Ample - New Technology Integration

Throughout the project's development, we investigated several new technologies to aid us in constructing the best project possible. The several which we chose, will be introduced and outlined below.

Three.js : https://threejs.org/
Three.js is a library that merges 3D computer graphics with web application development. It utilizes WebGL, a graphics engine built upon OpenGL; giving web applications the ability to create complex 3D raster graphics. The most important objects Three gives us access to are the renderer, the scene, and the camera. The renderer object allows us to draw polygons within the scene, and view them within the frustum of the camera. For this project, we combined Three's visualization capabilities with HTML-5's web audio api. This allowed us to visualize the current playing song with 3D shapes or images; in reference to our project, this shape is a sphere. The web audio api has an audio-analyser object, which allows us to extract audio data in place. We can then convert this data to a variety of useful forms that allow us to control our Three.js visualization and react to certain frequency intervals of the song. Finally, using this augmented data we built a real-time music visualizer to accompany our music streaming application.

The following contains some code snippets that showcase how we used this library throughout our program:

```
this.scene = new THREE.Scene()

// Create a new Three.js camera
this.camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000)
this.camera.position.z = 5
```

Here, we create a new scene object so that we can add our visual to the scene to get rendered. We add a camera position, which is essentially just a viewing angle of the object we wish to create on an x,y,z plane.

```
// Create a new Three.js geometry and material
this.geometry = new THREE.SphereGeometry(1, 32, 32);
this.material = new THREE.MeshBasicMaterial({ color: 0xffffff });

this.renderer.setSize(400, 400)
```

As seen above, we create a new geometry object, which is our sphere shape on the first line, and we specify the dimensions of it. We then gave it color and rendered it out onto a 400 by 400 canvas.

Axios - HTTP : https://axios-http.com/

Axios is a promise-based http client that integrates directly with Node.js. It allows for a very streamlined approach to http requests, and it includes a variety of simplifying features that improve over the native fetch api. Axios has near-identical syntax to the fetch api, with several key advantages. The main advantage that specifically impacted our project development is Axios' automatic json transformation. When using the fetch api, you have to convert the response-data into json via an explicit function call. Axios does this automatically, thus you have the response-data in an operable form right away. We took advantage of Axios all throughout our project, but it was mainly used to send http requests to a server holding our MongoDB database. We did this when generating the list of song options on window load, when checking for existing usernames on sign-up, and when validating username-password pairs for log-in.

Here are some examples of how the Axios http client was used within our project:

```
async getSongList() {
  try {
    const response = await axios.get('http://localhost:5000/api/songs');
    this.songs = response.data;

  } catch (error) {
    console.error("Error fetching song list", error);
  }
},
```

Above, we utilize Axios to send an http get-request in order to retrieve the list of songs from our MongoDB database. As you can see, the response data can be used directly without having to parse the response string into a json object.

```
await axios.get(`http://localhost:5000/api/users/userName/${username}`)
    .then((res) => {
        response = res.data;
    })
    .catch((err) => {
        console.error(err);
    });
```

Here, we again use Axios to send an http get-request, this time with url parameters to access the account tied to the specified username.