Team Number: Section 011 team 4

Team Members:

Bernardo Fortunato

Bo Zhang

Dalbir Brar

Jared Jewell

Jacob Levato

Team Name: Mr. Worldwide

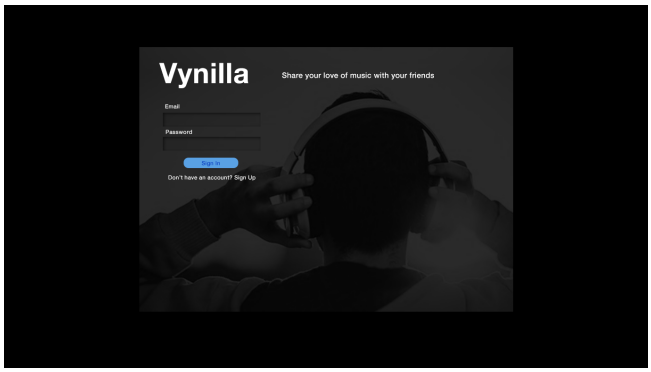
Revised Feature List:

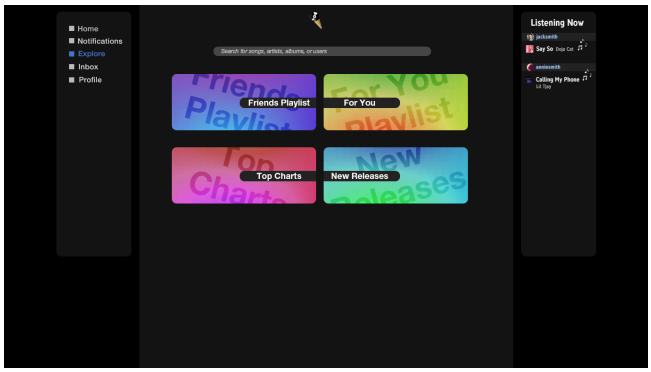
1. **Accessing the music on Spotify API**
    - (Specified below, under the web-services subheading)
2. **User Profile - Displaying a user's favorite music on their profile**
    - A user can showcase their favorite artists, albums, and songs for their friends to see.
    - To themselves, their own listening habits are visible. A user can make those statistics public
3. **Connecting with friends**
    - Users will be able to search for other users & 'friend' them, so that their activity will appear on each others' feeds.
4. **Sending music suggestions to Friends — Queue -> Playlist**
    - Users can make playlists and send them to other users.

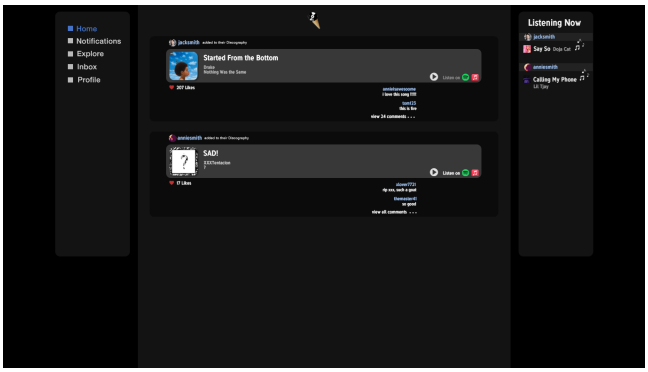
5. **“Social media” feed**
    - Users will see the actions and updates of their friends in a timely manner
    - ~~Stories-type *banner* on top of the SM *feed*~~
        - ~~Feed: Any update a user has to top 5: Artists, Songs, or Albums~~

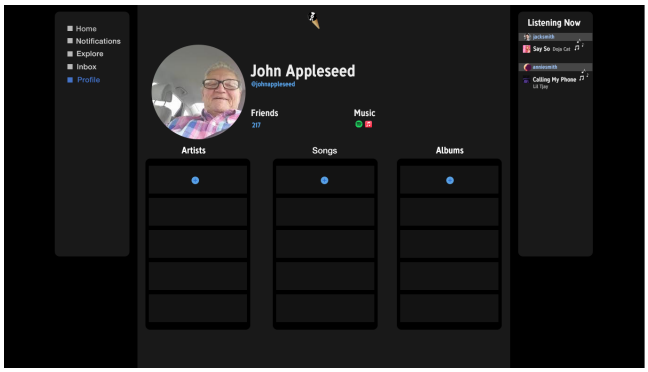
~~———~~Changes : Switched order of accessing Spotify and sending music suggestions to friends. Switched order of sending music suggestions and connecting with friends. Will not implement stories feed at the top of the page anymore.

Front End Design:

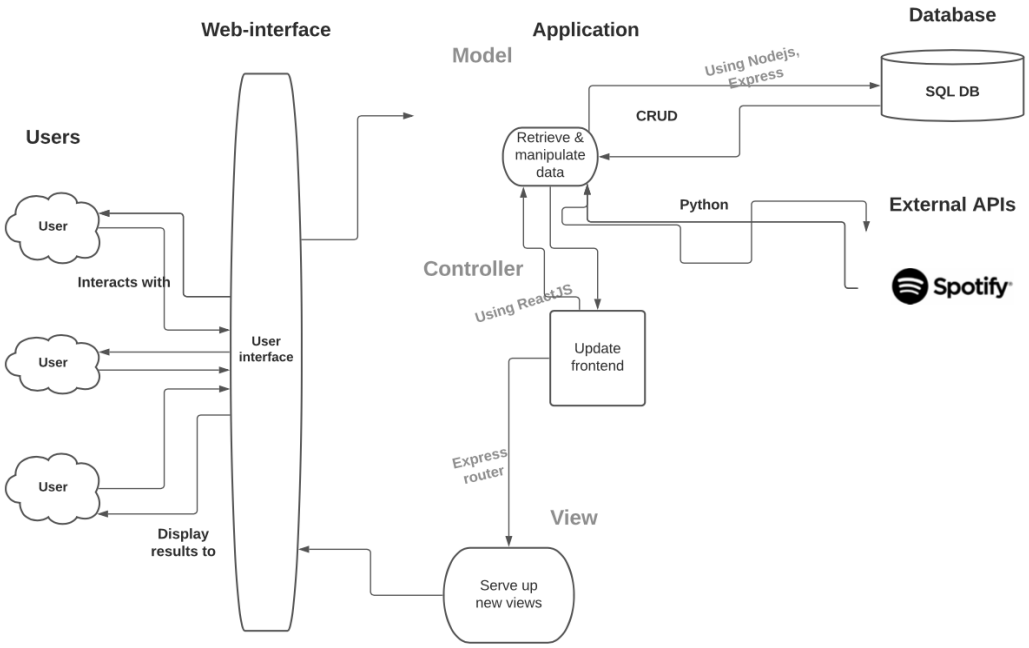








Architecture Diagram:



Note: we will be using a Javascript library (spotify-web-api-node) to communicate with Spotify, rather than Python.

Web Service Design:

We will use the Spotify API, and a Spotify Node wrapper, to allow Node to communicate with the Spotify server.

Description: We will make various API calls. For example….

Obtain a user's permission—dependent on which scopes of data we intend to access

- Get the song that a user is currently listening to (store this, for each user)
- Create a playlist on the user's account, populate it with songs (queue functionality)
- Browse their favorite genres & their related songs (populate explore page)

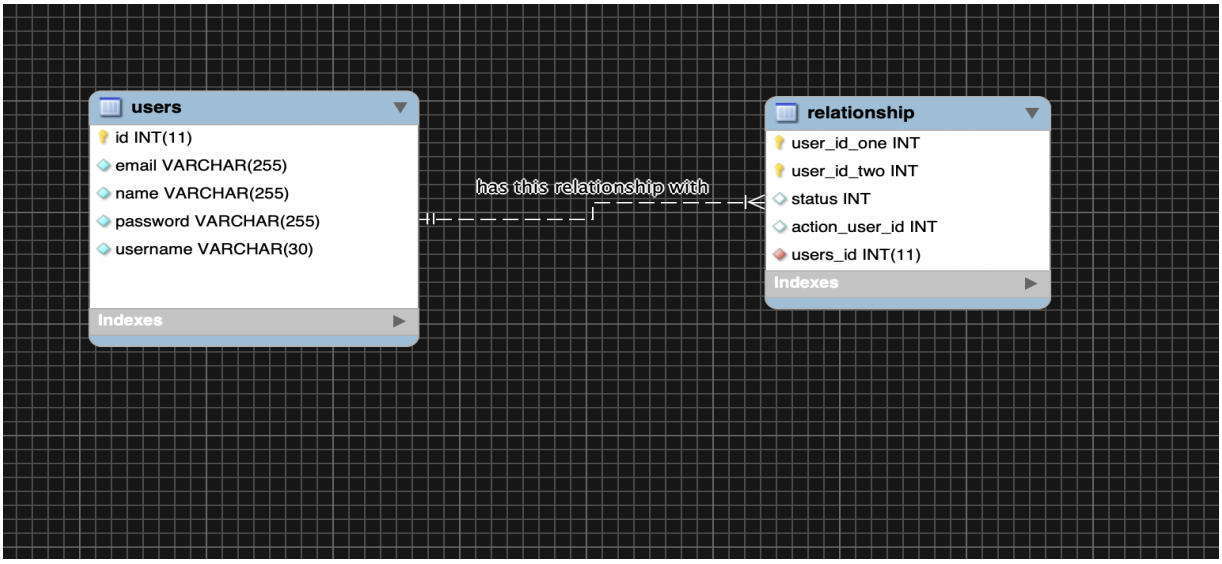
Data being passed: Details from the user's Spotify account such as their username, password, and music data, as well as those specified above

Database:

Name	@name	Spotify Pfp link	Top-artists Array	Top-songs Array	Top-artists	Last-song	Queue

Database design:

1. A Friends table (many to many) would allow users to connect with friends and view their music preferences.
2. A Queue element in the database will allow song and album recommendations to be sent to friends.
3. A main feed table in the database will store the Every single feed element posted, and will make the posts visible if a friend connection exists.



Individual Contributions:

1. Jake Levato - Connected all the webpages together with javascript and made a universal navigation bar. Cleaned the website up so it looks like the same theme throughout all pages. Started working on friends relationship status (not finished).
2. Bernardo F. - Created the sensational Milestone document. I worked on the active Spotify API calls to display concurrent user listening habits.
3. Dalbir Brar - Visualized Database. Added handlebar code to profile page (sending back HTML) wrote a simple get/post script to send get user's Queue from database and send back as cards onto profile page in a flexbox… truly visionary work.
4. Jared Jewell - Created demo Spotify API call, to give an example of the workflow that we'll be using to make API calls and Authorization. Currently, we are able to get a user's permission, for a variety of authorization scopes (get currently playing song, get favorite artists)
5. Bo Zhang - Worked on the explore page.

Challenges:

1. Creating a "this user is currently playing" section.
    a. Updated: get it to render in HTML. Currently, it is displaying in the console
2. Creating organized database features for storing every user connection
3. Keeping on track with our JIRA roadmap vs. just working on things randomly
4. Storing information from Spotify? Or do we just request information when we need it

    Back-up Plan: Always request on demand

    Pro: We KNOW we can do this

    Con: Could bloat performance