Team No. 19

Team Members: Adam Wallace 2592181, Alex Wittman 2966955, Marcus Leong 2865056, Miller Bath 2817389, Clay Beabout 2820559

Project Name: Tuun

Project Synopsis

Tuun is a web application with the primary goal of supporting an algorithm-based service to make music listening more synchronized for multiple listeners.

Project Description

Project is being undertaken to ease the dynamic of diverse music tastes. The main objective would be to allow any social group to connect and instantly generate a playlist that pleases everyone.

Differing tastes in music makes it difficult to pick who should be in charge of playing music during social events. By pooling the top picks from each individual participating, we can allow everyone to enjoy the music being played. In addition to that problem, if you want to play relaxing or invigorating music there is no filter on Spotify to do so based on a user's liked songs. Another problem this project addresses is the fact that there is no way to select a genre of music based on a user's liked songs.

The end result of this project will be a web based application that will allow multiple people to join a room with their Spotify accounts. The application will collect data from their Spotify accounts and select music that everyone will be able to enjoy.

Project Milestones

- First semester:
    - Basic Web Server Implemented: 11/18
    - Communication with Server to Database: 11/25
    - Communication with Server to Front-end: 11/30
    - Basic Web Player: 12/1
    - UI for front-end complete: 11/28
    - Document Existing Code: 12/5

- Second semester:
    - Spotify authentication: 2/9
    - Server Logs API Responses: 4/1
    - Web Server Full Implementation: 4/21
    - Document Server-Side Code: 5/1
    - Server Logs Front-end Authentication: 5/1
    - Full Web Player: 3/1

Project Budget

- Hardware, software, and/or computing resources
  - Domain Name, Web Server Hosts, Basic Security Service
- Estimated cost
  - Domain Name: $40/yr
  - Web Host: ~$5/mo
  - Cloud flare: $20/mo
- Pluralsight subscription for new web technologies: $35/mo

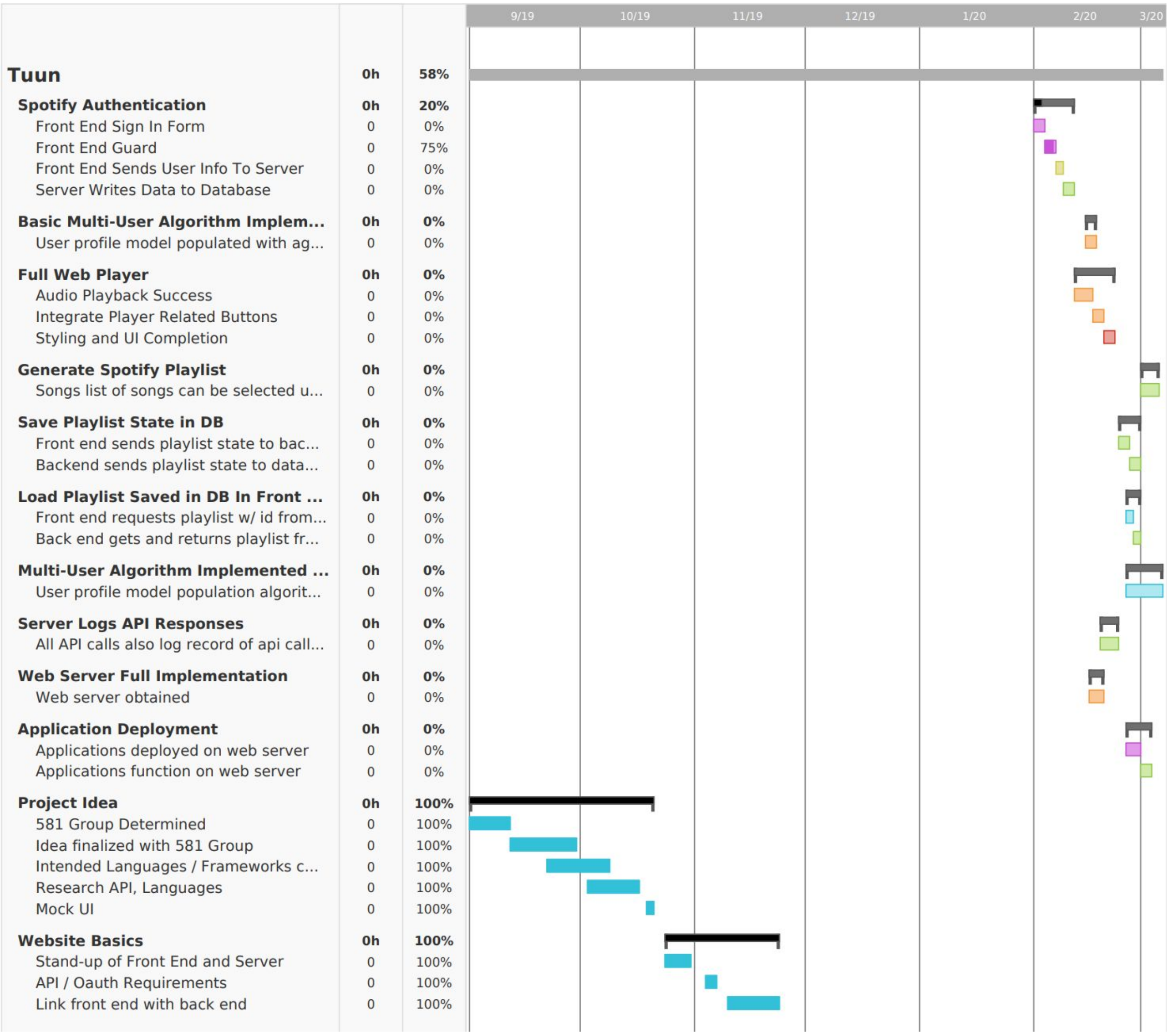
Total = $40 + (7 * $5) + (7 * $20) + (7 * $35) = **$460**

- When they will be required
  - **November 1st, 2019**

Work Plan

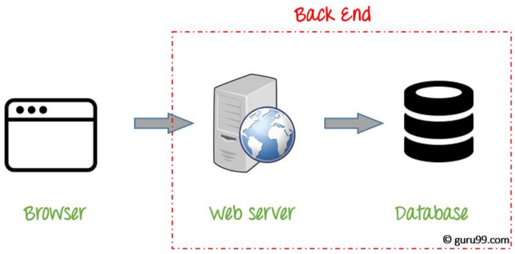
- Who will do what
  - Adam: Server-side, C#
  - Alex: Spotify & Apple Music API / Web Player
  - Clay: C# Expert [ULTRA PRO] [LEAD HACKER]
  - Miller: React & C#
  - Marcus: React & Authentication

Gantt Chart



Tuun	**0h**	**58%**
Spotify Authentication	**0h**	**20%**
Front End Sign In Form	0	0%
Front End Guard	0	75%
Front End Sends User Info To Server	0	0%
Server Writes Data to Database	0	0%
Basic Multi-User Algorithm Implem...	**0h**	**0%**
User profile model populated with ag...	0	0%
Full Web Player	**0h**	**0%**
Audio Playback Success	0	0%
Integrate Player Related Buttons	0	0%
Styling and UI Completion	0	0%
Generate Spotify Playlist	**0h**	**0%**
Songs list of songs can be selected u...	0	0%
Save Playlist State in DB	**0h**	**0%**
Front end sends playlist state to bac...	0	0%
Backend sends playlist state to data...	0	0%
Load Playlist Saved in DB In Front ...	**0h**	**0%**
Front end requests playlist w/ id from...	0	0%
Back end gets and returns playlist fr...	0	0%
Multi-User Algorithm Implemented ...	**0h**	**0%**
User profile model population algorit...	0	0%
Server Logs API Responses	**0h**	**0%**
All API calls also log record of api call...	0	0%
Web Server Full Implementation	**0h**	**0%**
Web server obtained	0	0%
Application Deployment	**0h**	**0%**
Applications deployed on web server	0	0%
Applications function on web server	0	0%
Project Idea	**0h**	**100%**
581 Group Determined	0	100%
Idea finalized with 581 Group	0	100%
Intended Languages / Frameworks c...	0	100%
Research API, Languages	0	100%
Mock UI	0	100%
Website Basics	**0h**	**100%**
Stand-up of Front End and Server	0	100%
API / Oauth Requirements	0	100%
Link front end with back end	0	100%

Preliminary Project Design [1046 words so far]



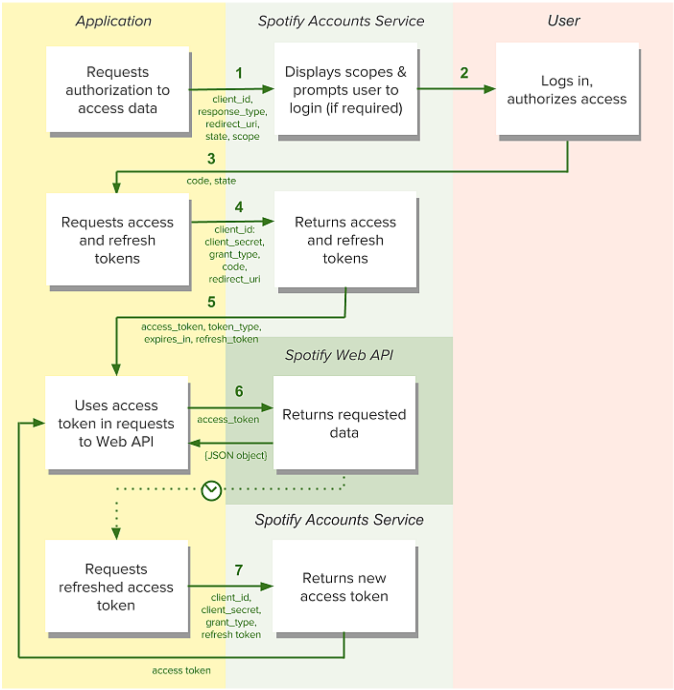
The server, running on a web hosting service, will be responsible for a large portion of the application's logic. If a user chooses to create a profile, it will pass their token data to be stored on the database. It will also be responsible for creating websocket connections for each user in a room, as well as storing the room's state. Updates made to the playlist will need to be passed through the websocket connection to the server, and reported to any of the websockets' subscribers to keep a constant state across all connections. Any of these actions could be logged on the database as well. Information about requests involving the Spotify API will be passed to the server to be logged in the database. It will be implemented using C#, and will be responsible for serving the front-end files to each user. The database will be hosted on AWS, and C# will need to be configured to support this.

To gain access to a user's Spotify information we will need to register our application with the Spotify API. Once registered we will have access to the Spotify API through our application.

For a user to use our application, they will need to login to Spotify through our web application. This will request a token from Spotify that is associated with the user's account.

With this token, we can now make requests to the Spotify API and receive data from those requests.

This token will expire one hour after receiving the token. This token can be refreshed by using the refresh token that comes with the authorization token. A more detailed outline of the process is shown in the chart below.


Application
Spotify Accounts Service
User
Requests authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Displays scopes & prompts user to login (if required)
2
Logs in, authorizes access
3
code, state
Requests access and refresh tokens
4
client_id: client_secret, grant_type, code, redirect_uri
Returns access and refresh tokens
5
access_token, token_type, expires_in, refresh_token
Spotify Web API
Uses access token in requests to Web API
6
access_token
{JSON object}
Returns requested data
Spotify Accounts Service
Requests refreshed access token
7
client_id, client_secret, grant_type, refresh token
Returns new access token
access token

The Spotify API requests will be done on the client side in JavaScript.

To gain access to a user's Apple music information we will need to construct a developer token as a JSON object. This token registers our application with Apple music, so we can have access to the API.

For a user to use our application, they will need to login to Apple Music through our web application. Logging in will request a music token from Apple Music to allow our web application to have access to their account information.

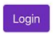
With this token our application will be able to make requests to their account information via JSON response objects. The requests and JSON handling will be done on the client side in the browser in JavaScript.

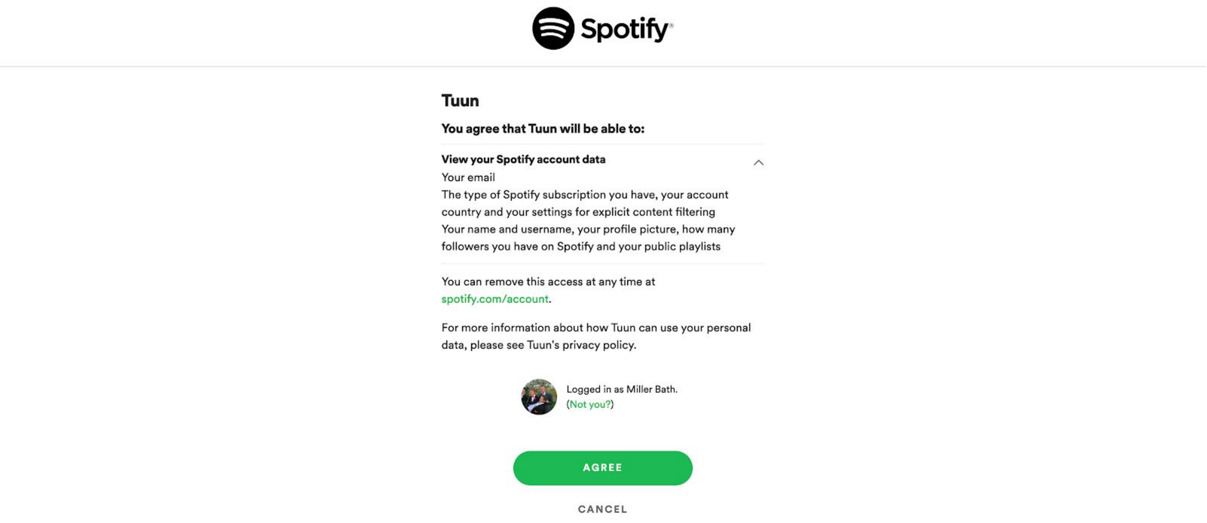
The web player will be based on *Spotify's* web player setup. Simple and efficient. A web player will pop up in the user view once a playlist has been generated. Each user pooled into a lobby will be given access to a web player with that specific playlist along with a link to *Spotify* to add the generated playlist to their public playlists. Our web player will be written in *React.js* through our *C#* application. Web requests to generate the web player will be handled by our application's server (as noted above).

The purpose of the web player is to simulate a similar experience found through *Spotify's* music player features. The web player will be very interactive. Our player will include multiple buttons, sliders, options and actions to provide a multi-faceted experience for our users. Common buttons may include: *Play Next, Pause/Play, Play Previous, Queue songs* and *Shuffle.* Each button is self explanatory in its purpose within a playlist player. The slider will be placed below the *Pause/Play* button allowing users to scrape across a song. Similar to the *Spotify* player, we will likely display the album cover on a secondary web player page. The list of songs will also be shown inside the webplayer to give a user the full scope of their listening experience.

The overall user interface will be simple and straightforward. There will also be two main sites to access Tuun. One version is Tuun.com. This is the image presented below that will have access to all the main features that we intend. The other website URL will be Tuun.in. From that URL, you will have a very simple "Enter Tuun Pin" text above a box, and below the box a submit button. It will take you to the collaboration page where you will incorporate the different users in the room's playlists.

On the main site, there will be several options for users to choose from. We will have a Spotify authenticated login that will show the users profile that is currently logged in. From there, you can view previous playlists that you created or collaborated with, or sign out. On the main page you will have a code for friends to join with, or a shareable link to send to friends. In addition to the features above, we will have a custom web player developed by us in order to play songs from both Spotify's API and Apple Music's API. This is required because the native web players in both APIs do not allow you to play music from both services if the playlist has songs from both Apple Music and





Tuun

You agree that Tuun will be able to:

View your Spotify account data

Your email

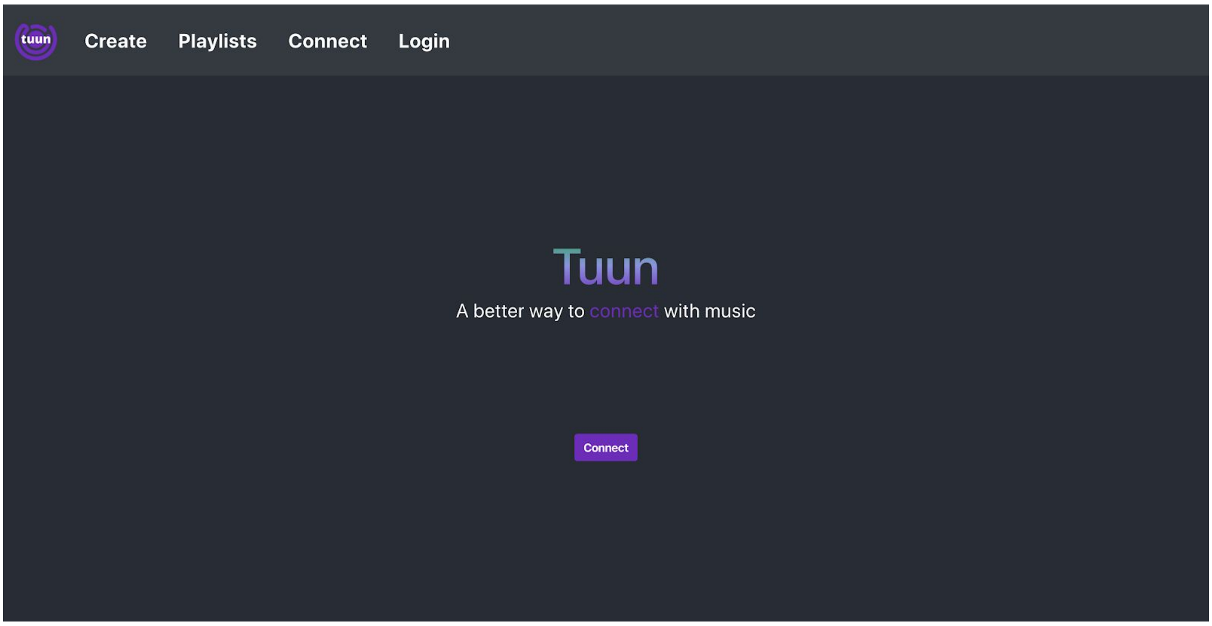
The type of Spotify subscription you have, your account country and your settings for explicit content filtering

Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

You can remove this access at any time at spotify.com/account.

For more information about how Tuun can use your personal data, please see Tuun's privacy policy.

Logged in as Miller Bath.
(Not you?)



We will be supporting the most up to date versions of Chrome and Firefox. No other browsers at this time will be up for consideration due to severe compatibility issues between modern code and outdated browsers(i.e. Internet Explorer).

The web player functionalities of both the Spotify API and Apple Music API only support the most recent versions of Chrome and Firefox. These functionalities do not support any mobile browsers, but the APIs still support general requests on all browsers.

C# in combination with React.js will be used for the entirety of the project. Database solutions like MongoDB or SQL might be used, however, we have chosen MongoDB. These languages are simple to use and can vary widely in their ability to complete the diverse requirements that we have for our project, Tuun.

We will host the web server on Amazon Web Services(also known as AWS). This will contain all of our code for React and C#.

Ethical Issues

The goal of *Tuun* is to provide a more synchronized musical-listening experience for multiple people within a certain proximity. In order to implement our service properly, we will require access to multiple users *Spotify* data including username and public playlists. Accessing this data is ethical sound because all accessible data is already public knowledge. Users agree to share their username and public playlist data by agreeing to the *Spotify terms of service* (URL below). *Spotify* users provide other users access to their own *public* data, but as an end user of the *Tuun* application, users will have to allow the application access to their *private* Spotify data. This may include personal information like user email, display name, country of origin, external URLs as well as listening history. Users personal data will not be shared outside of the application to other users and will only be held by the server during the initial pooling of users into a lobby. The only data that will persist will include user playlists and musical interest trends to best aggregate data from multiple sources. Due to this analysis, we believe the potential negative impacts of the project to be very minimal. Our conclusion is that our application's methods to achieve its goals are ethical. https://www.spotify.com/us/legal/end-user-agreement/

Intellectual Property Issues

Tuun is a web service based on the media streaming giant, *Spotify.* Our product utilizes *Spotify*'s open-source API in order to access user data. While many other open-source projects have utilized *Spotify's* developer tools, we are not aware and have not found another web service that accomplishes what we plan to. While *Spotify* may be considered "developer-friendly", we cannot access the *personalized playlist generation* algorithms *Spotify* makes to craft well-optimized music listening experiences for each of its users. Our algorithms will set out to achieve a similar task for multiple users. Most open-source projects leaning on *Spotify's* API are not officially licensed and therefore are fair game for code reuse. However, our product will surround a uniquely created algorithm purposed to aggregate user data and generate playlists for them. With that, any third-party tools used within our application will require verification and reason for their use.

Change Log

Updated completion dates and added documentation deadlines under project milestones.

We changed the idea of having a static playlist generated and displayed on the web page to a dynamic playlist in our own web player. This will allow us to have more freedom and be less reliable on the APIs because it would be a music player made by us. It will also allow for users to change a playlist while listening to it, which was not possible before.

We also changed to having two distinct URLs. One for PIN login information and the other to log into your account where you can manage and create shared playlists.