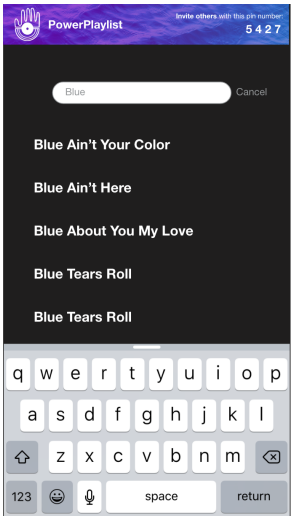

Figure 4.5: Guest Searches For Song in Mobile View and PowerPlaylist Returns Matching Song Titles.

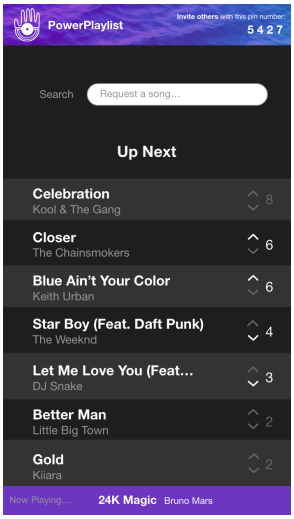

Figure 4.6: Homepage With Songs Coming Up Next.

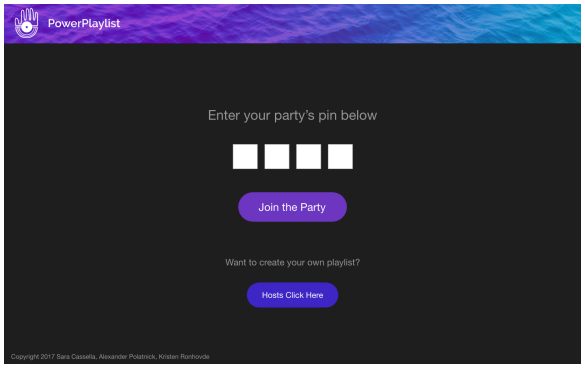

Figure 4.7: Pin Authentication Page On Desktop View.

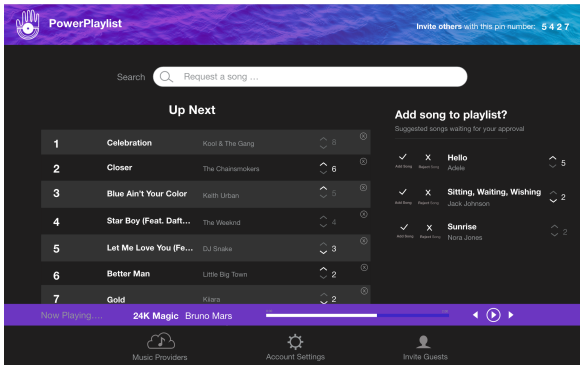

Figure 4.8: Home Page on Desktop View.

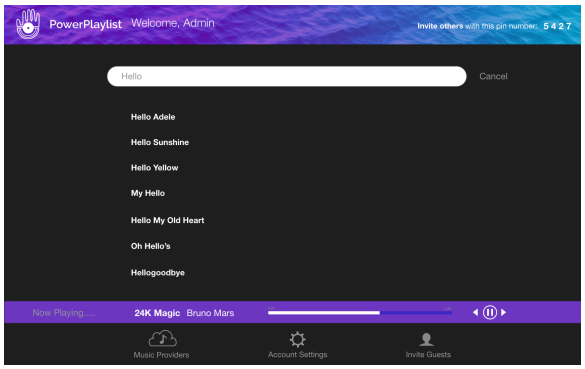

Figure 4.9: Searching For Song in Desktop View.

Chapter 5

Architectural Diagram

PowerPlaylist was built with a client-server architecture as you can see in Figure 5.1. All of the playlist data, and any other data that is relevant to a party is stored on the server. The clients, which are both the guests and the hosts, send AJAX [5] requests to the server in order to receive information needed for each of the interfaces. As a response, the server returns to the clients the requested data. The clients do not communicate with one another like they would in a peer-to-peer architecture [6], rather all communication is done through the server. Clients will be periodically updated because regular AJAX requests are sent to the server in order to receive the latest data. The host is sending the same sorts of AJAX requests, except the host has more power in terms of manipulating the playlist and playing the music. Whichever device the host is using (does not matter as long as they can access a web browser) will have to be connected to some sort of speaker system to play music. Examples of this are through bluetooth or an auxiliary cord. The server is also connecting to music providers through endpoints. The server uses these endpoints to access the appropriate song data without actually streaming a song to play it. After requesting a certain song or list of songs, the server abstracts the data for a seamless integration of the song data from the different music providers to the interfaces. This means that regardless of where a particular song is coming from, it will look and act the same once it is displayed on the user interface. In order to abstract the data we implemented a form of a object-oriented architecture [7]. Instead of classes of songs to abstract the data, we divided necessary song data like the title and artist into arrays which are easily translated to the front-end code.

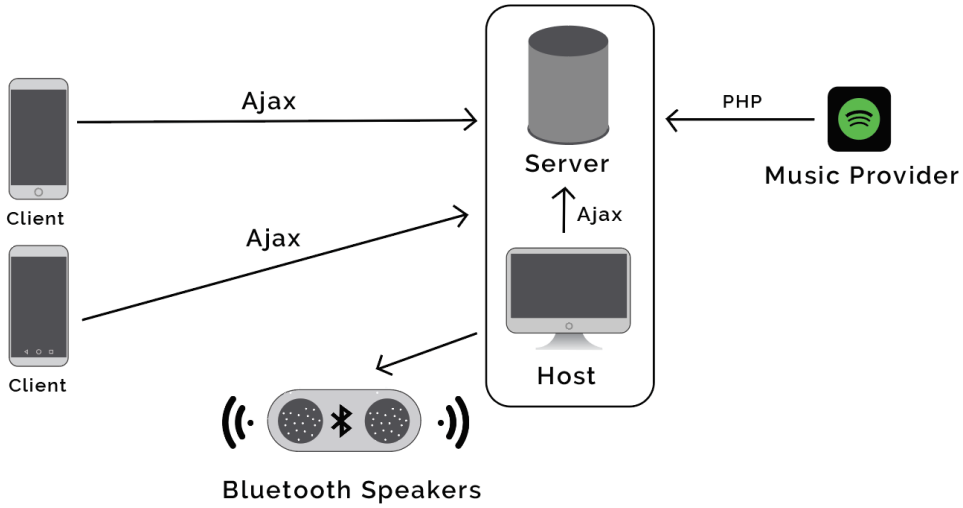

Figure 5.1: Architectural Diagram

Chapter 6

Design Rationale

The rationale behind making the system a web application relates to the idea that the system should be easy to access and use for the guest or attendee at an event. The guest uses his or her smart-phone in order to access use the system on-line. The benefit of doing this instead of creating a native application for iPhone or Android is that when a guest shows up to an event they are usually not interested in having to download the application and waiting to authenticate. Instead, it is likely they would choose to not use the application at all, suggesting songs to the DJ in person. Because we created a web application, users do not have to download anything, rather they can quickly access the web page on-line and authenticate with one simple PIN, chosen by the host of the event. Our original thought was that the host would need to be running a web server on his or her computer in order to allow access to the playlist, however our final design decision was to have this hosted on-line, so essentially the host acts as a guest to an external server. This means that the ease of use for our web application is not just for the guest, but also for the host, unlike our initial design. This ease of use is done by having a client server architecture with dumb clients. In order to make it as easy as possible for the guest to get started using the system, the clients in this architecture simply message the server in order to access its data. The clients send feedback to the server such as suggesting songs from the hosts library but does not save any data itself nor can it communicate directly with other clients. This architecture is more specifically called a blackboard client server [8].

The system is also an object oriented architecture. Creating classes allows the system to abstract data so that it can be used in the user interface seamlessly. Because the system is pulling from multiple music providers, the way we extract the data from each third party music provider is slightly different. The system needed a way to take in all this data and present it in a similar fashion. By using classes to interpret data from each provider, we have functions that if

interpreted by another class can abstract the data that is being extracted. For example, if we have a song class, this class would be the parent class of a SpotifySong class and a iTunesSong class. This means that the two later classes inherit the functions and variables from the song class. The song class is an abstraction of the underlying data which allows the user interface to not have to worry about any of the underlying classes, promoting unity of code and ease of implementation.

We catered our system based on the users requirements. The system allows users to suggest songs into a suggested queue (separately from the selected playlist queue) because event attendees want to have a say in the music that is playing, but we did not want to take away the right of the DJ to select the songs that he or she decides on playing. For example, if a guest suggests an inappropriate song for the setting, that song should not be automatically added the hosts carefully curated playlist, and the host or DJ should still have the final say in what is being played. To help with this, our system includes upvoting and downvoting functions. Not only can the DJ get an understanding on which songs should be played, if enough people downvote a particular song, it will be removed from the suggested playlist. The system may in the future also include a setting where the DJ can allow the most upvoted songs to automatically be at the top of the playlist queue in order to appeal to the largest group of guests.

The system takes information from multiple music providers. The system should not limit the user to selecting songs from one or another because alone they are limited. The system will ideally use Spotify, Pandora, and iTunes library as a starting point because together the magnitude of available music is vast. The negative to using multiple providers is that it will be difficult to implement and possibly authenticate.

During the entire building process of our system we used GitHub [9] for version control, documentation because it helped us save and keep track of our progress and back track if there was a mistake that caused us to take a step back.

Chapter 7

Technologies Used

Listed below are many of the technologies we used to develop PowerPlaylist. They encompass the languages we used to write the code, any frameworks we used, and any other tools we used to help create the system. All of these listed technologies ended up playing key roles in the making of the project.

7.1 HTML 5

HTML [10] is a very standard web development tool that allowed us to create the structure of our web application. HTML5 specifically allowed us to play music out of the browser with the audio tag, which ended up being key to the project as it allowed for the host to also act as a client. By doing so, the host can use either a computer or a phone to manage the playlists.

7.2 CSS

We used CSS [11] for styling and aesthetics within our user interfaces. With CSS we were able to make an intriguing and intuitive user interface both for the host and the guests.

7.3 Javascript

Javascript [12] was used in order to conduct front end functionality for the web application. Any of the functions directly related to the user interface was done using javascript or JQuery.

7.4 JQuery

JQuery [13] was an important part of our project because it allows for easy DOM manipulation. If we needed to generate list items, like a song on a playlist for instance, we implemented JQuery to do so. JQuery also allows for browser compatibility and allowed us to use AJAX requests.

7.5 AJAX

Through JQuery, we used AJAX requests in order conduct communications between the clients and the server. The AJAX requests were encoded and decoded as JSON messages which are efficient. In addition to allowing for requests to the server, written in PHP, AJAX calls allowed us to update specific parts of our web page without rerendering the entire page and wasting battery life.

7.6 PHP

We used PHP [14] to serve a back end or server side code. Music data is abstracted with PHP and it is used to connect to music providers through endpoints. The AJAX request were made to the PHP in order to receive information on the server. The PHP also wrote playlist data and general party data, such as PINs to text files stored on the server.

7.7 MAMP

We used MAMP [15] as a local host server in order to build and develop the system on our laptops. It allowed us to run PHP and server side code easily to allow for debugging and alpha testing throughout the implementation process.

7.8 Amazon Web Services

Once the system is completed, it will need to be hosted on a commercial web server. Amazon web services [16] will allow us to do this and it can process a lot of requests efficiently which works well with our system.

7.9 Adobe XD

We used Adobe Experience Design (Adobe XD) [17] to create the user interface and prototype of our platform. Adobe XD allowed us to create each visual element individually then bind them together in various orders to create an interactive prototype of our system.

7.10 GitHub

We used GitHub for version control and storing important documentation such as source code, images, and Adobe illustrator files.

7.11 LaTeX

We used LaTeX [18] for creating our Design Document, User Manual, and Install Guide because it formats our documentation in a clean, presentable manner that is easy to read and easy to create.

Chapter 8

Timeline

This prototype was the result of a year's work, beginning with the initial design decisions in May, 2016. From then on, we mapped out the design and functional components of the system we were hoping to build. This planning process was primarily Fall quarter, 2016. In January, 2017, we began writing code and creating the initial prototype, with the final prototype completed by May, 2017, where we presented our project at the Senior Design Conference. For the future, we plan on implementing user accounts, and also hosting our site on a commercial web server so it can be used by anyone.

8.1 Spring Quarter 2016

- Initial plan, ideas for user interface and functionality, setting up an advisor for the upcoming year.

8.2 Fall Quarter 2017

- Creation of our initial Design Document, which outlined how we were going to proceed with our implementation and its requirements, the architecture of our prototype, user experience with our platform.

8.3 Winter Quarter 2017

- Initial implementation of the platform, starting with the user interface and connecting it with the backend functionality.
- Researching the APIs of Spotify, Soundcloud, and other music providers.
- Trying to ensure our architecture was consistent throughout the platform to ensure seamless

integration between the API's we would be connecting with and the front end and server functionality.

8.4 Spring Quarter 2017

- Completion of the front end and server implementation.
- Continued research on the APIs for different music providers.
- Testing the usability among our team members and peers.
- Senior Design Conference where we won Best in Session.
- Final Design Document outlining our completed platform.

Chapter 9

Risk Table

In Table 9.1 we have outlined the risks that threatened our progress during the year, and how we initially thought they would impact us, with the resulting impact in the 4th column.

Risk	Consequence	Probabiltiy	Severity	Impact	Mitigation Strategies
Lack of coding knowledge	Took on tasks beyond what we have learned in class	.7	8	5.6 - Delayed us greatly	Researched the languages we will be writing our system in, Work together on each section to help teammates along with implementation, Meet with advisor consistently for help.
Lack of time management	Team does not leave enough time to complete time consuming tasks	.5	7	3.5	Fell behind winter quarter, made up for lost time spring quarter
Illness	One or more of the teammates get sick and cannot work on the project for a certain period of time	.6	4	2.4 - actual 0	Was not an issue for our team
Lack of cooperation from outside APIs	APIs from different music providers do not fit well with the system or are too difficult to implement	.5	5	2.5	Did lots of research on each API, and chose to implement the usable APIs. Currently we are still in progress
File loss	Loss of progress in code and documentation of system	.1	10	1 - Actual 0	We used GitHub for versioning and saving our code, nothing was lost
Ambiguous Requirements	Lack of understanding about what the customer wants or failure to meet our personal requirements	.3	7	2.1 - actual 1	Communicated consistently with our team and friends to ensure we were all working towards the same goal and meeting our personal requirements
Miscommunication	Lack of communication between teammates causing less time to meet up or aspects of the project to be incomplete	.4	7	2.8 - actual 1	Met consistently, twice a week with our advisor and five times a week as a group

Table 9.1: Risk Table

Chapter 10

Test Plan and Results

Our test plan for the prototype consisted of Alpha testing, Beta testing, and Stress testing, to ensure our system was functional in a variety of environments and settings.

10.1 Alpha Testing

Our test plan started with Alpha Testing, where we, as a group, used the platform vigorously to check for edge cases, anomalies with with the user interface, and other problems that could occur with our front end or saving to the server.

10.2 Beta Testing

We then moved on to Beta Testing where we brought the user interface to our friends and peers to get feedback on what was working and what was not, as well as making sure no edge cases were missed during our Alpha testing.

10.3 Stress Testing

We initially thought our stress testing would be to ensure the server being run on the hosts computer could handle the number of requests being made, however since we have decided to host our application on a commercial web server this is no longer a concern of ours. Instead, we tested how well concurrent requests from users were handled, and if they would interfere with the suggested playlist in any way, or cause any problems with our update function for the host and guest.

Chapter 11

Societal Components

Every senior design project at Santa Clara University touches the community in some way. Below we explore the interaction PowerPlaylist has in society, as well as its greater impact in our community.

11.1 Ethical

An ethical issue that we focused on early on in the development of our system was whether or not it was legal or ethical to connect our system to music provider services. We knew that we wanted to use Spotify's songs and authentication in our system, but we wanted to make sure that we did so legally. The way that we ensured we were handling this situation ethically was by making only the host authenticate with the music provider. That way the host is not unintentionally breaking copyright laws, and guests are only suggesting songs available from the host, not actually selecting the songs from Spotify.

11.2 Social

We envisioned PowerPlaylist as a tool to bring the community together to collaborate on music. We created a collaborative playlist so that the host and the guests of a party can work together. We added the upvote and downvote features to encourage interaction between members of an event. We feel that PowerPlaylist is a tool that connects the community over a shared love of music.

11.3 Political

While our project is not political in nature, it still has an impact on society. This project brought team members from different backgrounds together to share skills and perspectives so that everyone could play a role that matters.

11.4 Economic

Engineering costs were very low for this project. We received no funding for this project and we spent no money creating it. We used our personal laptops to create this project, however, so that is an economic consideration. Guest and hosts may use this system for free, so this system is very economical for everyone.

11.5 Health and Safety

Our system has no health or safety issues other than making sure that the users use our system in a safe space - which is outside the scope of our project. We do want to make sure we treat users data in a safe and private manner and not share this information with outside parties.

11.6 Manufacturability

Engineering this system is time consuming, but it is very cheap. This product just requires a laptop and server to be built. Because our system is a web application, a native mobile app does not need to be downloaded by the guest, it can be easily used by anyone with a mobile device.

11.7 Sustainability

Our system will only remain sustainable if we add adjustments to our code over time. Technology is changing at a rapid rate and the devices we designed our system to run on may soon become obsolete. Therefore, our system must be maintained over time. However, our system will be able to be used for quite a few years until the technology mentioned earlier changes. In a broader sense of sustainability, our system does not help a community live more in a more sustainable way.

11.8 Environmental Impact

Our system has a very small environmental impact. It was created with only a laptop and time. However, our system does have an environmental impact to consider: the digital tools used to create PowerPlaylist contribute to harmful electronic waste all around the world, which is out of the scope of our project.

11.9 Usability

Usability was a great concern for this project. We wanted to make the system intuitive and natural to use. We made all design choices with our particular audience in mind. For example, we know that our audience will potentially be using our system in low light situations. Because of this we designed our interface to use dark colors and high contrast. In addition, we know that searching for songs is one of the most important tasks of a guest. Therefore, we placed the search bar in the front and center of the page. Each design choice had the audience in mind to aid usability. In the future, we would like to perform large-scale usability testing on our system.

11.10 Lifelong Learning

This project inspired each of us to learn new material in a rapid time frame. We became much more confident in web technologies such as javascript, jQuery, and AJAX. We also adopted a new design tool, Adobe XD, to create our initial prototype. We build upon the foundation that we learned in our computer engineering classes here at Santa Clara University and learned new things every time we sat down as a group to work on PowerPlaylist.

11.11 Compassion

Over our four years together at Santa Clara University the Engineering School has instilled in us the importance of compassion. We know that it is our most important job as engineers to detect areas of suffering or need and build solutions that aim to relieve that suffering, or give a solution to an existing problem. Though our system is modest and does not relieve suffering around our community, it still strives to be a source of positivity at Santa Clara. Our system has compassion because it is inclusive and allows everyone to collaborate together in a positive environment dedicated to music.

Chapter 12

Lessons Learned

Implementing the client-server architecture was more complicated than we were anticipating, especially given the number of features we wanted to include for both host and guest. We needed to make sure the the frequency of requests was not a burden on anyone's device, and that ended up being more complicated to create than we originally thought. Because of this complication we learned a lot about server implementation, and the minor details that go in to ensuring a seamless system.

We are also glad that we chose to do a project that we were interested in because we were able to devote a lot more time and energy to completing this project. Since it was something we all felt we would like to use in our lives and share with our friends, we wanted to ensure we produced a product that was user friendly and functional.

Chapter 13

Conclusion

For our senior design project, which built upon the foundations that Santa Clara University set for our team, we created from scratch a system that allows guests to search and request songs at an event, and gives the host power to play those songs. Throughout the course of our senior design project we learned some very important lessons. First of all, our team learned that the foundation that the computer engineering department at Santa Clara University has allowed us to develop products with meaning. A second important lesson our team learned was that it is important to leverage the unique skills that each team member has in order to create a multifaceted platform. Our platform, PowerPlaylist, has many advantages. Firstly, It was cheap to create and free to use. Secondly, there is no need to download software to use our system. Because each playlist has a URL, the host can post it on social media and get song suggestions before an event begins. In addition, the host has ultimate control of what music is being played, so they still have authority over an event. Lastly, members of our community have shown great interest and want to use our product right away. Despite these many advantages, our solution is still lacking in a few ways. There are minor bugs that need to be fixed, our system is currently not very scalable, and we would like to connect our platform to multiple music providers such as Spotify, Pandora, or iTunes. Looking forward, the next steps for PowerPlaylist are to connect music providers to our system, create measures to improve the scalability of our platform, and to perform usability testing to make sure we are designing the most intuitive and valuable system we can.

Chapter 14

Appendix A: User Manual

How to create a new playlist:

1. On the authentication page, hosts click the button that says Hosts Click Here.
2. Host enters details about your playlist, such as playlist name.
3. System generates a pin code for host to share with guests.

How to invite guests to join the playlist:

1. When the host creates a playlist, PowerPlaylist will automatically generate a pin authentication code.
2. Hosts will give this 4 digit code to guests so they may enter it on the pin authentication page and can access the playlist.

How to request a song:

1. Guest or host clicks the white search bar in the top center of the homepage.
2. User types in the song name and press enter.
3. A drop-down list appears with matching song titles.
4. User clicks on which song to request from the drop-down menu and it is automatically requested.

How to approve or reject a selected song as a host:

1. When a guest requests a song, it is filtered into a queue called Suggested Songs.
2. This Suggested Songs list is visible to hosts on their homepage.

3. Next to each song is a check mark that says approve or an X that says reject.

4. Users click the check mark to approve the song and place it in the Up Next playlist, or click the X to reject and delete a song.

How to upvote or downvote a song:

1. When a song is suggested and approved by the host it will be placed in the Up Next playlist on the homepage of PowerPlaylist.

2. Next to each song title is an up arrow or a down arrow and a counter variable.

3. User clicks the arrow pointing up to upvote a song. The counter will increase by one.

4. User clicks the arrow pointing down to downvote a song. The counter will decrease by one.

Chapter 15

Appendix B: Install Guide

How to install PowerPlaylist:

1. Hosts will go to www.powerPlaylist.com

2. Create an account with Username and Password

3. Begin authenticating with music providers to connect to your account

Bibliography

[1] "Music for Everyone." Music for Everyone - Spotify. N.p., n.d. Web. 09 June 2017.

[2] "About Pandora." Pandora - About Pandora. N.p., n.d. Web. 09 June 2017.

[3] "ITunes." Apple. N.p., n.d. Web. 09 June 2017.

[4] Beal, Vangie. "Client-server Architecture." What Is Client-Server Architecture? Webopedia Definition. N.p., n.d. Web. 09 June 2017.

[5] "Ajax (programming)." Wikipedia. Wikimedia Foundation, 22 May 2017. Web. 09 June 2017.

[6] "What Is Peer-to-Peer Architecture (P2P Architecture)? - Definition from Techopedia." Techopedia.com. N.p., n.d. Web. 09 June 2017.

[7] "Object-oriented Analysis and Design." Wikipedia. Wikimedia Foundation, 07 June 2017. Web. 09 June 2017.

[8] "Distributed Blackboard Architecture." Distributed Blackboard Architecture. N.p., n.d. Web. 09 June 2017.

[9] "GitHub." Wikipedia. Wikimedia Foundation, 08 June 2017. Web. 09 June 2017.

[10] "HTML5." Wikipedia. Wikimedia Foundation, 08 June 2017. Web. 09 June 2017.

[11] "Cascading Style Sheets." Wikipedia. Wikimedia Foundation, 09 June 2017. Web. 09 June 2017.

[12] "JavaScript." JavaScript.com. N.p., n.d. Web. 09 June 2017.

[13] Jquery.org, JQuery Foundation -. "JQuery API." JQuery API Documentation. N.p., n.d. Web. 09 June 2017.

[14] "PHP Manual." Php. N.p., 08 June 2017. Web. 09 June 2017.

[15] GmbH, Appsolute. "MAMP and MAMP PRO." MAMP and MAMP PRO. N.p., n.d. Web. 09 June 2017.

[16] "AWS Free Tier." Amazon Web Services, Inc. Amazon.com, n.d. Web. 09 June 2017.

[17] "Adobe Experience Design CC (Beta)." Adobe. N.p., n.d. Web. 09 June 2017.

[18] LaTeXProject Team. LaTeXfor Authors. N.p.: LaTex, 31 July 2001. PDF.