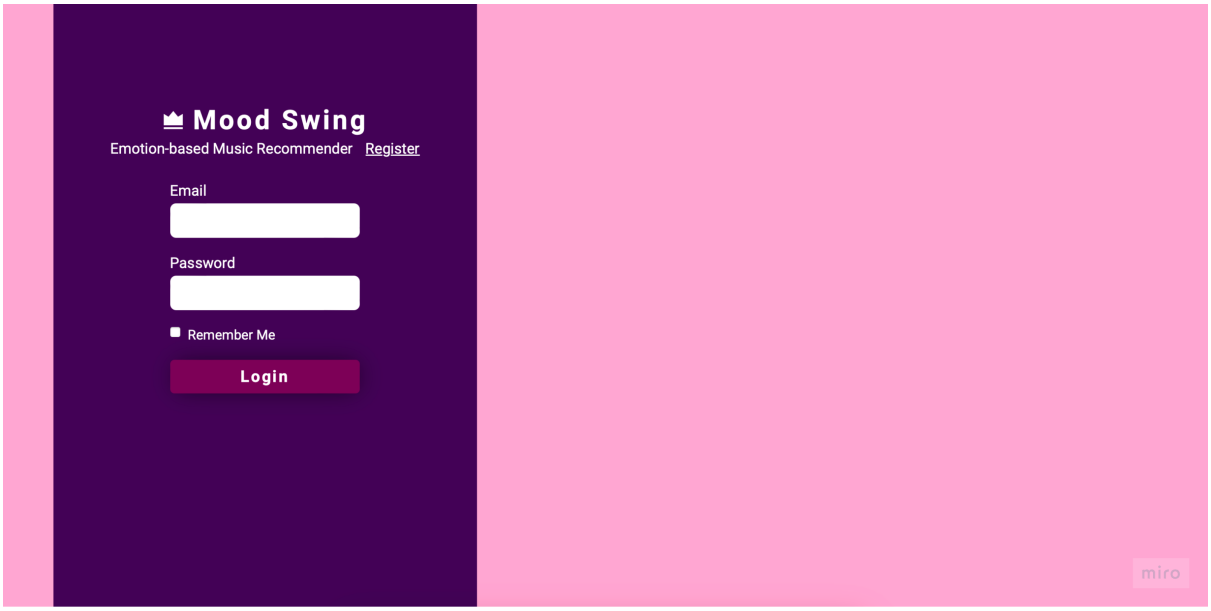
MOOD SWING
Web Application for Sentiment Based Music

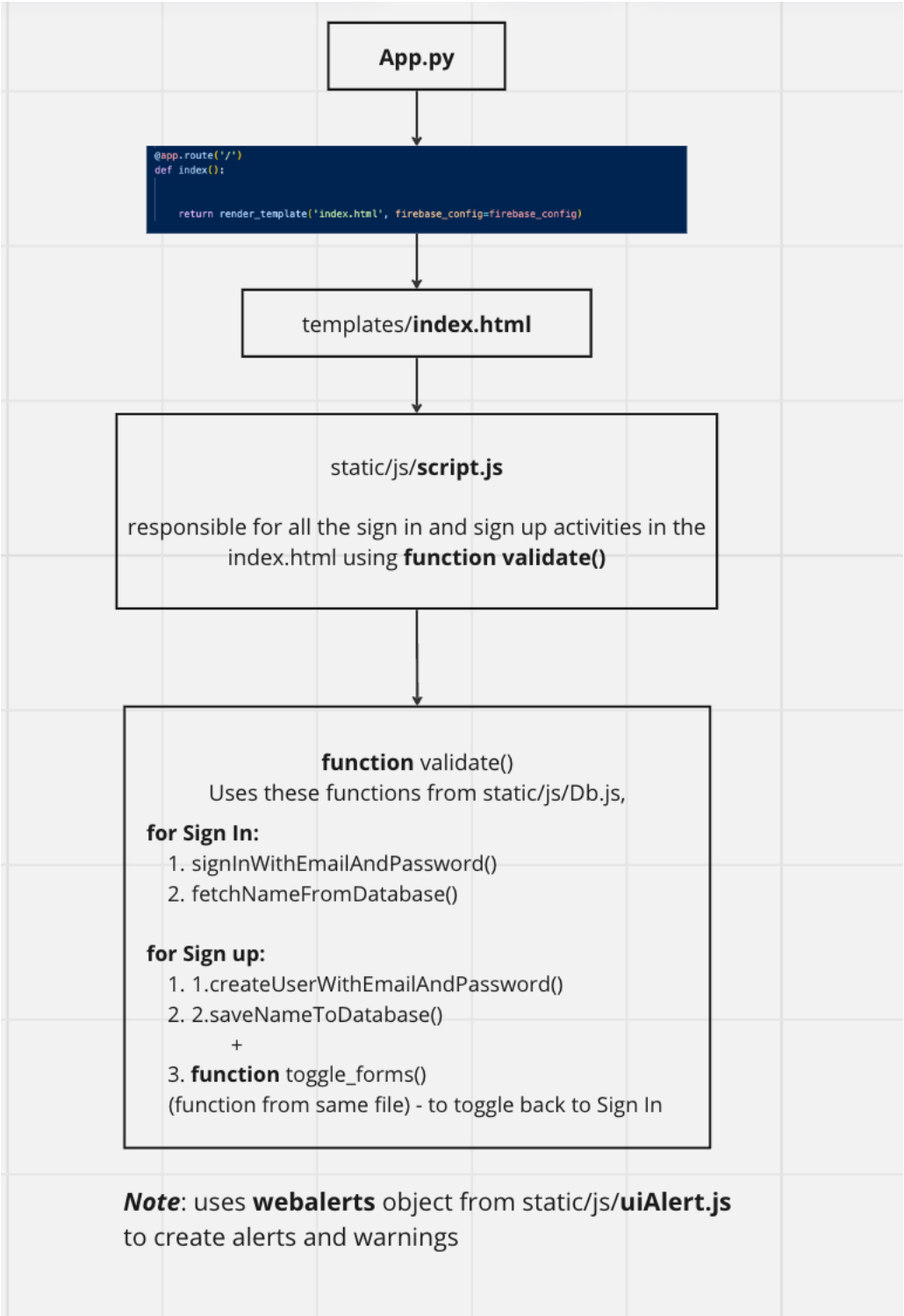
Tech stack:

1. **Python Flask**
2. **Firebase Real time database, Storage, and authentication**
3. **JavaScript, HTML, CSS**

Index Page & Authentication:

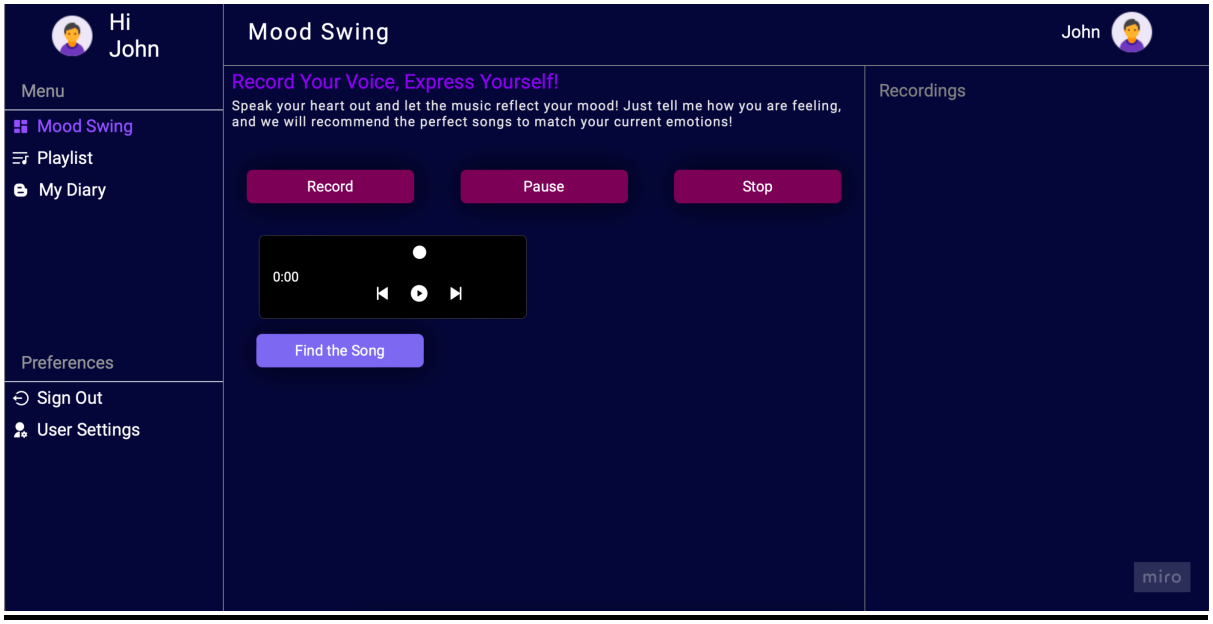


1. Start the App by running **App.py.**
2. Uses the template from **templates/index.html**
3. In that html file will use **static/js/script.js** as a script file.
4. In that Sricpt file there is a function called **validate() –** that will take care of all the **Sign In & Sign Up** activities
5. That validate function will use some functions from **static/js/Db.js,** to connect with firebase authentication.(refer the block diagram below)

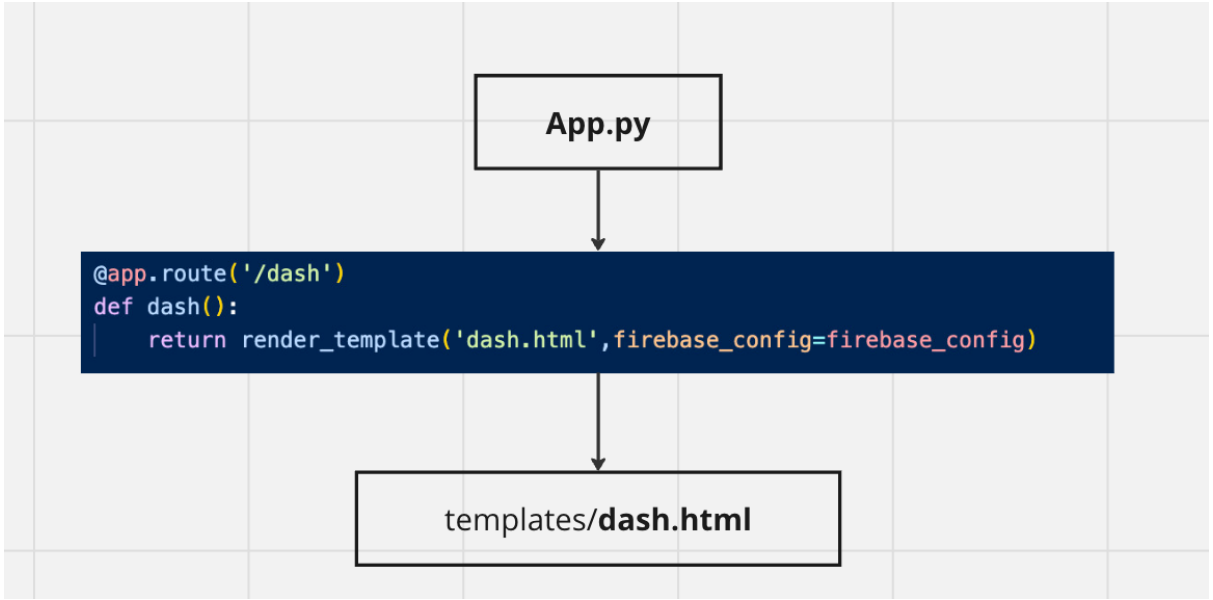


---------- END OF **Index Page & Authentication**----------

Dashboard Page:

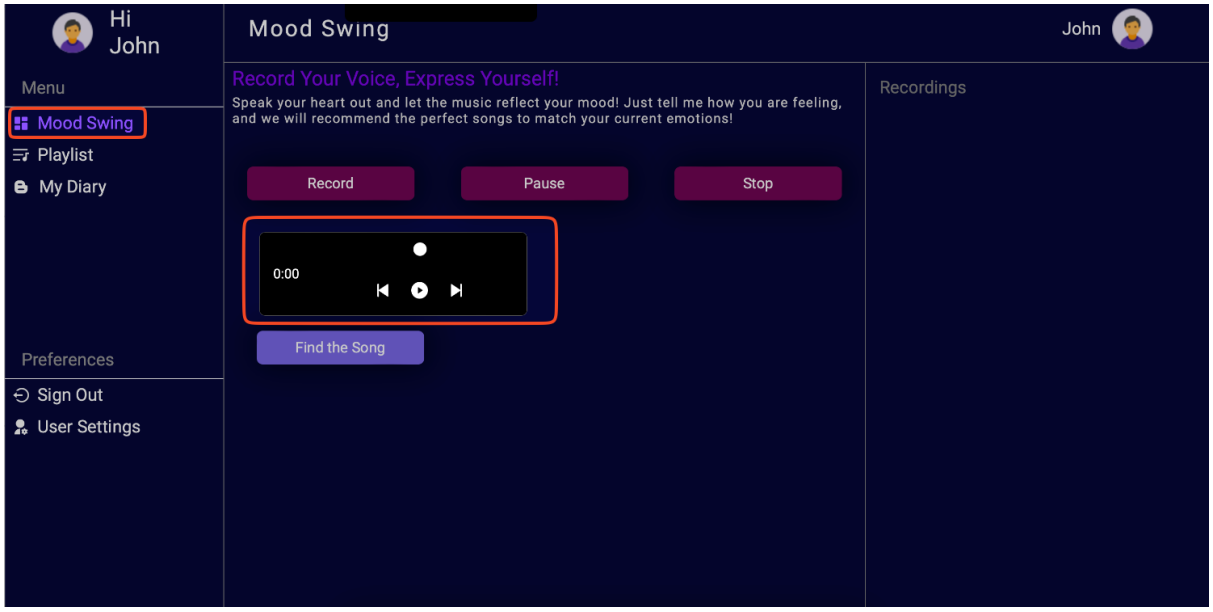


1. After successful sign in **app.py** will render the **dash.html** template from **templates/dash.html.**



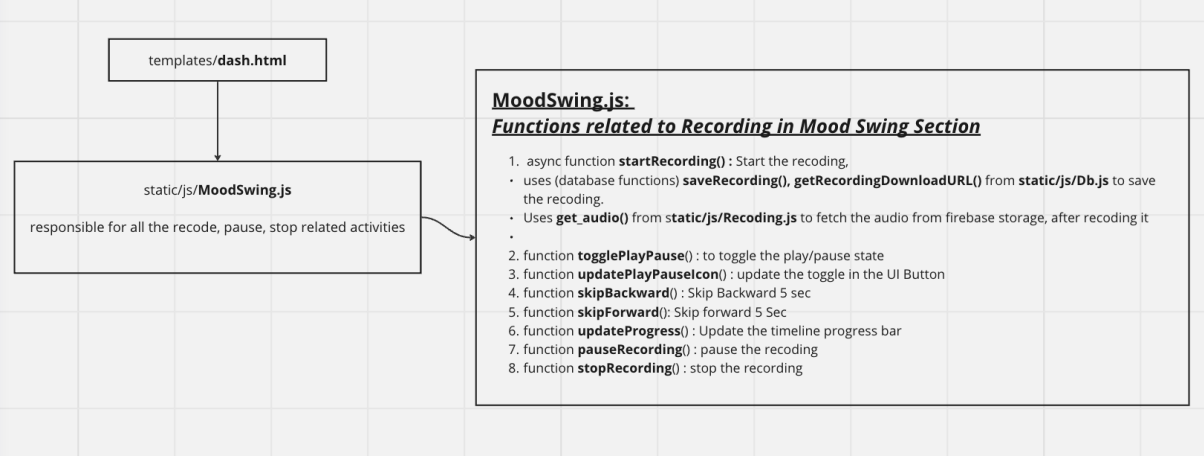
---------- END OF **Dashboard Page** ----------

1. Dashboard – Mood Swing

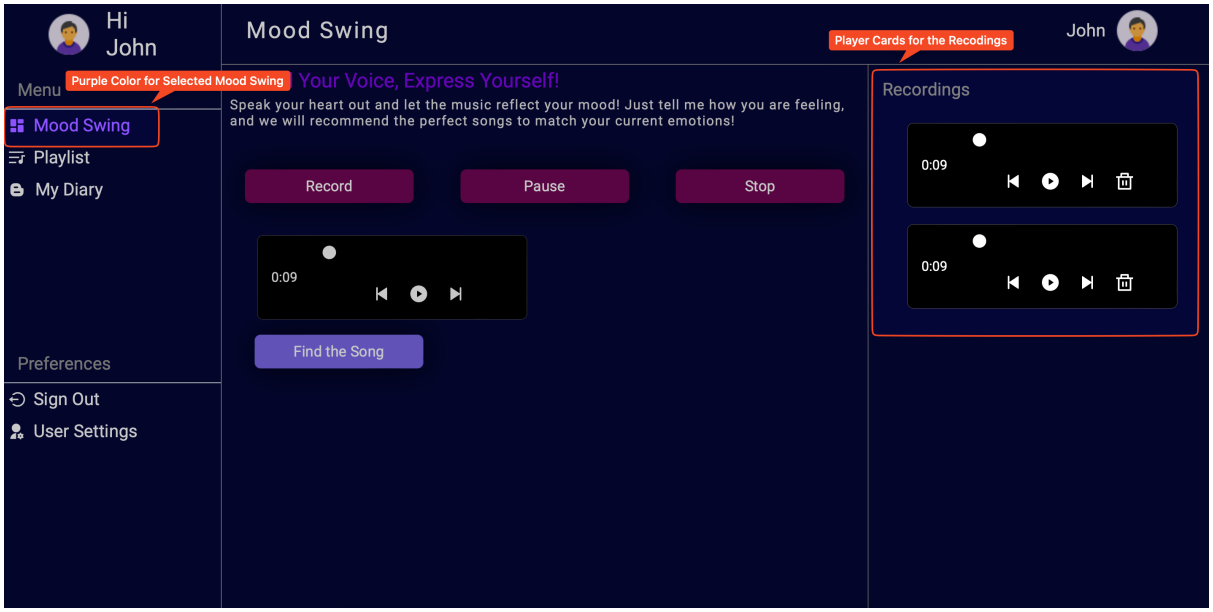


1. dash.html file will use **static/js/MoodSwing.js** as a script file.
2. There are several functions in MoodSwing.js which is responsible for Recording related activities.(refer the block diagram below)

NOTE: Some of those function will use functions from **static/js/Db.js and static/js/Recording.js**



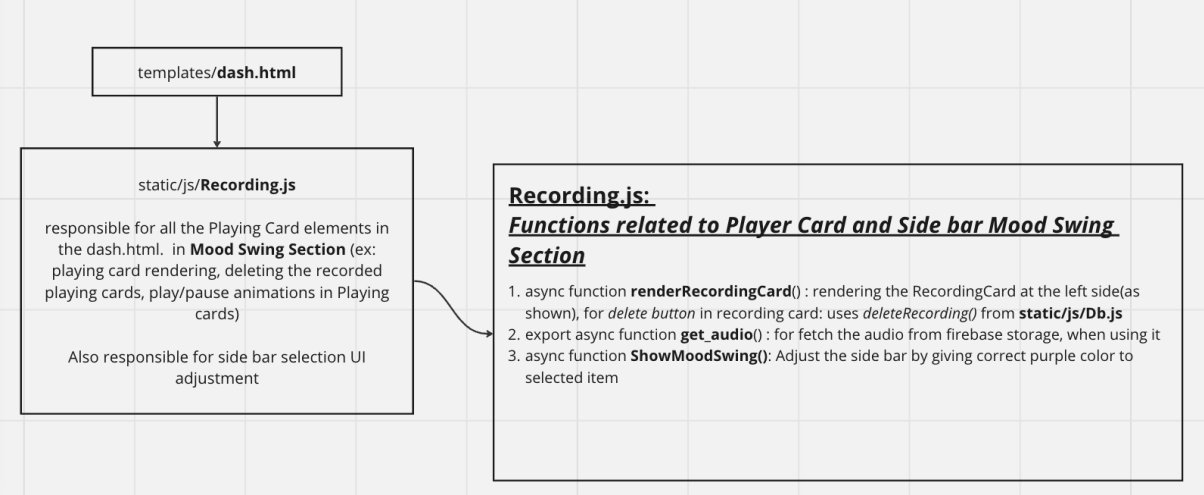
3. After finish recoding it will store in the firebase cloud storage by using saveRecording() function from **static/js/Db.js.**
4. Also, recodings will shown in the dashboard as **Playing Cards** as Shown below.

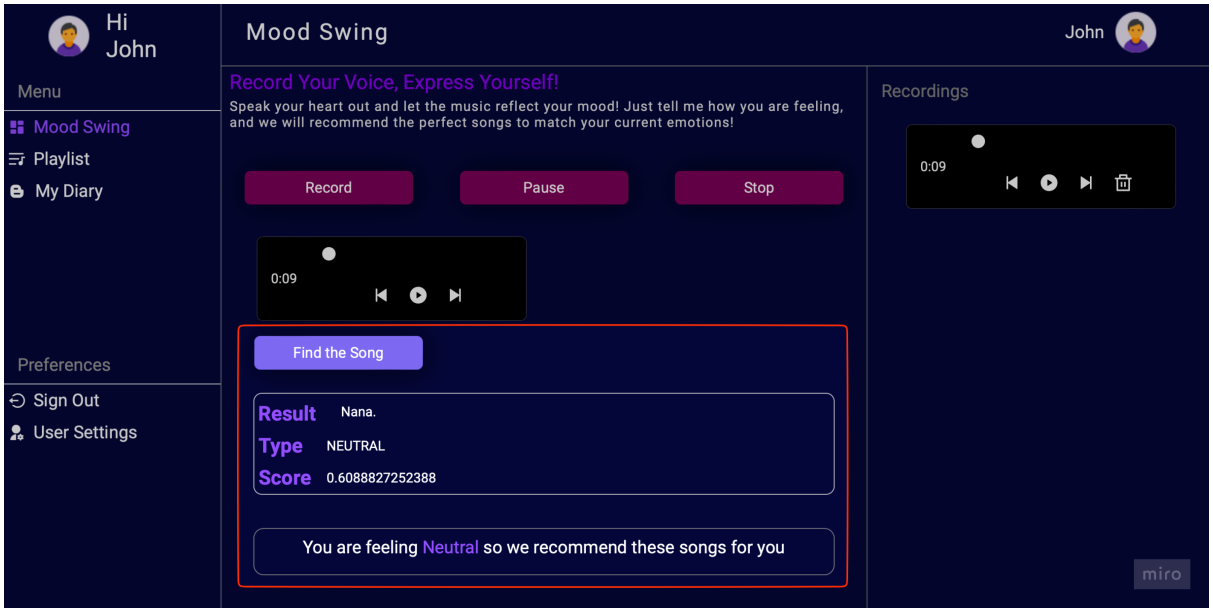


5. dash.html file will use **static/js/Recording.js** as an another script file.
6. There are several functions in **Recording.js** which is responsible for **Player Card** related activities.(refer the block diagram below)

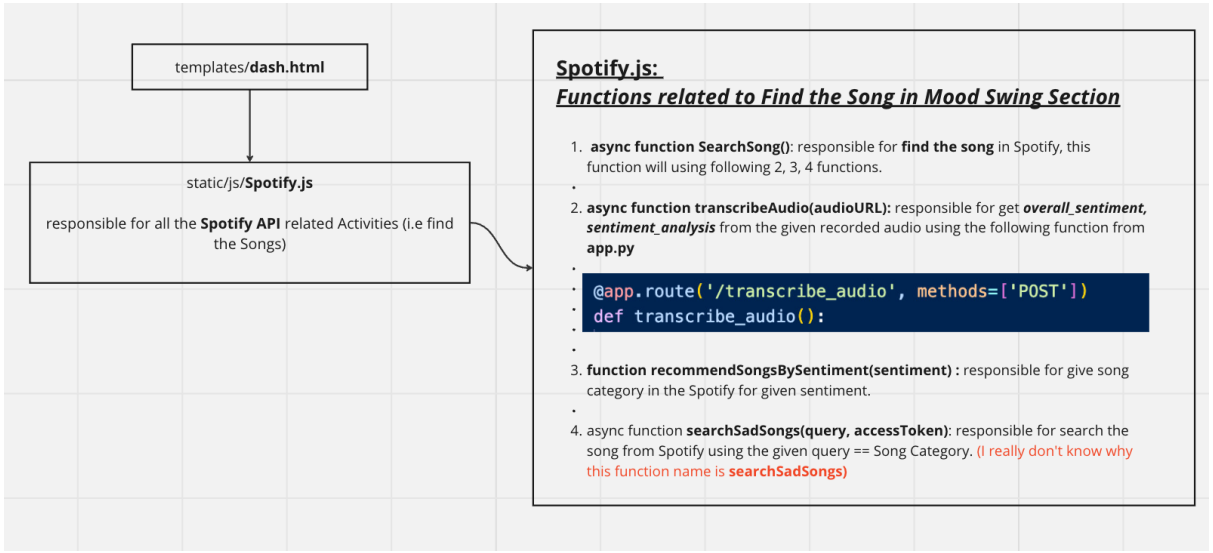
NOTE: Some of those function will use functions from **static/js/Db.js.**

7. Also there is a function in **Recording.js** which is responsible for **adjust the side bar** selection by giving a separate color as shown in the above screen shot.





8. After recoding the audio, Using **Find the Song button** we can search the songs which is suitable for our mood in the recoded audio.
9. For that, dash.html file will use **static/js/Spotify.js** as an another script file.
10. There are several functions in **Spotify.js** which is responsible for **Spotify API** related activities which is Find the Correct song for our current mood.(refer the block diagram below)



In below NOTE you can get an idea about how sentiment analysis function (***transcribe_audio ()***) really works

NOTE: ***transcribe_audio ()* FUNCTION IN *app.py***

EXPLANATION FOR THE *transcribe_audio()* FUNCTION IN *app.py*

1. The `**transcribe_audio()**` function is a Flask route designed to transcribe an audio file specified by its URL (`audio_url`) and perform sentiment analysis on the transcription results using the python's **`aai` module**.

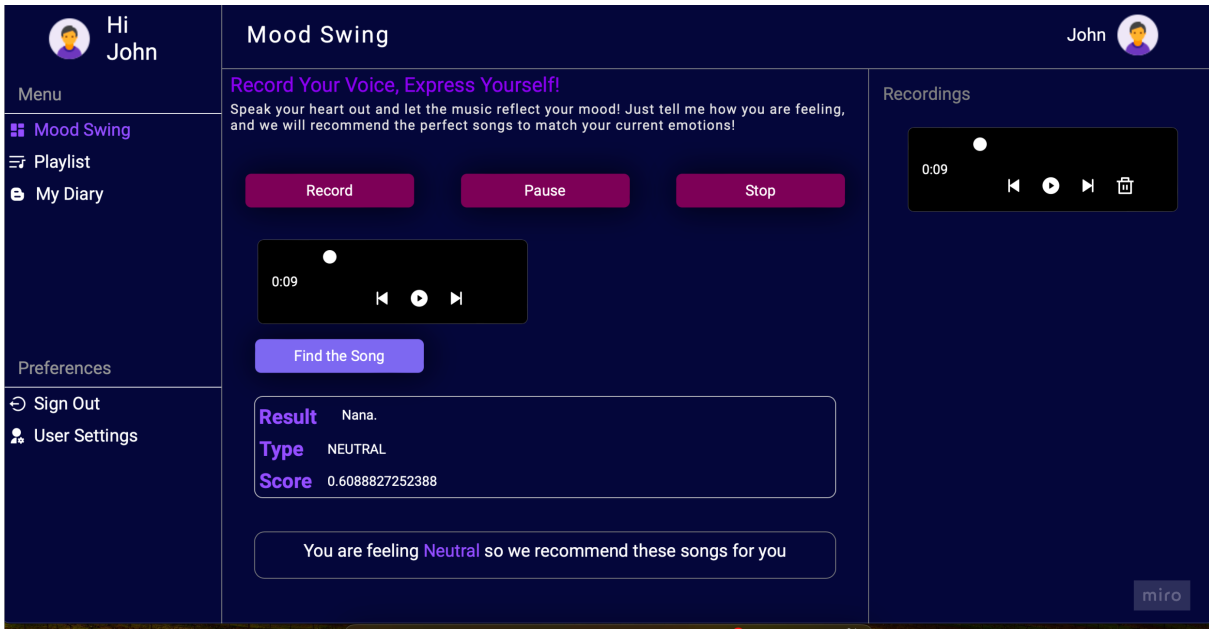
2. This module likely encapsulates functionality for **audio transcription and sentiment analysis**. The input to the `aai` module includes the **audio URL (`audio_url`)** and a transcription configuration (`TranscriptionConfig`) with sentiment analysis enabled.

3. The `Transcriber` class is then used to transcribe the audio file based on the provided configuration. **The output from this process includes a transcription (`transcript`) along with sentiment analysis results for various segments of the transcribed text.**

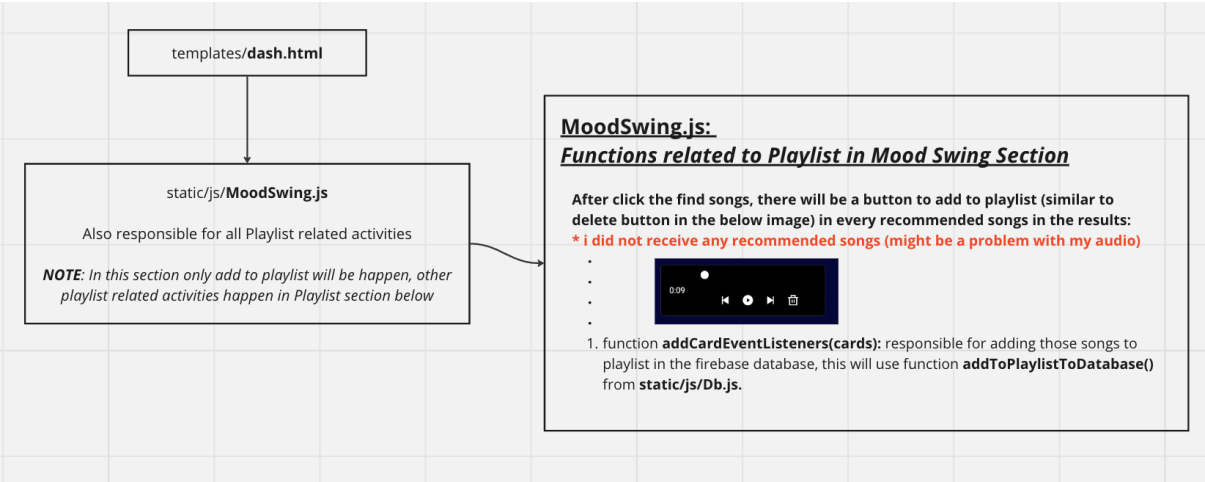
4. The function aggregates these results to **calculate an overall sentiment (`overall_sentiment`) based on the most frequent sentiment label.**

5. The final output is formatted into a **JSON response (`result`) containing the overall sentiment and sentiment analysis details for each segment of text analyzed,** which is then returned to the client.

6. The function also includes error handling to catch and report any exceptions that may occur during the transcription and sentiment analysis process.

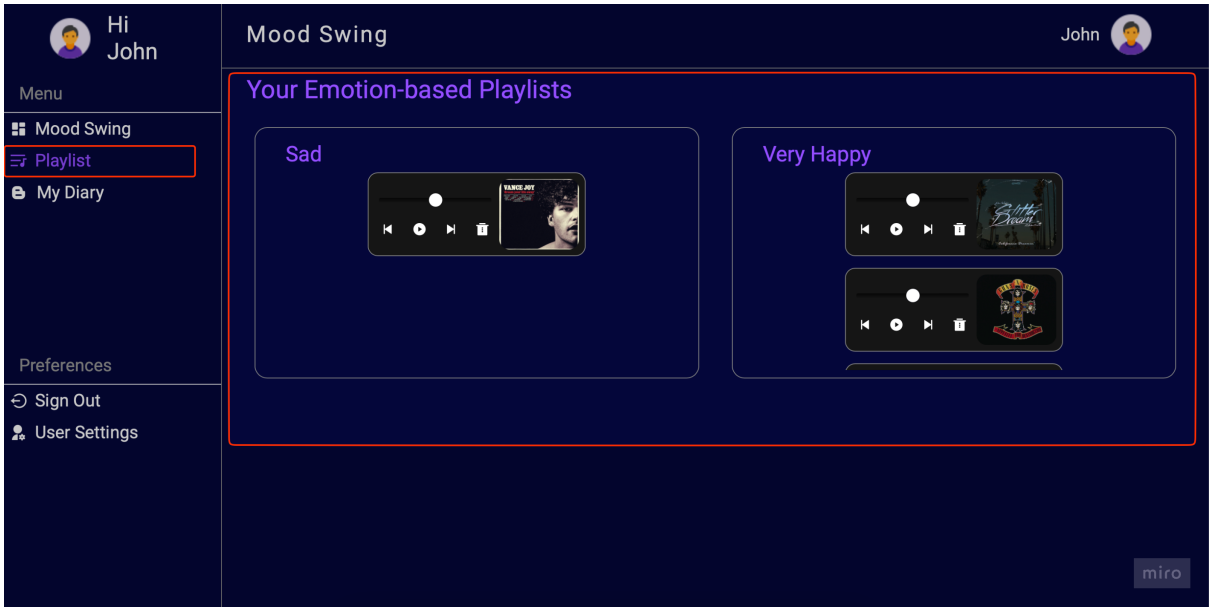


11. After find the recommended songs from Spotify API, we can add them to our own playlist in the firebase database.
12. As Stated, dash.html file will use **static/js/MoodSwing.js** as a script file.
13. There is a function called **addCardEventListners(Cards)** in the **MoodSwing.js** which is responsible for add songs to playlist in the firebase database. .(refer the block diagram below)



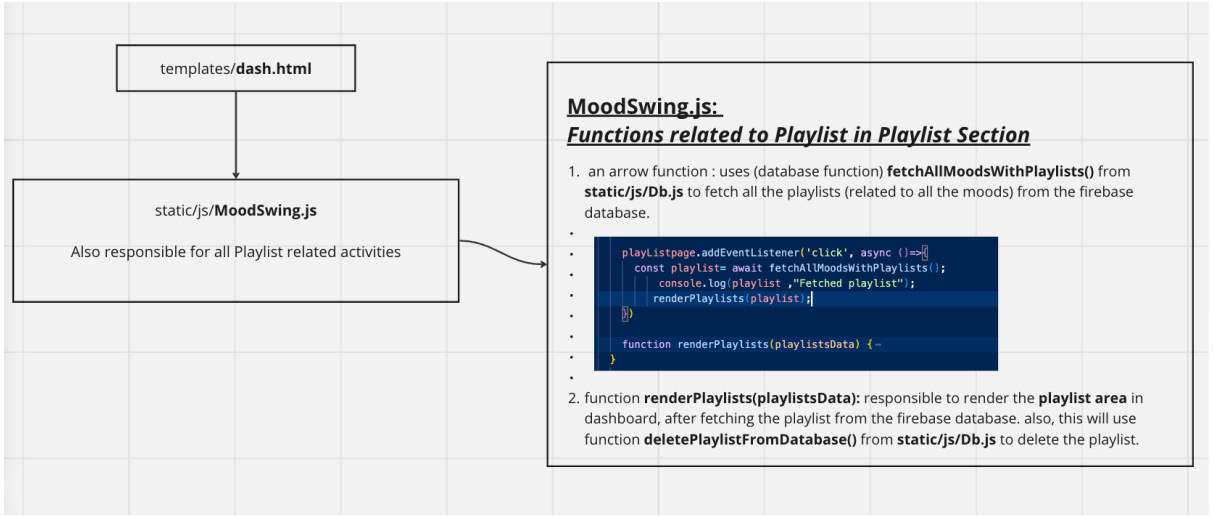
---------- END OF **Dashboard – Mood Swing** ----------

2. Dashboard – Playlist



1. As Stated, dash.html file will use **static/js/MoodSwing.js** as a script file.
2. This file is also have functions which is responsible for activities related to playlist section in the dashboard (i.e delete playlist, render playlist).(refer the block diagram below)

NOTE: add playlist also done in the **MoodSwing.js** (explained in the last part in **dashboard - mood swing** above)

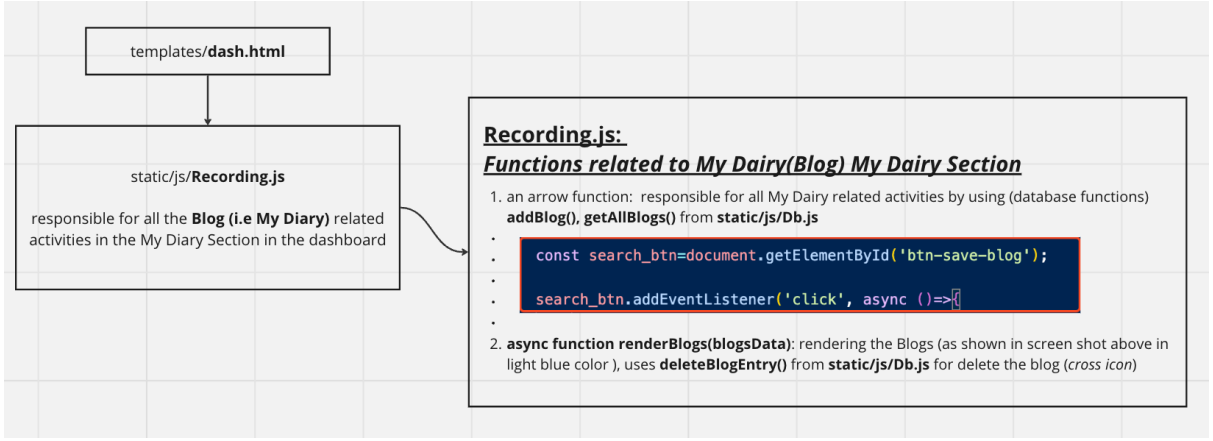


---------- END OF **Dashboard – Playlist** ----------

3. Dashboard – My Diary

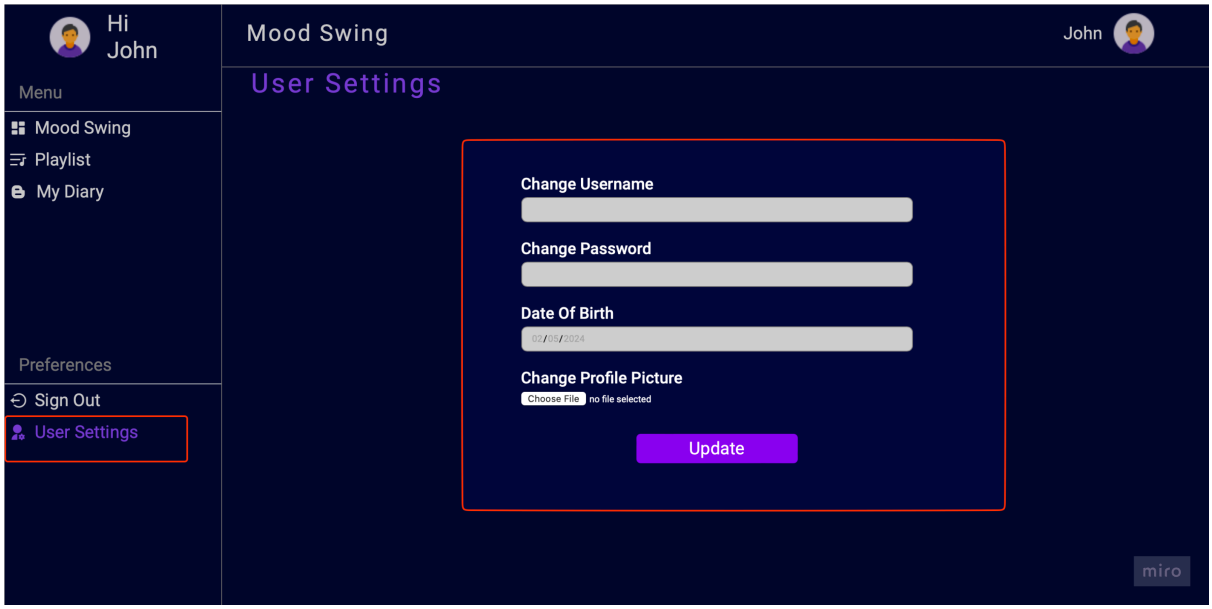
1. As Stated, dash.html file will use **static/js/Recording.js** as a script file.
2. This file is also have functions which is responsible for activities related to My Diary section in the dashboard (i.e add blog to firebase database, delete blog from firebase database, get all the blogs from the firebase database, render all blogs in the webpage).(refer the block diagram below)

NOTE: **adding blogs, get all blogs** will happen inside **arrow function**, **delete and render** will happen inside the **renderBlog()** function.



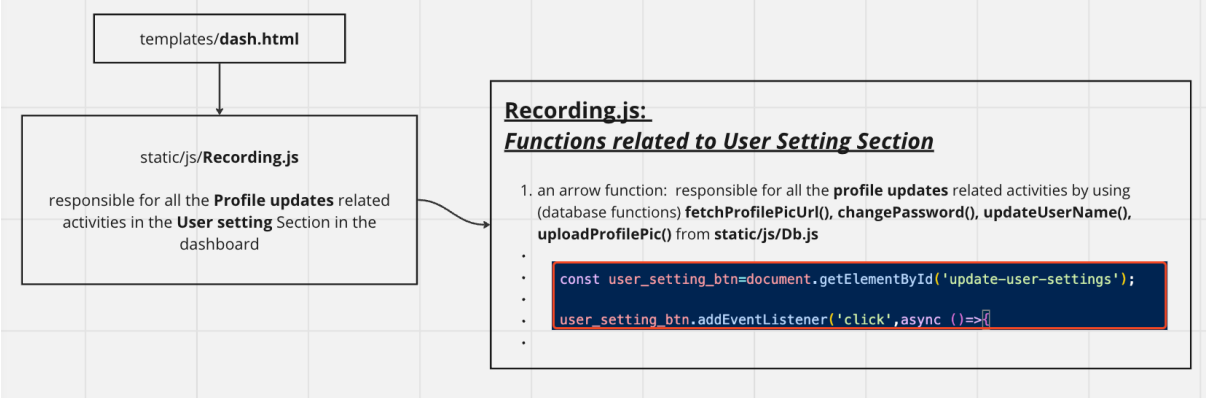
---------- END OF **Dashboard – My Diary** ----------

4. Dashboard – User Setting



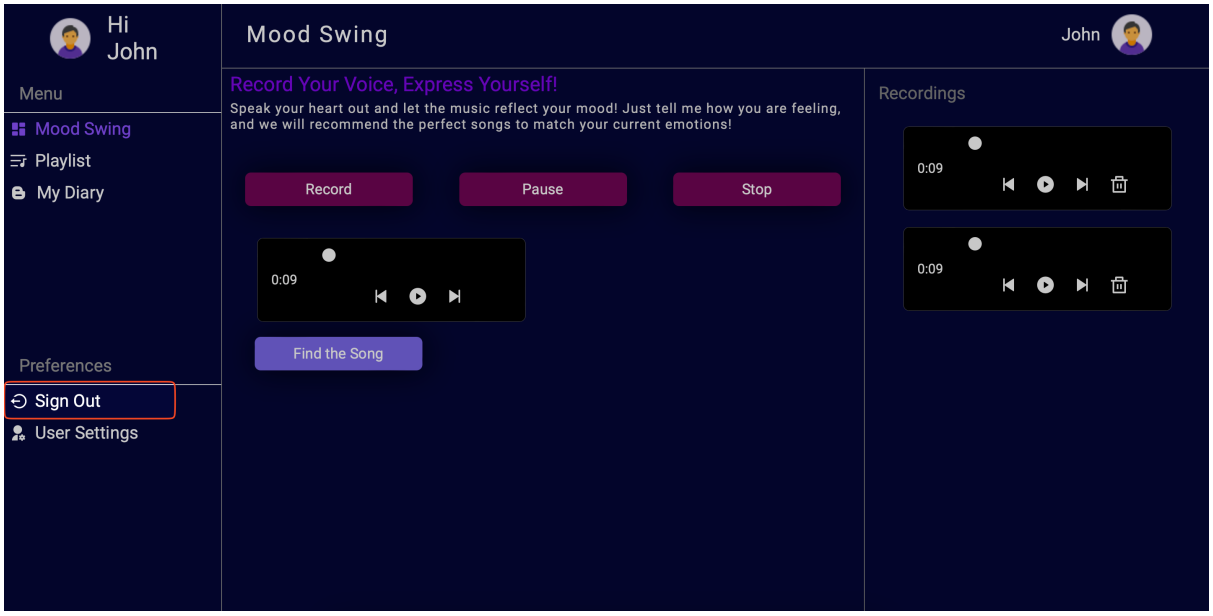
1. As Stated, dash.html file will use **static/js/Recording.js** as a script file.
2. This file is also have functions which is responsible for activities related to User Setting section in the dashboard (i.e change user name, change password, date of birth update, change profile picture).(refer the block diagram below)

NOTE: arrow function will use functions from **static/js/Db.js**



---------- END OF **Dashboard – User Setting** ----------

5. Dashboard – User Setting



Sign out will be taken care by the **static/js/MoodSwing.js.**

```
document.getElementById('sign-out').addEventListener('click',()=>{
  sessionStorage.setItem('Sentiment',null);
  localStorage.setItem('name',null);
  localStorage.setItem('User_id',null);
  window.location.href="/";
})
```

---------- END OF **Dashboard – Sign out** ----------

Appendix
(Function name will represent the functoriality of each function.)

Firebase is used for real time database, storage and authentication

Db.js *Functions*

function **writeToDatabase**(data)
export function **saveNameToDatabase**(userId, name)
export function **fetchNameFromDatabase**(userId)
export async function **saveRecording**(audioChunks,time)
export function **addBlog**(userId, content)
export async function **getAllBlogs**(userId)
export async function **deleteBlogEntry**(blogEntryId,userId)
export async function **changePassword**(newPassword)
export async function **updateUserName**(userId, newName)
export async function **uploadProfilePic**(userId, file)
export async function **fetchProfilePicUrl**(userId)
export async function **getRecordingDownloadURL**(uid, recordingId)
function **readFromDatabase**()
export function **addToPlaylistToDatabase**(previewUrl, mood, imageUrl, songName)
export async function **fetchAllMoods**()
export async function **fetchPlaylistsByMood**(mood)
export async function **fetchAllMoodsWithPlaylists**()
export async function **deletePlaylistFromDatabase**(userId, playlistsMood, playlistId)
function **writeToRecordings**(data)
export async function **deleteRecording**(userId, recordingId)
export async function **fetchDownloadURLsWithIds**(userId)

export {
auth,
createUserWithEmailAndPassword,
updateProfile,
signInWithEmailAndPassword,
writeToDatabase,
readFromDatabase,
writeToRecordings }

1. Spotify API Connection:

In script/Spotify.js,
There is **two function** responsible for this:

1. function authenticate()

```
function authenticate() {
const clientId = 'fbf8d98118b146f98869ecb2c789015b';
const redirectUri = encodeURIComponent('http://127.0.0.1:5000/dash');
const scopes = 'user-read-private user-read-email'; // Add any necessary scopes

window.location.href = `https://accounts.spotify.com/authorize?client_id=${clientId}&redirect_uri=${redirectUri}&scope=${scopes}&response_type=token`;
}
```

This function is responsible for initiating the Spotify authentication process.

It constructs a Spotify authorization URL (https://accounts.spotify.com/authorize) with specific parameters:

- **client_id:** The Spotify client ID (obtained from your Spotify Developer Dashboard).
- **redirect_uri**: The URL to which Spotify will redirect the user after authentication. This URL should match one of the redirect URIs configured in your Spotify application settings.
- **scope**: The required scopes (permissions) for accessing user data. In this case, the function requests permission to read the user's private information (e.g., profile and email).

After constructing the authorization URL, the function redirects the user to this URL using **window.location.href,** initiating the Spotify authentication flow.

2. function getAccessTokenFromHash()

```
function getAccessTokenFromHash() {
    const hash = window.location.hash.substring(1);
    const params = new URLSearchParams(hash);
    return params.get('access_token');
    }
```

After successful authentication and redirect back to your application's redirect_uri, call getAccessTokenFromHash() to extract the access token from the URL hash fragment and store/use it for making API requests to Spotify on behalf of the authenticated user.

2. Firebase Initialization:

Following script block will be found in both **template/index.html & template/dash.html**

```
<script>
    var firebaseConfig = {
        apiKey: "{{ firebase_config.apiKey }}",
        authDomain: "{{ firebase_config.authDomain }}",
        projectId: "{{ firebase_config.projectId }}",
        storageBucket: "{{ firebase_config.storageBucket }}",
        messagingSenderId: "{{ firebase_config.messagingSenderId }}",
        appId: "{{ firebase_config.appId }}",
        measurementId: "{{ firebase_config.measurementId }}"
    };
</script>
```

- This script block initializes a **JavaScript object** named **firebaseConfig** with Firebase configuration parameters.
- The values of these parameters are placeholders **({{ firebase_config.<param> }})** that will be dynamically replaced by values that defined/read from **app.py** (code block from app.py shown below)

```
firebase_config = {
        "apiKey": os.getenv('FIREBASE_API_KEY'),
        "authDomain": os.getenv('AUTH_DOMAIN'),
        "projectId": os.getenv('PROJECT_ID'),
        "storageBucket": os.getenv('STORAGE_BUCKET'),
        "messagingSenderId": os.getenv('MESSEAGING_SENDER_ID'),
        "appId": os.getenv('APP_ID'),
        "measurementId": os.getenv('MEASURMENT')
    }
```

NOTE: below 3 parts are not important for the project.

1. Explanation for taostify.js:

- **node modules/toastify-js → File is directly download from** https://github.com/apvarun/toastify-js

  ***this code has not been used in project

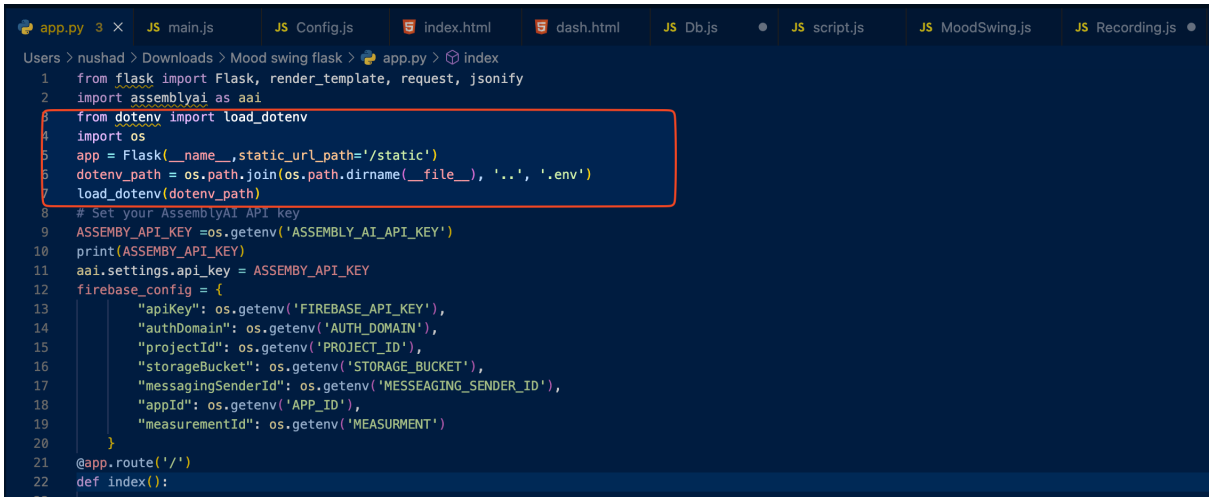
node modules/toastify-js/src/toastify.js

EXPLANATION FOR THE *node_modules/toastify-js/src/toastify.js*

- **Toastify** is a JavaScript library designed to implement **toast notifications**, which are **unobtrusive messages that appear temporarily on the screen** to provide information to users without disrupting their workflow. These notifications are commonly used across web applications, mobile apps, and desktop software to deliver updates or feedback discreetly.
- The library starts by encapsulating its functionality using a module pattern, ensuring **compatibility with both CommonJS environments (like Node.js) and browser environments.** The main `Toastify` function serves as the entry point for creating toast notifications, accepting various options to customize appearance, duration, and behavior.
- **Key components of the library include default options (`Toastify.defaults`) defining standard configurations for toast messages, an initialization object (`Toastify.lib.init`) managing toast instances, and functions (`buildToast`, `showToast`, `removeElement`) for constructing, displaying, and removing toast elements from the DOM.** These functions enable dynamic and interactive toast notifications, enhancing user experience by providing important updates in a user-friendly manner.
- In summary, Toastify empowers developers to integrate toast notifications seamlessly into their applications, offering flexibility in customization and straightforward APIs for creating and managing toast messages. The library's modular design and DOM interaction ensure efficient delivery of information to users without interrupting their activities, making it a valuable tool for enhancing user engagement and usability in modern software interfaces.

2. Explanation for dotenv:

- ***node module/dotenv file also not used in the project code
- But below part of the code in **app.py** uses **dotenv python module** *(not the one from node module/dotenv)*,
- but **we don't have a .env** file to configure the environment variable, **so we can skip that part in the code**.
- If you're working with a Flask application and you want to configure it without a .env file, you can directly set environment variables in your operating system.



3. Explanation for config.js:

- **config/config.js used to load environment variables from a .env** file into the Node.js environment using the dotenv package, and then export those variables as part of a configuration object.
- *** Since we don't have .env file in the project config part also won't be a part of the project.