**Synchrony**

Initial Report Write-up

Jacob Alspaw

Alex Hemm

Owen Helmstetter

Ted Timbrell

Bobby Wagner

Synchronized streaming services provided by a dedicated team aiming to enhance the music listening experience of users through shared playlist collaboration and discovery.

**Version 1.0**

September 21, 2018

**Table of Contents**

[**1. Introduction**](#_8rhjvvyeg1p) **2**

[1.1 Purpose](#_9anjco2t4uo8) 2

[1.2 Vision](#_urteur9js5hh) 2

[**2. Features**](#_ejiqbtkj3k2r) **3**

[2.1 Lobbies](#_ufkxyer4hyvr) 3

[2.2 Streaming](#_jfxbbevtt98a) 4

[2.3 Real Time Communications](#_cdtwr2d91o7w) 4

[2.4 Stretch Goals](#_2fgsd71w64f) 5

[**3. GUI Mockups**](#_lb6ar0vv8bmt) **7**

[**4. Languages / Tools**](#_iwuzr71izzs3) **11**

[4.1 Frameworks](#_5r24vudrbxv6) 11

[4.2 Tools and APIs](#_drzbdm477nxu) 11

[**5. Architecture / Design**](#_aes2sbfuunr9) **12**

[5.1 Application Architecture](#_dl11jcwzqeat) 12

[5.2 Client-Side Architecture](#_rqwjx040i7s1) 13

[5.3 Key Components](#_nyahx7my9uau) 14

[5.4 Database Design](#_szr7vag6y0b5) 14

[**6. Testing / Documentation**](#_is4s0pt23gvf) **15**

[6.1 Testing](#_yvquchxkn2wc) 15

[6.2 Documentation](#_y708kx5qx9ol) 15

[**7. Meet the Team**](#_eyifygkthr8o) **16**

[**8. Development Plan**](#_xvznbe1qhb7) **18**

[8.1 Schedule](#_my0u0uz55m87) 18

[8.2 Allocation of work](#_mnq90irn90u7) 19

[**9. Final Discussion**](#_unzsk0rbufc6) **19**

[9.1 Risks and Mitigation](#_bxki89buubk) 19

[9.2 Scalability and Design Constraints](#_ufpr98nz6mdp) 20

[**10. Conclusion**](#_d9y2q4jlvw2w) **20**

## **1. Introduction**

### **1.1 Purpose**

The music industry and the ways in which we listen to music are rapidly evolving. The development and introduction of music-streaming services has given music listeners a new form of easy access to any and all types of music. Listeners use these streaming services in tandem with their devices to create the music listening experience many have come to know and love. Even though the devised system for listening to music often proves unproblematic, it is nowhere near perfect.

The current ways in which we listen to music are flawed. Standalone speakers often subject others to forced listening conditions. They carry a stigma in public areas where peace and quiet are expected thereby limiting their use. Audio from mobile phone speakers is poor or mediocre at the very least compared to most inexpensive headphone sets. High quality and cheap headphones have reduced the need for cumbersome standalone speakers in people’s everyday lives, however, headphones create a barrier between listeners and degrades the shared listening experience that standalone speakers promote. So how can we ensure that music remains a shared experience amongst many enthusiasts?

Improving upon the current music-streaming services is just one way in which we can ensure a shared experience amongst listeners. Current methods for contributing to the music listening experience lack a key feature. It is apparent that industry-leading music-streaming providers, like Spotify, have simply overlooked the needs for synchronously playing music amongst multiple users. These large services only allow a stream to be sent to and controlled by one person at a time. When the services limit stream access to a single person, they thereby limit the overall user experience. If stream access could somehow be given to multiple users, the user experience would dramatically improve with means for group listening and collaborative control.

### **1.2 Vision**

The next evolution in how we listen to music is Synchrony, a synchronized streaming service provided by a dedicated team aiming to enhance the music listening experience of users through shared playlist collaboration and discovery. Synchrony acts a wrapper to mainstream music providers, captures their streams, and then relays the stream to Synchrony users. Our service allows users to collaboratively listen to music in groups (even when wearing headphones), and enables control over the current song queue for each person in the group . Synchrony will allow users to keep using their individual devices and streaming services while effortlessly and seamlessly sharing their experience with friends.

Synchrony’s functionality is found in its many lobbies where users engage in interactive listening experiences. Users will be able to create and join lobbies, invite their friends, search for songs to add to the lobby queue, control soundtracks, and use real time communications. Because these lobbies offer shared synchronous playback, Synchrony acts as a fantastic solution to many problematic ways in which we listen to music today.

There are many ways in which Synchrony can be used. Playing music in public through speakers is considered a social taboo. Our platform would allow users to avoid disturbing others on public transport, or at serene beaches or parks where a cumbersome standalone speaker is either unavailable or inappropriate. Synchrony also enables users multiple controls over a stream. Is someone connected to your bluetooth speaker and playing their terrible music? Connect both phones to a Synchrony lobby and you can skip their music and add your own song suggestions to the queue!

## **2. Features**

### **2.1 Lobbies**

*Overview*

The primary feature for our application is listener lobbies. These lobbies are the main interface for users to have an interactive listening experience. Synchrony will allow patrons to join other lobbies or create their own lobbies. Listener lobbies will not only provide a graphical user interface that lets users interact with the audio streaming service, but it will also display other components such as the song metadata and real time communications.

*Creating a lobby*

If a guest clicks on a create lobby button, then the create lobby page will dynamically load. From this page, guests can select from a list of supported lobby types. Lobby types are limited by the number of streaming service APIs the development team will have access to. After the lobby type is selected, the guest can proceeds to create a new lobby of that type by hitting the continue button and they become the host of that lobby. A unique identification code is generated and the host can share this code their friends to allow them to join the lobby. The host will have to share this code by either showing other people in person or sending it through an alternative communication platform. The host will have control over the choice in audio tracks that are played and can control the position in the audio track.

*Joining lobbies*

If a guest clicks on a join lobby button, then the join lobby page will dynamically load. From this page, guests can enter the unique identification code associated with a lobby. If the code is correct, then the guest will be granted access into the lobby. If the code is incorrect, then the guest will be shown a validation error and will be asked to enter a different access code. Guests will receive this code from either the host of the lobby or any other user that is currently in the lobby.

*Leaving lobbies*

Hosts and guests will both be able to leave an active lobby. Once the user has successfully left the lobby, they will have the option to either join a lobby or create a new lobby.

*Kicking users from lobbies*

If a user is being abusive or spamming, hosts should have the option to kick the user from the lobby. If we have time, we will allow other users to vote to kick another user and have them automatically kicked after a given amount of votes.

*Setting guest alias*

When a user is joining a lobby, they should be able to pick a display name. Two users in the same lobby should not be able to have the same name.

### **2.2 Streaming**

*Overview*

The streaming service feature is responsible for delivering synchronized audio to each user in a lobby. The service will still be able to maintain the current time position on the audio track, and will display the song metadata obtained by the web API.

*Spotify API*

The Spotify API will serve as the source of audio for each listener lobby. Synchrony will use Spotify’s Web Playback SDK to serve songs on the client side and their standard Web API to play requested songs from the backend. Synchrony will use the web API to maintain the current position of the audio track and send this data to all listeners in the lobby.

### **2.3 Real Time Communications**

The final main feature Synchrony will implement is real time communications between listeners in the lobby. Real time communications refers to any live telecommunications that occur without transmission delays and will act as the interactive part of Synchrony’s listening experience. The real time communications that our development team will implement is a chat room that will allow users to write messages to a discussion board that is displayed to everyone in the lobby. The chat system will list the time the message was posted and the name of the user who posted it. Users can include messages that contain reactions to the current audio file or request another track or style of music to be played. When a new song begins to play, metadata about the song will be listed in the chat room.

### **2.4 Stretch Goals**

*Native support*

We want to bring Synchrony to as many users as possible, so extending our app to have native support for iOS and Android devices is, while a tough goal to achieve, something we would love to add.

*Song proposal voting system*

Since users may have trouble agreeing on which songs to play, we would like to provide support for polls between different song options, where all users in a lobby can vote on which song to listen to next.

*Additional music sources.*

For the time being, Synchrony only supports use of the Spotify APIs. Our development team would like to expand support to many more music streaming services like SoundCloud, SiriusXM, Pandora, Apple Music, and Amazon Music Unlimited. Doing so would allow users access to additional sources of audio increasing their options within a lobby.

*Switching between multiple audio sources*

With support for multiple sources of audio, lobby hosts should be able to choose which source to use when requesting songs. Should time permit, we will allow multiple streaming services to be enabled within one lobby. Users would be shown search results from multiple sources, offering a wider variety of songs to queue. This would require using both the Spotify and other APIs within the same lobby, and dynamically switching between APIs based on user requests.

*Guest access control for queue*

Allow the host to change a lobby's access control settings. This includes whether or not guests can skip, queue, or pause/play songs.

*Rich links in chat*

When users paste a link to a picture from an image hosting site such as imgur.com, embed that image in chat so users don’t have to open it in a new tab. Similarly, if a users posts a link to a song from a source supported by Synchrony, get the metadata about that song and include it below the link so people in the lobby can see the song title, artist, and album cover.

*Formatted text in chat*

Using syntax similar to markdown or HTML, allow users to make their chat messages bold, italic, or more depending on time constraints.

*Emoji support in chat*

Add an emoji keyboard so users can include emojis in their chat messages.

*Voice communications for lobby*

The only concrete requirement for real time communications is the text chat system, however, our team would like to further develop our real time communications and implement voice communications. Adding a voice communications system to Synchrony would require user access to a microphone, but would drastically improve upon Synchrony’s interactive and collaborative experience.

*Dedicated User Profiles*

Implementing user profiles would allow guests to be easily identified and make the experience more personal. The profiles would store information such as a name, other contact info, and genres of music they like to listen to. Other users would be able to view profile names in the lobby and be able to access their profile page that displays the profile data.

*Friends list*

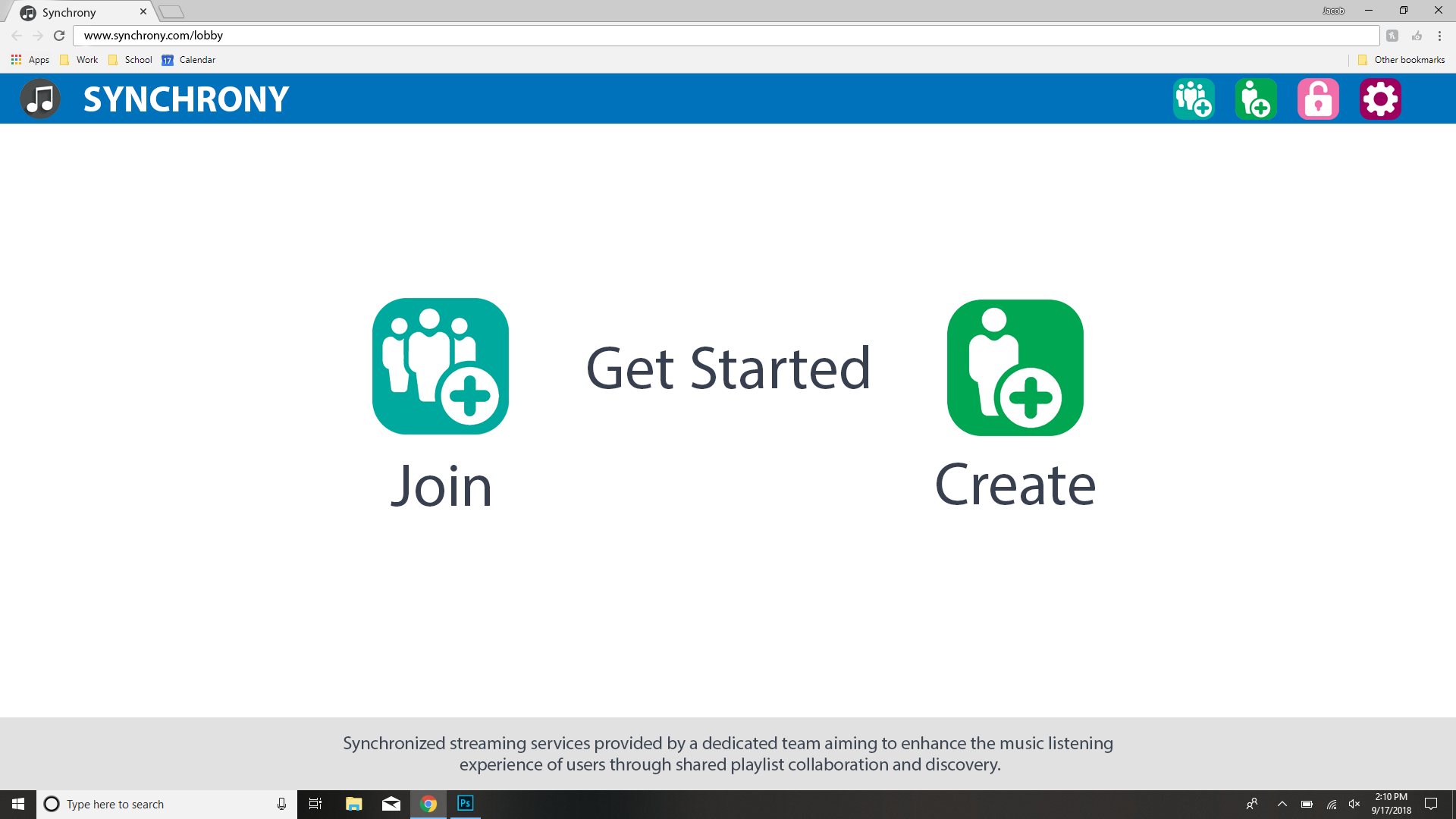
Assuming we decide to implement user profiles, allowing users to add friends would ease the process of joining a lobby. Friends could join another friends lobby without the need for an access code. Users could see what friends are online and what songs their friends are listening to in their respective lobbies.

## **3. GUI Mockups**

Below you will find brief user interface mockups of what the team anticipates are going to be Synchrony’s most visited pages. Synchrony’s pages include, but are not entirely limited to home, join, create, lobby, login, and settings. Lesser visited pages were not considered for the mockups.

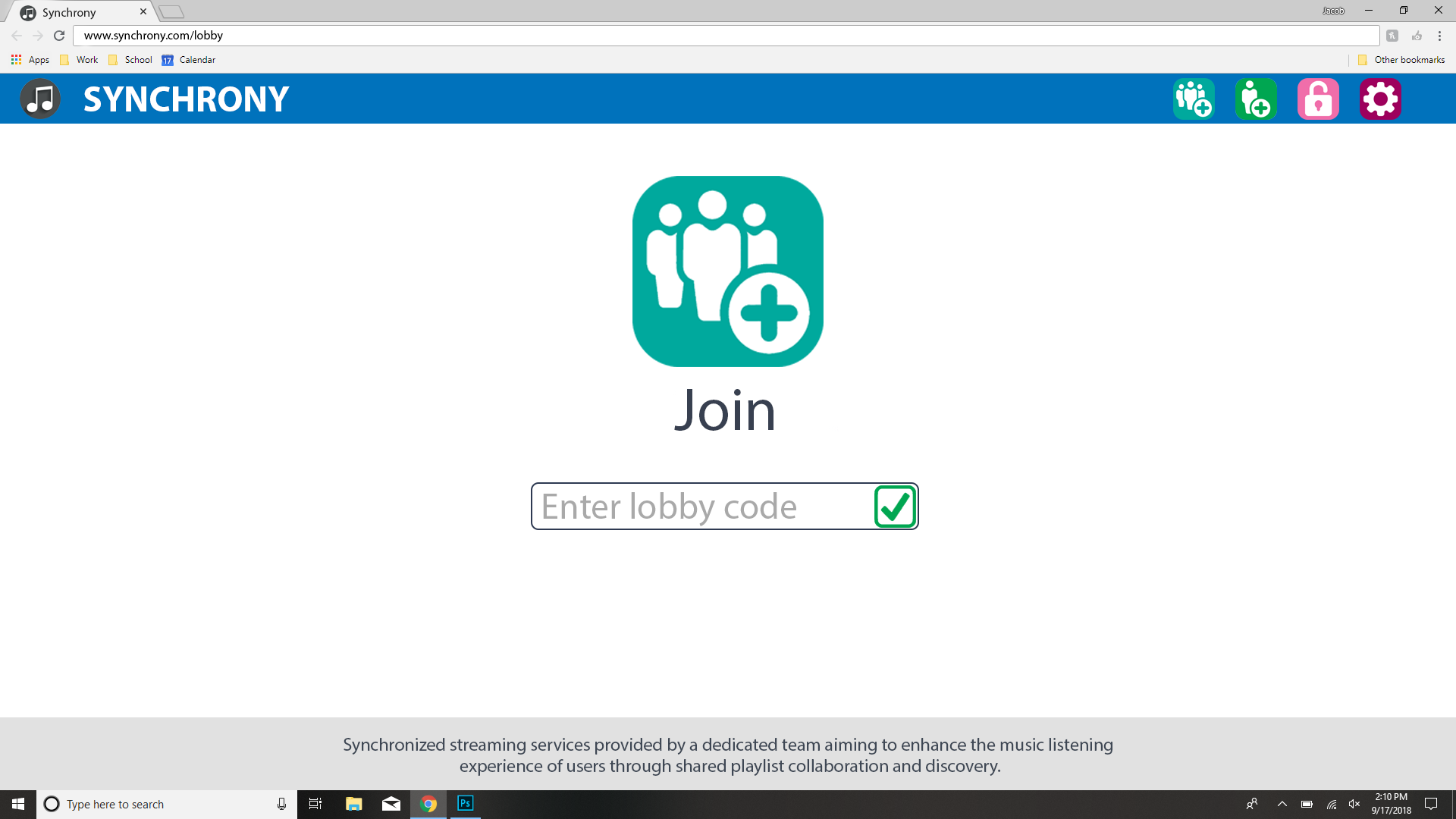
*Home Page*

The page guests will see when the base URL is visited. The top banner is part of website’s layout and will remain consistent regardless of what page a guest visits. This banner will show the service name and icon, a join lobby button, a start lobby button, a login button, and a settings button. The remaining portion of the layout page describes our service using our current mission statement. Guests will be greeted with additional options to join or create a lobby to expedite and ease the process. Figure 3.1 shows a mockup of what the home page will look like:

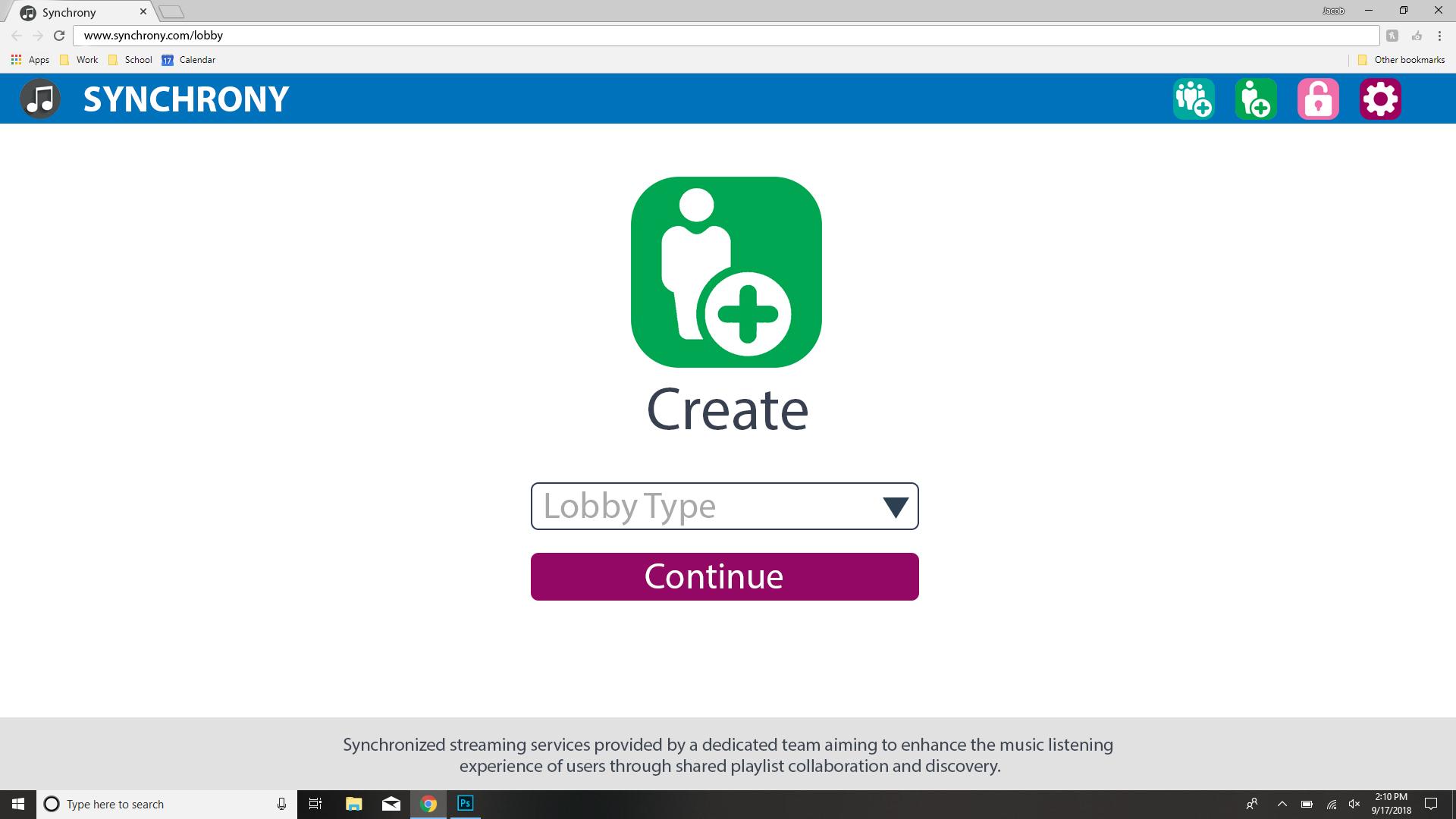


*Figure 3.1, Home Page*

Figures 3.2 and 3.3 show the mockups for joining and creating a lobby:



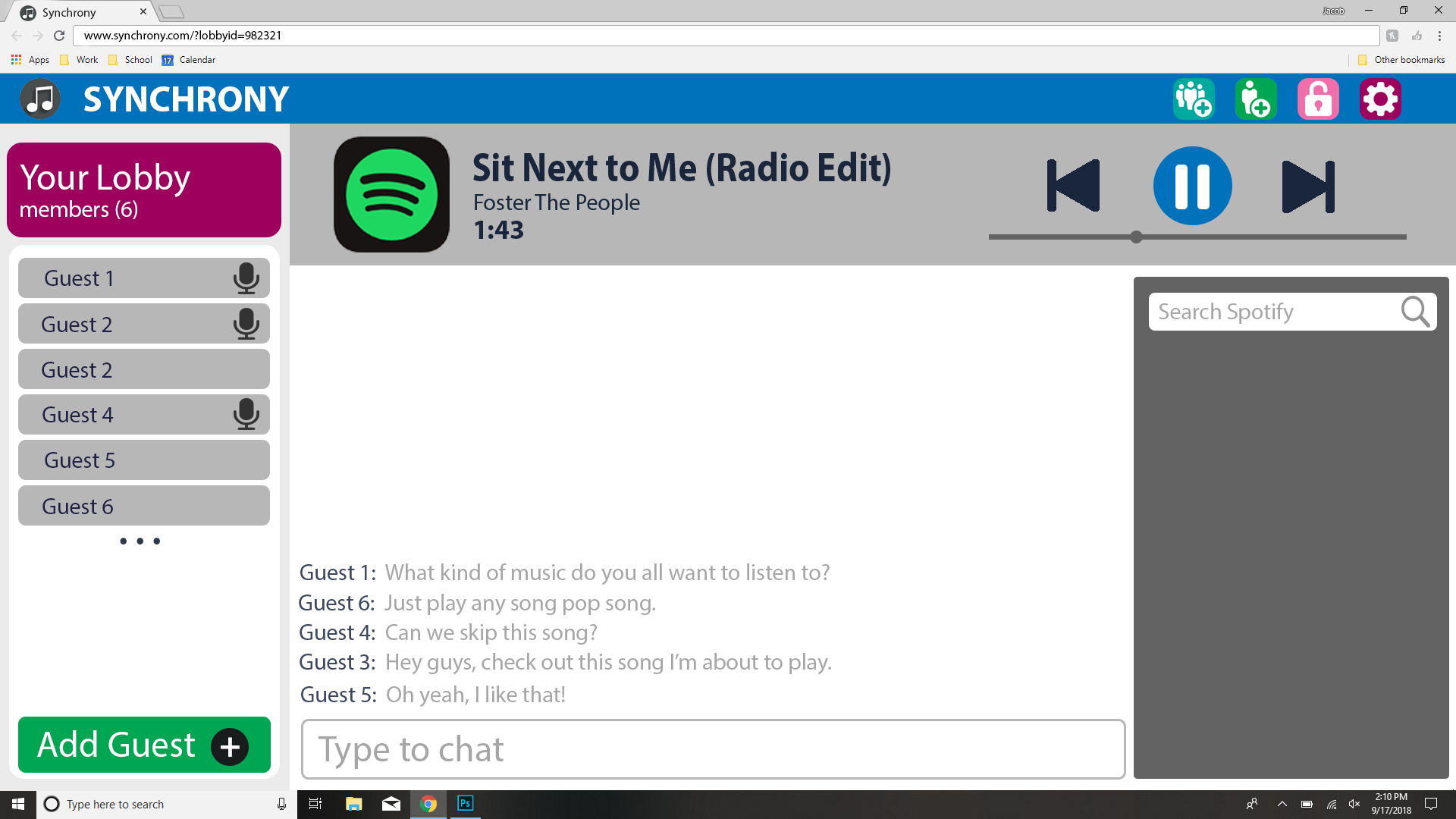
*Figure 3.2, Join Lobby*



*Figure 3.3, Create Lobby*

*Typical Lobby*

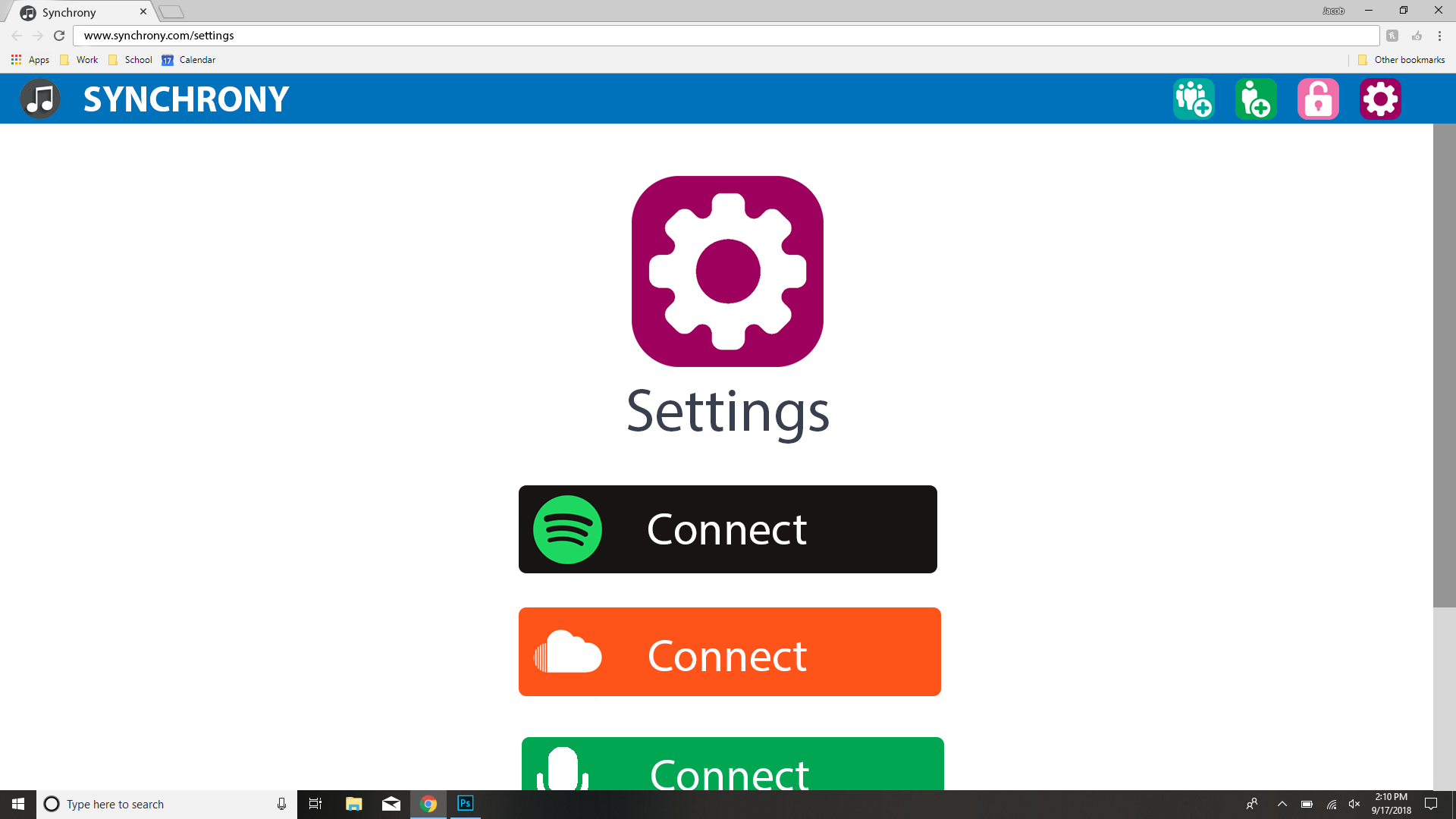
Once a lobby has been entered, you will notice four main components. The left side bar contains basic information about the lobby: the members and individual member information, the lobby type, and the unique lobby access code. The add guest button will allow lobby participants to invite new members by copying the unique lobby access code to their clipboard. Lobby participants are required to distribute the code to potential new guests. The middle of the page shows the chat log and allows lobby participants to send messages with a possibility of full emoji support. The right sidebar is used to search for music and add songs to the current queue. The search results are dependent on lobby type and search text. The top banner will show the currently synchronized song along with a set of controls to play, pause, and skip tracks, as well as control the part of the song playing through the use of a slider. Figure 3.4 shows a mockup of what a typical lobby would look like:



*Figure 3.4, Typical Lobby*

*Settings*

If a guest clicks on the settings button, then the settings page will dynamically load. From this page, guests will be presented with a set of configuration options for their Synchrony experience. The options include, but are not limited to setup of their Spotify account, other music streaming service accounts, microphone, and guest name. Figure 3.5 shows a mockup of what the settings page would look like:



*Figure 3.5, Settings Page*

## **4. Languages / Tools**

### **4.1 Frameworks**

Synchrony will use python’s Django framework for the back-end, and the latest version of Angular (6.1.7) for the front-end. We chose Django because most of us already know python or will be able to learn it quickly, and Django is designed to require less code and development time than other back-end options. We chose Angular for the front-end because we knew we needed a single page application framework but none of us had any experience in Node.JS, and our research suggested that Angular would allow for more code reuse across different platforms and make it easier to add native mobile support later on in the development cycle with NativeScript, since we decided to shoot for native Android and iOS support as a stretch goal. As is standard for Angular development, we will use TypeScript and RxJS in lieu of JavaScript and promises.

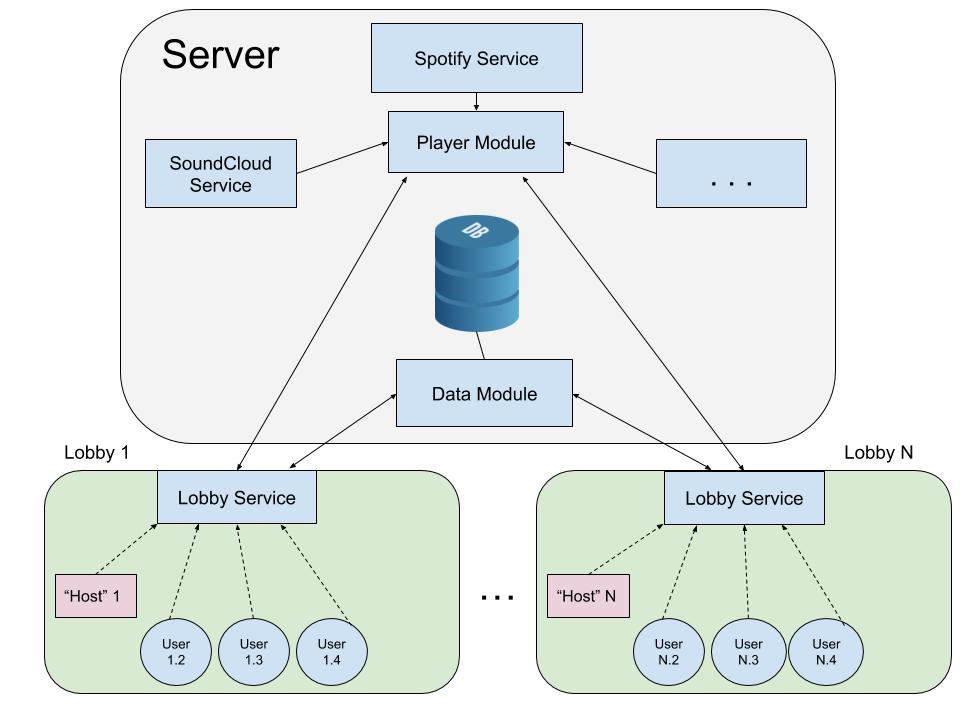
### **4.2 Tools and APIs**

For package management, we will use npm on the front-end and pip on the back-end. The client side of the project will be built with the angular-cli. A connection to the Django server, which should be always running, will be made during initialization. Spotify is our first choice source for song playback, so we will use their Web Playback SDK to serve songs on the client side and their standard Web API to play requested songs from the backend. We consider playback from other audio sources as a stretch goal.

## **5. Architecture / Design**

### **5.1 Application Architecture**

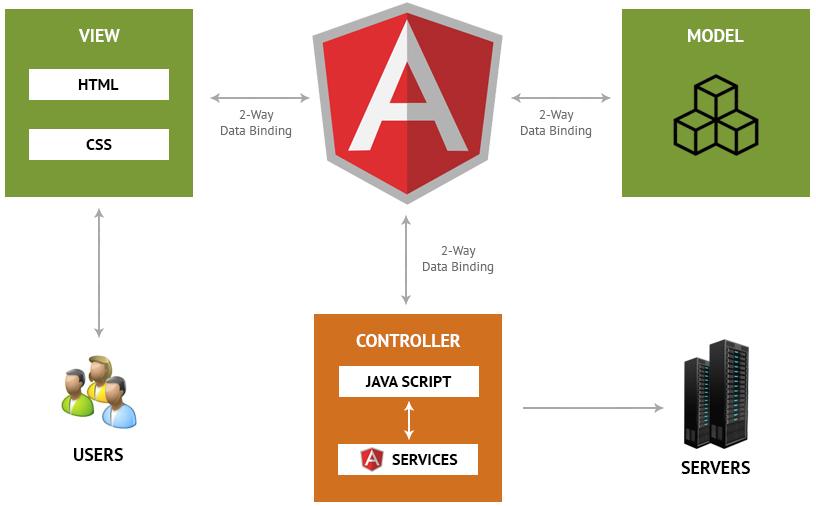
The high level architecture for Synchrony will follow the client-server model. The server will handle API calls for songs and synchronize playback, and any given client will pass its requests off to an instance of the lobby service, which will calculate playback delays, keep track of current users in the lobby, and sync the lobby’s status with the player module on the server side.



*Figure 5.1: The architecture of Synchrony*

### **5.2 Client-Side Architecture**

The client side of Synchrony will follow the model-view-\* based architecture used by the Angular framework, where the model is a *class/interface*, the view is a *template*, and the controller/viewmodel/etc. is a *component*.



*Figure 5.2: Architecture for the Angular client*

### 

### **5.3 Key Components**

Components are the basic building blocks of an Angular app. Each component defines:

* Any necessary imports needed by the component
* A component decorator, which includes properties that allow you to define the template, CSS styling, animations, etc..
* A class, which is where your component logic is stored.

Here are the main components that will come together to make the lobby page, which holds the app’s main functionality:

song-bar.component

* Display info on current song, controls for play/pause and skipping tracks, etc...

chat.component

* Chat history and message bar for the user to enter text

search.component

* Search bar for users to enter song queries, displays results below the search bar

lobby-info.component

* Show who is in the lobby and the lobby name or invite code

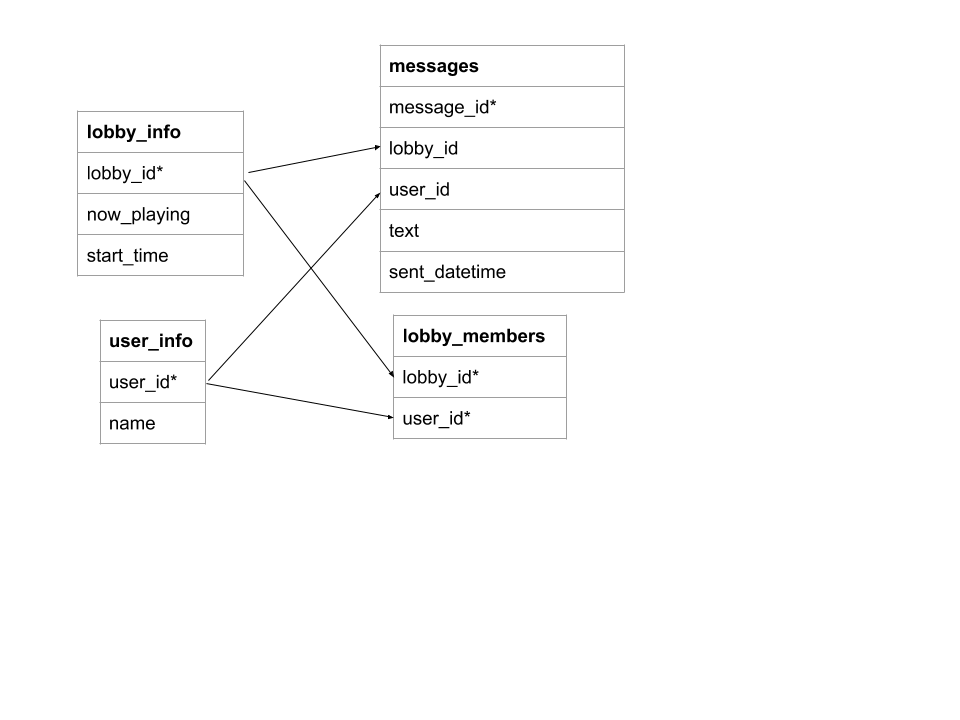
title-menu-bar.component

* Display the name of the app, Synchrony, and buttons to open things like the settings menu

### 

### **5.4 Database Design**

Our database will run on postgresql which is an object-relational database. It will be connected to the django backend. It contains 4 tables, lobby\_info, user\_info, lobby\_members and messages. Primary keys are denotes in the following drawing by a \*. Foreign keys are denoted by an arrow.



## **6. Testing / Documentation**

### **6.1 Testing**

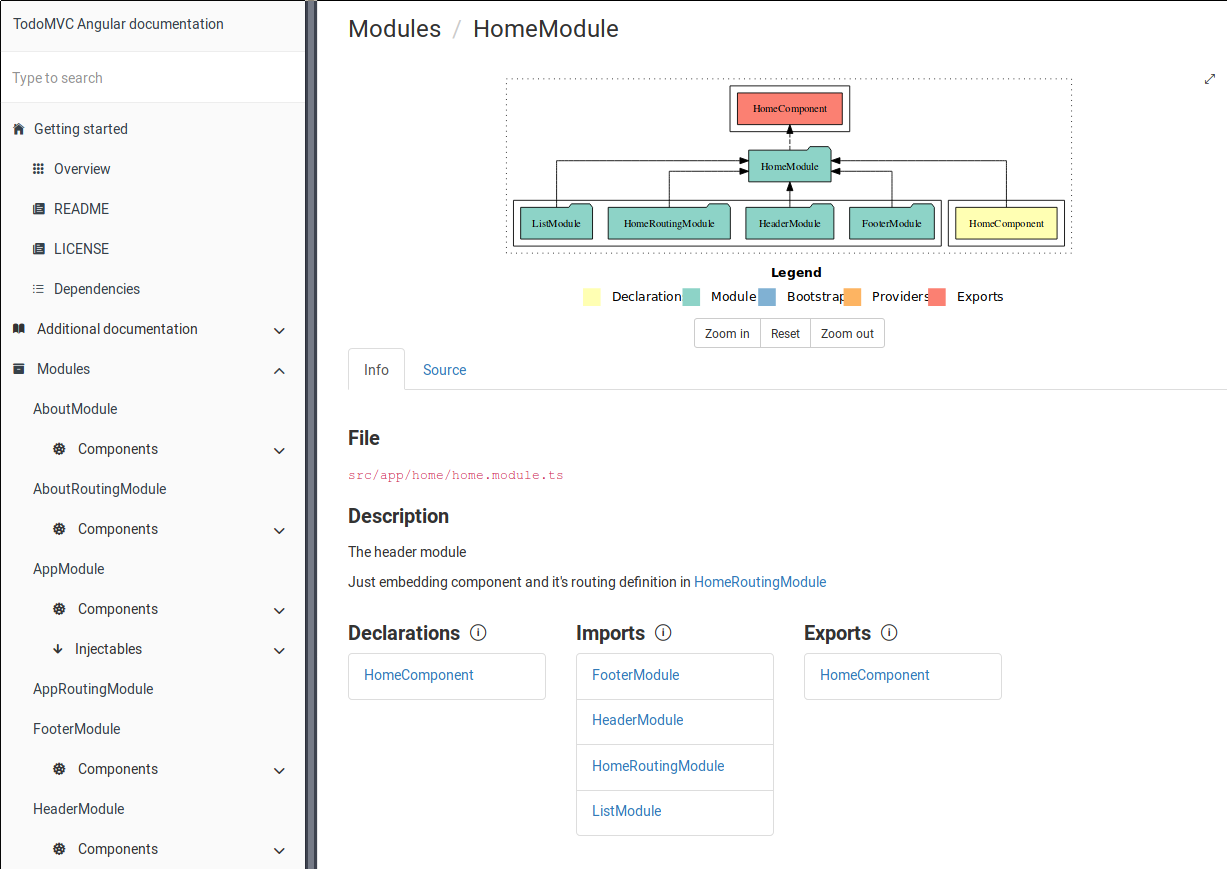
Each method in the backend will be tested using Django’s built in testing tools which are based on the unittest module in python3. We will generate coverage reports using coverage.py.

Testing the frontend will be done using the [Jasmine test framework](https://jasmine.github.io/) and the [Karma test runner](https://karma-runner.github.io/2.0/index.html). With these tools, we can run ng test in the command line to automatically execute all tests and view a summary of the results, and we can append the flag --code-coverage to see our coverage metrics. Using the Angular CLI for testing means that any time a change to the code is saved our tests will automatically start to run in the background. As a result, we avoid the need to set up any sort of test automation.

### **6.2 Documentation**

For Django we will generate documentation using django’s admin docs model. This will allow us to write all of our backend docs as a docstring allowing for easy viewing during development as well as version control of our documentation.

To generate documentation for the frontend, we will use [Compodoc](https://compodoc.app/guides/getting-started.html), because it allows us to write comments in the standard JSDoc style and then generate fancy Angular-specific documentation with a single terminal command and serve it as a web page.

****

*Figure 6.1: an example of documentation generated by Compodoc*

## 

## **7. Meet the Team**

*Jacob Alspaw*

Jacob has industry experience using the .NET Framework and C#, HTML5, CSS3, JavaScript, TypeScript, and SQL. He has a working knowledge of C, C++, Java, and Matlab. Jacob excels at web-development and accredits his knowledge of the subject matter to a software development company called Pointe Blank Solutions. Located on the west side of Cleveland, Jacob worked as a Software Engineer Intern from May of 2016 until August of 2018 where he routinely developed new features and performed system maintenance using C#, ASP.NET MVC 5 Controllers, Knockout JavaScript, SQL Server databases, both server-side and client-side rendering systems, code ownership paradigms, automated deployment systems, version control systems such as Git and Mercurial Hg, and mass integration and unit testing.

*Alex Hemm*

Alex is well experienced experienced in Java and has basic experience in Python, C. Moreover, he is sure that he can quickly learn other object oriented programming languages. Alex has intermediate experience with HTML5, CSS3, and beginner experience with web MVC frameworks, specifically Spring MVC Thymeleaf. Alex has also taken EECS 393, a Case Western Reserve University course on all things to do with the software development process. In this course, a group of his peers nominated him team leader for a large team-driven project..

*Owen Helmstetter*

Most of Owen’s experience is in Java, C++, and Python, but he is learning to enjoy mobile and web development as well. His work experience includes speeding up the medical imaging software in a biomedical research lab, working on a backend service for an IDE, and later creating a demo app for that service using the Angular framework and a custom REST API. He is also interested in Linux and software engineering and was the team leader of his EECS 393 group.

*Ted Timbrell*

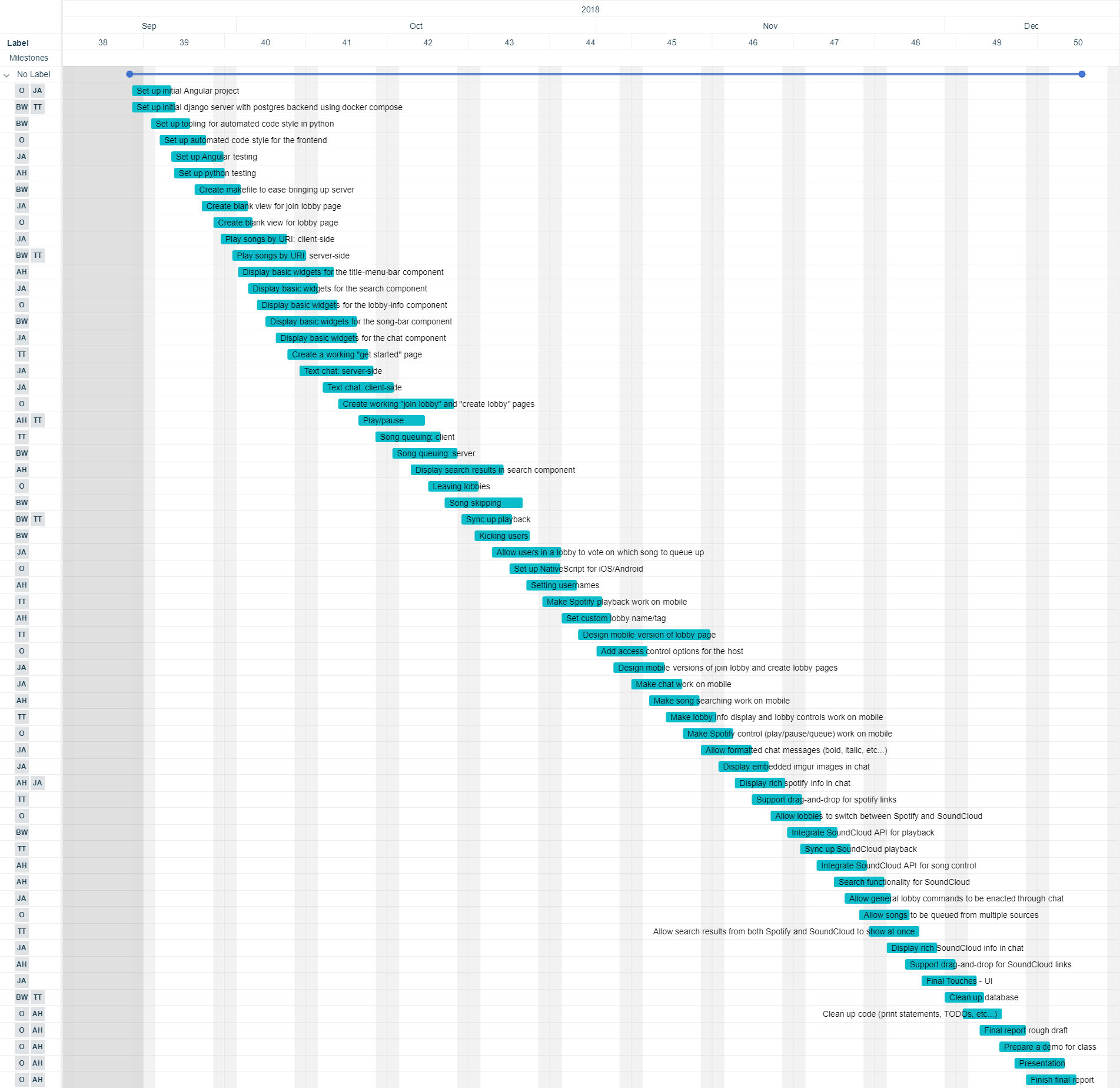
Ted mainly uses Python, but is also proficient in other object oriented programming languages like Java, C#, and C++. He finds comfort working in web development and database design using HTML5, JavaScript, CSS frameworks, PHP, SQL, PostgreSQL, and MongoDB (especially with Geo-Spatial data). Ted works for a machine learning lab during his free time. He has the required skill set to create a machine learning model if we need to use one in our project. Ted has advanced knowledge of Git, Subversion, and Slack which will surely help with project management. The development team accredits Ted with providing the project idea.

*Bobby Wagner*

Bobby has significant experience working with back-end machine learning development. He is proficient with python and has experience using tools such as Django, Pyramid, XGBoost and Spark at scale. He has also spent significant amounts of time with infrastructure tools such as Docker, Kubernetes, and Amazon Web Services. In his freetime, Bobby work’s on an open source privacy preserving machine learning library called OpenMined. The library is designed to be a drop-in replacement for PyTorch and Tensorflow while adding the ability to collaboratively train and evaluate without having to reveal the machine learning model or data to other parties.

## **8. Development Plan**

### 8.1 Schedule



The ownership of each user story is signified by the initials on the left side of the chart. “JA” stands for Jacob Alspaw. “O” stands for Owen. “BW” stands for Bobby Wagner. “AH” stands for Alex Hemm. “TT” stands for Ted Timbrell.

Source, for readability: <https://trello.com/invite/b/bfmVnImb/16b1cbab7d230538a4e01acb9e207a7d/senior-project>

### 8.2 Allocation of work

The allocation of work can be seen in the above Gannt chart. Each group member will focus on an individualized feature and will help elsewhere when needed. Jacob will be focusing on implementing all features for Synchrony’s real time communications. Owen and Alex will be focusing on our client side implementations. Bobby will prioritize setting up our backend services. Ted will switch between helping with the synchronization and testing on the server side and helping with the front-end development.

## **9. Final Discussion**

### 9.1 Risks and Mitigation

Synchrony presents a fair amount of risk when it comes to legality. The simplest solution to providing a synchronous streaming service is to have our servers download and play the audio to each of the users like a radio station or livestream. However, this would clearly be a breach of terms for most major platforms. Spotify says as much in section 8 of their terms of service, stating that, “‘copying, redistributing, reproducing, “ripping”, recording, transferring, performing or displaying to the public, broadcasting[...]’” is prohibited for any reason. Our strategy with Spotify is to conform to the terms of service by requiring users to either sign in or use publically available client-side functions. Our product will simply act as a coordinator. For any stretch-goal involving third-party services we will of course aim to conform to terms and copyright laws. In the absence of a feasible, fully-compliant solution, our strategy will be to design the platform as a protocol or relay rather than a distributor or broadcaster. This should, within reason, absolve our platform of direct legal issues and puts the onus on our users for their behavior.

Although we are legally allowed to use Spotify’s client-side API for non-commercial purposes, there is risk due to it still being in beta. There is potential that certain functions might change over the course of time. This will have to be mitigated by keeping a close eye on the Spotify team’s development progress and by making sure we’re not using any methods or functions that are going to be deprecated soon. The API’s core function is to more easily deliver the functionality that we need, so it is unlikely that we’d lose the resource entirely without a significant change in direction from Spotify.

Another source of risk is our ability to actually provide a quality synchronous playback. This might seem trivial but the quality depends highly on how we handle the synching of our listeners music. This almost becomes a philosophical issue as there is a tradeoff between one user experiencing the entire song versus lagging behind the others. This will require extensive testing and tuning in order to get it right. It will also require creative methods of dealing with the latency between our servers and the clients.

### 9.2 Scalability and Design Constraints

One incidental advantage to our choice of client-side playback is that it makes our platform relatively lightweight and highly scalable. On the server side, instead of transferring the full bandwidth of the music data we only have to send out semi-frequent updates to each of the clients about the state of playback. It also reduces the load on our database as we only have to store the relevant user data for each client and the generic state and timestamp for each lobby. On the client side the only addition is the query to our servers to check that they’re synced. Aside from that it’s no different from any other chat program or music streaming service in terms of resources needed.

The most likely source of issues with scaling will come from a lack of adequate bandwidth or processing power to serve the clients with their synch updates quickly enough given a certain mass of users. The nice part about this is that the critical number of users is independent on the number of lobbies. A lobby’s state is just a row in our database, so we could theoretically have all of those users in one lobby with little to no issues.

## **10. Conclusion**

Our development team came together rather quickly. Based on our online comments in the Canvas discussion board, it was obvious that each member shared a passion and appreciation for music. Music is an ever changing, always transforming *thing*, yet it always remains incredibly intimate and emotional. We wanted to contribute to this evolution process in some way that built upon listener relations. Our development team contemplated many ideas that we hoped could be the next big thing for music. Eventually, we settled on the concurrent music streaming service called Synchrony.

Synchrony was founded upon the ideals that bringing people together is pertinent in appreciating and enjoying the music listening experience. Most people have their own music listening preferences that they enjoy listening to, especially within our group. However, nearly everyone can agree that the art form is an important part of life, so we can express ourselves as human beings. By appreciating music, it makes it easy to unite and relate to others who are different than ourselves and thereby discover new music through mutual collaboration.

Synchrony is going to be a long term solution in uniting those around us in contributing and discovering new music. If even a few people enjoy the services provided by Synchrony or discover something new, then Synchrony’s development will have been worthwhile and meaningful, but we won’t necessarily have succeeded. Our development team is so truly dedicated to this service such that we don’t just think of Synchrony as a temporary project that stops production when we complete our final capstone deliverables. Instead, we see this project turning into something much larger that has potential for affecting the lives of thousands, perhaps more. We believe Synchrony can have a real lasting effect on the music listening experience, and our team will do our best to ensure the user experience is everything it can be and our vision is achieved.