TNPG: Ctrl Alt Defeat
PM Emily Ortiz (Rain), Ryan Lee (Luigi), Brian Wang (Snowy), Yuki Feng (Ben)
P4 Design Doc
SoftDev Pd 8

Target 🚢 Date: Tuesday, June 6, 2023

Website Description
Site name: Crocodile Covers

- Based on a childhood cartoon character and recent film adaptation, Lyle Lyle Crocodile, a singing crocodile, Crocodile Covers utilizes Lyle as its mascot for its website.

Users Can

- Search for a song by inputting the song title or the respective artist into a search bar
    - Up to 10 results are shown
    - Extra: Button for showing more results
- Add songs to their playlist
- Play songs and show the lyrics of songs
- Have word clouds to display the most common words of songs in that playlist

FEF
Bootstrap

- Display word cloud above playlist
- Buttons to play songs
- Accordion for song lyrics

HTML Files

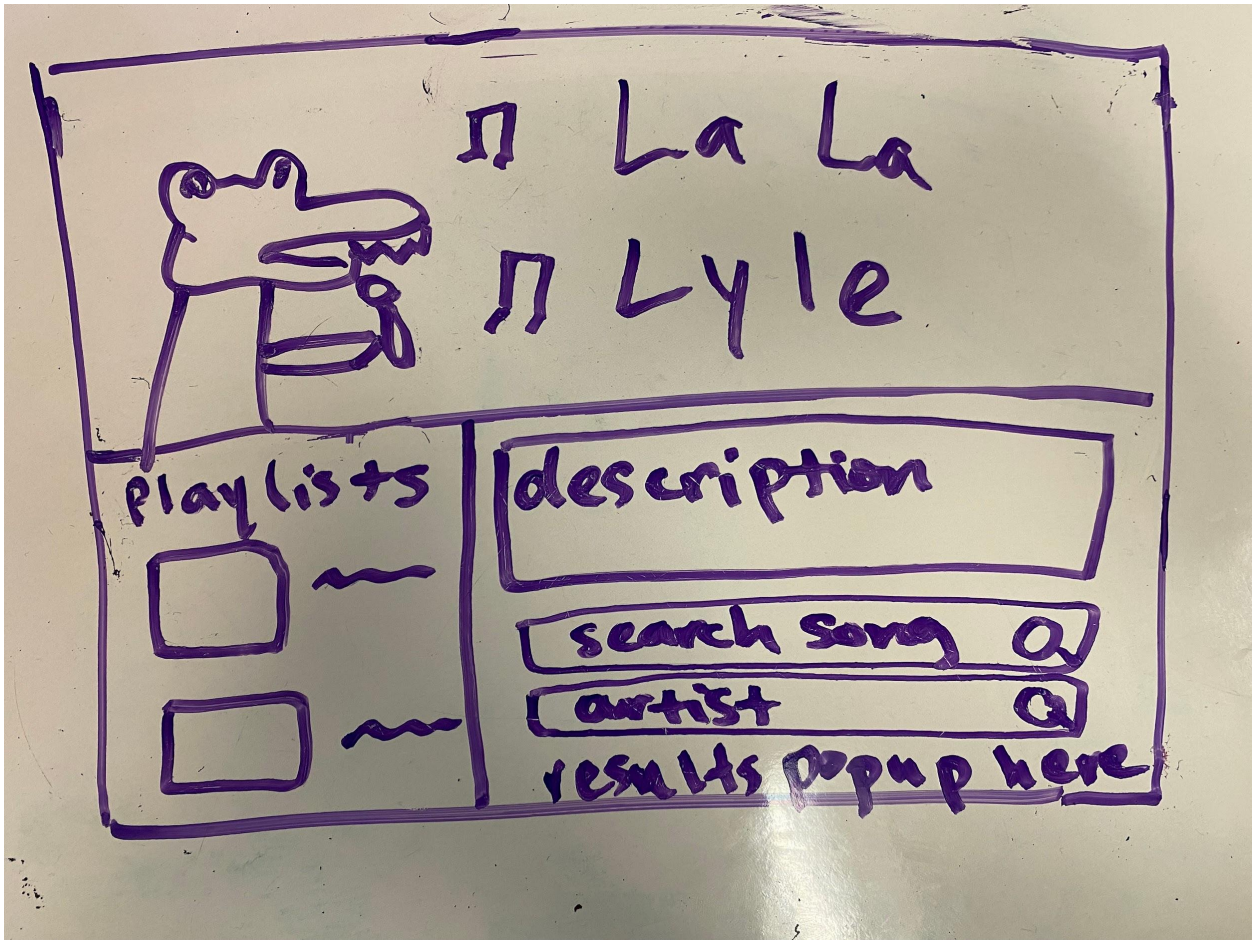
- login.html
- register.html
- index.html

Statuses returned to register page from __init__.py:

- Username namespace conflict (register)
- Password confirmation error (register)
- Incorrect login (login)

Fields needed:

- username (register, login)
- password (register, login)
- password-confirm (register)
- entry (index) (search query)



APIs

- MusixMatch: https://developer.musixmatch.com/
    - Provides song lyrics sorted by a multitude of tags (artist, genre, etc…)
    - Database too big to store locally; will have to do repeated calls for lyrics
    - does not use OAuth
- Word Cloud: https://wordcloudapi.com/
    - Create word clouds off of inputted words
    - Can change appearance, size, parameters of word cloud
    - Can output word clouds directly to HTML

Methods bread needs from API:

- (maybe) TAKES a song name snippet, RETURNS a list of songs from partial word search
- TAKES a song name, RETURNS the entry to that song
- TAKES a song entry, RETURNS the lyrics (in csv or string) to that song
- TAKES a song entry, RETURNS the artist
- (maybe) TAKES an artist, RETURNS the songs created by the artist
- TAKES a dictionary of words and occurrences, RETURNS the word cloud image

DB

Sqlite3

Login Info	
Username	Password

User Playlist			
Id of song (for easy access)	Song name	Artist name	Lyrics

Global Playlist (Extra)				
Id of song (for easy access)	Username for the person who added the song	Song name	Artist name	Lyrics

Methods bread needs from DB:

Login

- TAKES a username and password, and creates a user entry
- TAKES a username and password, RETURNS whether or not that information is accurate (verify login)
- TAKES a username, RETURNS whether that username is currently being used

User Playlist

- TAKES a song name, artist, and lyrics, and assigns an ID and creates an entry
- TAKES a song name, RETURNS its corresponding ID
- TAKES an ID, RETURNS its corresponding ID
- TAKES a song ID, removes the entry from the table
- TAKES a song name, removes the entry from the table

Store Musics in DB vs Pulls:

DB: The music database is massive. Unless we specifically pick the top of such and such, it's unlikely we'll be able to actually fit all the information we need locally. This poses an interesting compromise, it would be possible to merely store the information for the first few, say, many thousand, but doing so would result in a much more restrictive selection for more niche listeners.

Pulls: The only real downside to this would be the need for constant pulls. The only major problem we foresee with this would be the pull quota for MusixMatch, of which it might be possible to default to a second API, although I doubt this would be necessary unless our site actually takes off for some reason.

Store playlists in Global vs Cookies vs Local vs Login:

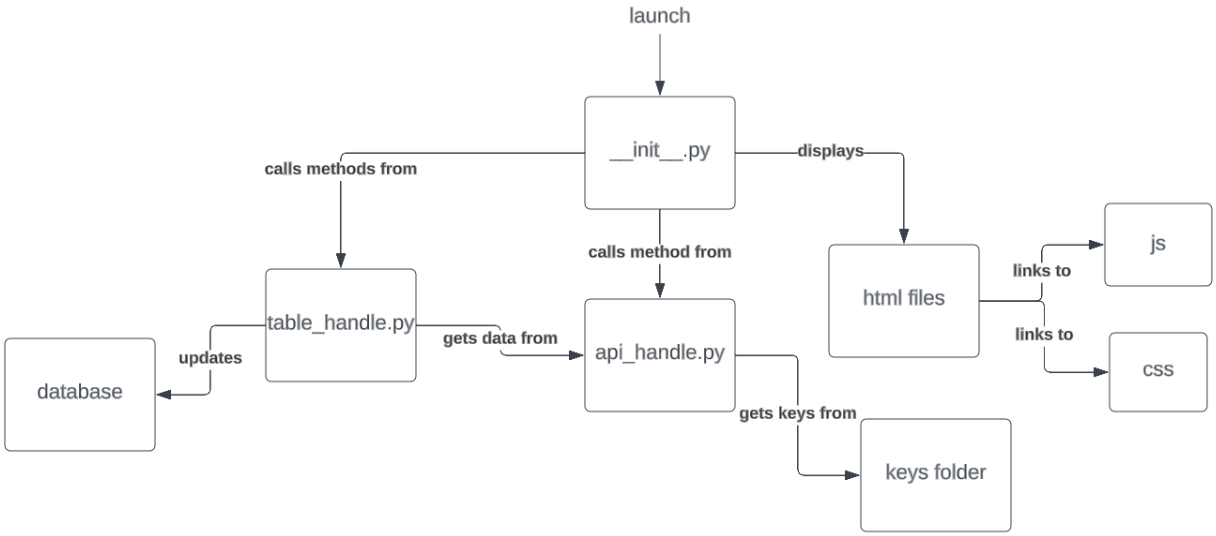
Login: Just a hassle for the user, that's about it. Would mean playlists are stored per login basis, so everyone could just have their own personal playlists on their own personal account.

Local: Makes some sense, since only one listener would be using one device. Problem is the fact that we are hosting this on a DO platform for everyone would mean that the playlists would be shared among everyone accessing our website via the DO served link. Would work fine on a small scale, though.

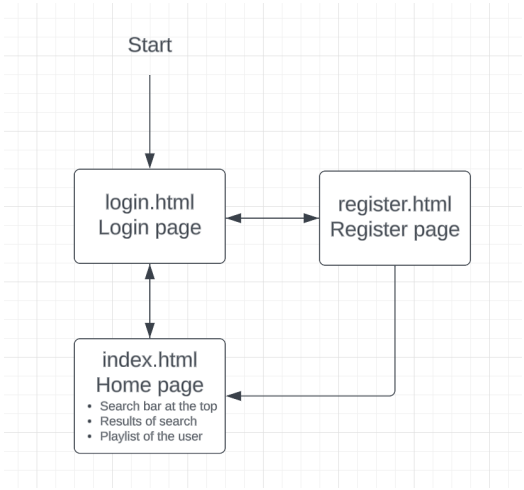
Cookies: Cookies?

Global: This one could be fun, what if we just had one shared database for everything? Would sort of be like local but instead of half-assing it we make the half-assedry official. Everyone's playlists are public, and anyone can pick up where someone else left off.

Components Map



Site Map



Search Page

- Search bar in nav bar for songs to look for
- Upon retrieval of the correct audio and lyrics, JS shows the lyrics on the page for the user
    - Have lyrics pop up as song progresses
- Have a pause and play button for music

Playlist Page

- Word cloud displaying most common words from lyrics in that playlist
- Word cloud can have filters for song title/artist/genre.
- Word cloud can also be hidden on click. Maybe like a dropdown?
- List of songs in playlist
    - button for each to play/pause
    - Accordion/dropdown for each song's lyrics

Task Breakdown

- Yuki - Visuals/Bootstrap, JS
- Brian - Flask
- Ryan - DB
- Emily - APIs

To-Do

- ☐ Make API cards - Emily
- ☐ Update readme launch codes and data section

Stretch Goals

- Light up words
- Multiple Playlists per user
    - Soundcloud for songs across playlists
- Global Playlist

Super Stretch (Hardware Intensive!!)

- Record audio, reference with audio tones should be, determine how well someone sung