GeoMelody

Technical Specification

Student names:

1. Christopher Cussen (19424146)
2. Ali Mazbouh (18372503)

Project Supervisor:
Tomas Ward

Table of Contents:

I. Introduction

Overview of the Project

For our project we decided to create a progressive web application which will take audio input through a mobile phone and will then use an existing API like Aud-D to identify the inputted song, record where and when the user inputted the audio into the PWA, and display information on both the song and the artist. We also plan to implement neural network technologies to reduce any unwanted noise in the recording to improve the system's ability to identify a song playing in a noisy environment.

While similar applications do exist, like Shazam for example, this project will be a progressive web application which not only identifies the song for you, but will be able to store every song you identify in a database accompanied by a timestamp of when you recorded the audio and where. This will make it easier for the user to remember which song they heard when they try to find the song later on. As well as this, the web app will recommend similar artists for the user to listen to based on the song that was recorded.

Our reasoning for implementing this service as a progressive web application over a mobile application is that we want the service to be as easily accessible as possible. If a user is at a music festival, for example, and hears a song that they like but do not know they will only have a very finite amount of time to identify the song before it ends and their chance is gone. PWA's require no download and so if a user wants to identify a song but hasn't got the app, the likelihood that they succeed in this is greater if the service is available instantly on the web. Also, PWA's implement progressive improvement principles and so the service will work for any user regardless of their preferred browser. So whether it's an Android phone or IOS, the service will be instantly available.

II. Motivation

While music recognition apps have existed before ours, like Shazam for example, our project offers something new by providing a progressive web app that not only identifies songs for the user, but it also stores these recorded songs in a database accompanied by a timestamp of when it was recorded by the user and the location of when it was recorded. These additional features will make it easier for users to revisit and listen to the songs they love.

Furthermore, our project aims to improve the accuracy of the song identification process in noise polluted environments using advanced neural networks technologies, offering a more reliable and efficient service than these existing apps. Our motivation behind implementing this service as a progressive web application is to make it as easily accessible to everyone as possible, regardless of their preferred browser or device. We believe that by creating a PWA that offers more than just song identification, we can provide a valuable tool for music lovers everywhere to help them learn and discover new music in a simple way, enabling them to broaden their music library and enjoy the songs they love.

Advantages of using location services in our app:

In addition to offering an efficient and comprehensive music identification service, our project integrates location services using Google Maps API. The inclusion of this feature brings several advantages for our users:

Memory triggering
By recording the location and time of each identified song on a map, our app offers a visual representation of the user's listening journey, making it easier for them to remember where and when they recorded each song. This feature will particularly come in handy for situations where the user is surrounded by new music and is constantly using the app to identify a collection of new songs in a short period of time, e.g. music festivals. By recording the exact time and location of each song submitted to their specific search history, they can revisit the map and retrace their steps, helping them to remember the exact songs they recorded at each location.

Sentimentality
Providing location services would also help the user create a meaningful and sentimental connection with the music they love. By associating each recorded song with a specific time and location, the user can look back at the memories and emotions they felt when they first discovered the song. The idea behind this is to help the user create a more profound and personal connection with both the music and the memory of when this recording took place.

III. Research

In recent years with the exponential growth and improvements in machine learning technologies, music identification apps have become increasingly popular with several well-known apps currently available on the market. In this section, we will be discussing some of these applications, including Shazam, SoundHound and MusixMatch and comparing and contrasting them as we did in preparation for creating our own music recognition app.

Shazam

Having been one of the earliest applications in this field, Shazam has grown and evolved to an extremely popular and powerful application for music identification since it was founded in the year 2000. The app uses audio fingerprinting technologies to enable users to identify songs and purchase them from within the app. Additionally, Shazam offers a range of different features, including social sharing and personalised music recommendations based on their search history. Shazam is available on both the Apple Store and the Android Store.

SoundHound

Soundhound is another very popular music recognition service that uses audio fingerprinting to identify songs for the user. As well as this, SoundHouse actually allows the user to hum or sing a section of a song and it is able to identify it for them. This additional feature is extremely useful for users who can remember a song but not the name of it or the artist. Once a song is recorded, the app's response to the user contains the name of the song, as well as the name of the artist, the song's lyrics, the album artwork, and similar music recommendations for the user. This app is available on both the App Store and the Android Store.

MusixMatch

MusixMatch is a music player and a lyrics app that also provides music recognition services. Like Shazam and SoundHound, this app also uses audio fingerprinting to identify the recorded songs for the user, and once a song is identified the app relays this information to the user with additional information like the name of the artist, lyrical content of the song and the album art. As well as this, the app can also be integrated with different streaming services like Apple Music and Spotify to lead the user to the page of the artist whose song was identified, helping the user learn more about the artist and broaden their music knowledge.

Upon comparing these various apps when doing research in this field for our own project, several similarities and differences became apparent. First of all, all three of these apps use machine learning, more specifically audio fingerprinting, as their method for accurately identifying the recorded song. Audio fingerprinting is essentially "a condensed summary in audio form of an audio signal that can be used to identify the file quickly or locate at speed items that are similar to it in a database of sounds/music"[1]. This process involves analysing the audio signal to extract features like the song's tempo, rhythm, pitch and other attributes that are unique to the specific piece of audio being examined. Once extracted, these features are used to generate a unique fingerprint ID which can be compared to existing song's fingerprints in a database to successfully identify the name of the song and information about it. So, upon learning this, it became clear that implementing such technology in our own project is essential to creating an accurate and reliable music recognition service

These different apps also have some key differences. Shazam's primary focus is on music recognition, with additional features like providing some music recommendations based on the user's search history. SoundHound provides this functionality, as well as being able to identify a given song based solely on melody as made evident by its ability to name a song based on the user humming or singing it from memory. MusixMatch on the other hand is primarily a lyrics app, with music recognition available as a feature of this. Being aware of these similarities and differences is essential for us to create an app that performs similarly to these apps, while also offering something new and fresh to the market.

Compared to these existing services, our PWA offers a unique advantage in terms of accessibility and convenience. Unlikely the apps mentioned above, our app doesn't require any download, making it easily accessible to anyone with internet access. This aspect of our app makes it particularly favourable in situations where the user only has a finite amount of

time or data to use the service before their chance is gone. Also, our app enables users to relate their search history to a specific time and place, a feature that is not available in the existing apps in the market that we have researched. This allows users to easily access and remember any song that they recorded in the past, even if they didn't immediately take note of when and where they recorded the song after identification.

Technologies and Frameworks Used

To develop our progressive web application, we employed several different technologies. First, however, we had some decisions to make in terms of what technologies to be using. E.g. What programming languages should be used? How do we get our app working on all devices (IOS, Android)? Should we use a framework to create our app and if so which one? The approach we eventually decided on was simply to create a mobile app with vanilla javascript, html and css that runs inside the browser like a standard website, but harnesses the power of native device features so that when it is run on a browser it acts and behaves like a native mobile app. Such functionalities include being able to access the app offline, being accessible from the home screen, etc.

Vanilla Javascript

Vanilla Javascript is a pure, unadulterated form of Javascript without the need of any third-party libraries or frameworks. The main reasons why we chose to implement Vanilla Javascript as opposed to other popular frameworks:

1. **Speed** → Vanilla Javascript offers faster loading times and better performance compared to most frameworks, and as speed is a priority for our service we felt this suited the requirements of our project well.
2. **Control** → after researching the topic, we discovered that using Vanilla Javascript allows more control over the project for the developers. Without the layers of restrictions and limitations imposed by using the likes of React, Vanilla Javascript would allow us to customise our code to our specific needs in a simple and sustainable way, making it easier for us to maintain the project over time.
3. **Lack of dependencies** → The lack of dependencies for Vanilla Javascript means that the app will not be affected by any external library changes or updates that are out of our control. This fact makes it a reliable and stable choice for developing our PWA as it helps us to enable the application to be usable across multiple devices and platforms.

Firestore Database

Firestore database is a cloud-based NoSQL document-oriented database. We chose to use Firestore database for the backend of our application due to its real-time synchronisation, scalability and ease of use.

1. **Real-time synchronisation** → The real-time synchronisation feature of firestore means that any changes to the database are instantly updated across all connected clients. This is essential for our project as it means that once a song is successfully identified and pushed to the database, this information is made immediately available to the user in the search history page of GeoMelody.

2. **Scalability** → Firestore database is designed to scale horizontally, meaning that it can easily handle large amounts of data and traffic without any negative impact on performance. This accommodates for a growing user search history without it affecting the speed or efficiency of the service.
3. **Ease of use** → Considering we had a very finite time to develop this app, finding the quickest, yet efficient, approach to creating it successfully was key to developing this pwa on time. Firestore's easy-to-use database requires minimal setup and configuration, and it is designed to be learned simply and easily for developers like ourselves who are new to designing and creating progressive web applications. As well as this, Firestore offers robust security and authentication features that ensure that a user's data remains safe and user-specific.
4. **Offline capabilities** → Firestore also offers robust offline capabilities, meaning that users can continue to use the app even when they aren't connected to the internet. This seamless user-experience regarding the user's connectivity status is a key feature of a progressive web app, and so Firestore database accommodates for this too.

Node.js

Node.js is a runtime environment for JavaScript that allows JavaScript code to be executed on the server-side of the project. As we are using Vanilla Javascript for the frontend of our project, Node.js seemed to be a suitable choice for building the backend of our project as it allows us to keep the same language and code-base throughout the project, making it easier to update and maintain our code. Other advantages for us using Node js include:

1. **It's lightweight and fast** → Node.js is built on a V8 javascript engine. This allows for javascript to be compiled into machine code which provides faster code execution. As stated, speed is a priority for our project so this fact about Node.js caught our attention very quickly.
2. **Scalability** → Node.js is designed to deal with large amounts of data very well, making it an appropriate choice for our project which would deal with large amounts of user data as their search history grows overtime
3. **Offered resources** → Node.js offers plenty of resources and libraries through npm that can be utilised to improve and enhance the functionality of our project.

HTML

HTML is an essential language for building PWAs, used to provide the content and structure of the web pages within the PWA. We used this language to define the UI (User Interface) of our website, along with css. This includes defining the content of the pages and how they are laid out, as well as optimising the UI for both desktop and mobile devices:

1. **Separation of concerns** → While JavaScript is responsible for adding interactivity and dynamic behaviour to the web pages, HTML's job in our project is to define the content and structure of our PWA. By separating these concerns, it allowed us to more

easily create a maintainable and scalable codebase. This is also beneficial for us when refactoring any aspects of the code for our project.

2. **Caches and Service Workers** → As stated, an essential aspect of a PWA is its offline-first experience it provides. By designing caches and a service worker for our project, we were able to store certain parts of the app's content locally on the user's device so that it is available and displayable to the user even without internet connection.
3. **CSS integration** → HTML works seamlessly with css, enabling us to design the look and feel of our user interface. We chose to integrate Materialize CSS which is based on Google's Material Design.

CSS

As stated above, we chose to implement Google's Materialize CSS in our PWA, enabling us to style and layout its content making it visually appealing and engaging for our users. Reasons for why this is essential to our project:

1. **Consistency** → CSS allowed us to layout and design GeoMelody in the way we wanted, that is engaging and consistent. Any well designed website / PWA / mobile app needs consistency in regards to it's design, and using css to style our pages headers, fonts, colours etc gives our PWA a cohesive feel that makes it easier for users to navigate and understand it.
2. **Separation of concerns** → Using CSS to style our PWA allows us to separate the design aspects of the project from the underlying structure and functionality of it done through HTML and javaScript. This makes it easier for us to develop, update and maintain the style of our PWA in an organised and efficient manner.
3. **Browser compatibility** → Similar to HTML, CSS is compatible with any browser which makes it great for making PWAs. It means the design and style of the project will be consistent across a wide range of devices and platforms.

Aud-D

Aud-D is the audio recognition API we decided to use for our PWA. It allows us to add music recognition features to our project. After some research into different API's we decided to use Aud-D for the following reasons:

1. **Accurate music recognition** → Aud-D uses a combination of acoustic fingerprinting and machine learning algorithms like audio fingerprinting, as is done in the existing music recognition apps mentioned above, to identify songs.
2. **Easy integration** → Aud-D is a RESTful API, meaning that it is optimised for good communication between the client and server using HTML. So, when the user sends a HTTP request to the Aud-D, its response is returned in a JSON format which makes it easy to parse and integrate in the app.
3. **Developer-friendly** → Aud-D offers a range of developer-friendly features, including documentation, SDK's and community support, giving us a step by step plan for integrating the API successfully with our PWA.

Google Maps API

For the location services used in our PWA, we chose to use Google Maps API, which is a powerful and reliable location service that allows us to find and record the location of the user after successfully identifying a song. There are several reasons why we landed on using this location service API:

1. **Comprehensive Mapping data** → This includes detailed maps which can accurately locate and display the exact location of the user when they used the PWA to identify a song. This is a key outstanding feature of our project so accuracy is key, and this API provides this very well. From this, we were also able to extract specific pieces of data like street names and specific addresses to display next to the name of the song in the search results
2. **Developer-Friendly** → The API is generally comprehensive and easy to understand and is deployable in a range of languages, including Node.js, making it relatively easy for us to implement it in our project.
3. **Cost-effective** → while implementation of this API in our PWA wasn't free, it's pricing plan was very reasonable compared to other similar APIs and so this suited us well.

IV. Design

System Architecture

The GeoMelody architecture was designed for speed and efficiency, to allow the user to quickly log in and click record to start identifying whatever song is playing around them. The UI is also designed to be clean and simple to avoid any confusion or misunderstandings for the user when navigating the PWA.

Frontend Design

The frontend of GeoMelody is optimised to be clean and simple. As the sole function of our project is to identify songs and record the user's search history, there was no need to overcomplicate the UI design which could lead to confusion for the user, wasting what could be a very finite period of time they have to identify the song. As well as this, it's designed to be visually appealing and engaging for the user. We developed the front end using HTML, Materialize CSS and Vanilla javascript. The UI consists of four pages in total: the login / sign up page, the home page, the user's search history page, and an about page.

Once the user accesses the site, they are immediately prompted to either login or create an account. The login page is optimised to remember a previous user who logged in on that device, so this process should not be a long one.
Once logged in, the user is brought to the home page of the PWA which consists of the record button which is located at the centre of the page to immediately draw the user's attention to it, and a map beneath it which jumps to the user's current location as soon as the song identification is successfully carried out. On top of this page, as well as the search history and about pages, is a navigation bar which shows the name of the current user logged in as well as a button to access the side navigation bar. Here is where the user can access the other pages of the site with ease.
The user's search history page consists of a table with 4 columns. The first column displays the name of the artists of the songs identified, the second column displays the names of each song successfully identified by the user, the third column shows the timestamp of exactly when the song was recorded by the user, and the final column shows the address of where the user was when the song was identified. All of the songs in this search history table are ordered in chronological order to organise their search history as efficiently as possible.

Backend Design

The backend of our project has a few roles: to act as a database to store the user's search history and implementing the Aud-D and Google Maps APIs that allow the front end components of the PWA to send the GET requests to them and get a response from them returned. Our backend has been developed in Node.js, the runtime environment that allows our JavaScript code to run on the server-side.

1. **Firestore Database** → The firestore database has two functionalities in our PWA: to store the user's search history data, and user authentication so that they can log in and out of the site. This is necessary to ensure that the user's search history is specific to themselves and nobody else can add to it.
    a. **Firestore's real-time database** → This database is designed to store and synchronise the music recognition responses in real-time. When the user records a song for recognition, the PWA can send the audio data to Aud-D for recognition and then store the API's response in FireStore's real-time database which is user-specific and made instantly available to view in the user's unique search history page. The real-time database is designed to handle large volumes of data and scale automatically, so it can accommodate multiple users to make their requests to the API and store their responses simultaneously, ensuring that the PWA remains responsive and scalable.
    b. **Firebase Authentication** → Firestore provides an easy-to-use authentication service that can be used to create user accounts for the PWA, authenticate the user and to store data that is specific to the user. To enable this service, we enabled Firebase Authentication inside the Firebase project settings and included the SDK in the *SignIn.js* file of our project, which handles the interactivity and dynamic behaviour of the sign in page of the PWA. Once

implemented, the SDK allowed us to develop the PWA so that the user had to authenticate themselves before they could log into their account, and once logged in all of the data that is pushed to the database is pushed to a specific collection that is unique and only accessible to the current user.

2. **API Development** → The backend also includes the integration of the APIs (Application Programme Interfaces) Aud-D and Google Maps. These APIs are used for the music recognition and location services of our project. This included importing the Axios library to make HTTP requests to the APIs, and adding an event listener to the record button of the home page which would trigger the PWA to send the GET request to Aud-D with the audio recording as well as simultaneously obtaining the user's current location from the Google Maps API.

User Interface

The UI of our project is a critical component of its success. Users expect a clean, intuitive and easy-to-use interface that allows them to record a song quickly and efficiently. One of the best ways of doing this is to minimise the clutter and provide the users only with the display information and controls that they need to use the app effectively. A clean UI would improve the overall user experience, making the app easier to use and leading to a higher user engagement and retention rate.

The UI framework we used is Materialize CSS, which provides pre-designed UI components that follow the principles of Material Design, a design language developed by Google. This design language emphasises simplicity, minimalism and the use of bold, colourful visual cues to help the user navigate the interface.

Adapting to different displays is a crucial necessity for a PWA to provide a consistent and optimal experience for the users across different devices. To achieve this, we used a manifest file and responsive web design techniques. A manifest file is a JSON file that provides metadata about the PWA, such as its name, icon, and theme colour By including a manifest file in our PWA, we were able to control how the application is displayed when it's installed on a user's device. This includes defining how the app is launched, as well as specifying the app's orientation, display mode and other features.

Responsive web design is a web development approach that aims to create websites and PWAs that adapt to different screen sizes and resolutions. With responsive design, the layout of GeoMelody changes dynamically based on the size and orientation of the screen. This allows the content to be presented in such a way that it is optimised for the device it is run on, making it more versatile and accessible to users.

Website Design

In designing the layout of GeoMelody, we adhered to the same principles mentioned above to keep the layout clean and simple, as this directly impacts the user's experience. We designed GeoMelody's layout to focus on consistency, simplicity and intuition. We designed the UI

using a clean colour pallet with three different shades of purple to match GeoMelody's app icon.

1. **Login Page** → The initial login page appears clean and uncluttered with a set of clear forms, one to login and one to create an account. This page is the first one the user will encounter when using the app, so it sets the tone of our project and therefore it's very important to keep it engaging, as it's important to maintain a consistent design throughout once the user logs in and enters the main bulk of the PWA.
2. **Home Page** → GeoMelody's home page, hosting the primary functionality of GeoMelody, is where the user can record and identify the song. Its layout is designed to be intuitive and user-friendly, with a prominent navigation bar and side navigation (as is consistent throughout the project), the record button positioned in the centre of the display, and a map beneath it that jumps to the user's current location upon successful song identification. Visual cues such as notifications are used on this page to inform the user on the name of the song and the artist of the recorded song upon identification as soon as it is added to the database of recorded songs.
3. **Search History Page** → this page was designed to be clean and to display all of the relevant information regarding the content of the user's unique data collection in the Firestore database. Using a table from Materialize CSS, we were able to create a tidy layout with 4 columns: Artist, Song, Time, and Location. The data for each identified song is shown here in a way that is uncluttered, with plenty of white space and bordering to help separate each item and make the data easy to differentiate and read.
4. **About Page** → Here is where the user can read about GeoMelody and its functionality. The layout is clean and straightforward.

Overall, GeoMelody's layout has been designed with consistency, simplicity and responsiveness in mind with the help of an organised hierarchy and HTML approach, Materialize CSS, and a working manifest file. This ensures that it looks and works well on any device, regardless of screen size. It is also organised in a logical and intuitive way to make it easier for the users to satisfy their expectations for the app.

V. Implementation

FRONTEND

The front end of our PWA was implemented using a combination of Vanilla Javascript, HTML and Materialize CSS. Vanilla Javascript is a pure, unadulterated form of Javascript without the need of any third-party libraries or frameworks and we utilised this to achieve the benefits of speed, control, performance and freedom that it provides.

All of the pages for the PWA are implemented using standard HTML components such as the boilerplate which includes all of the set meta tags and the title. The PWAs CSS page is also imported here

Sign In page

1. **index.html** → The contents of this page consists of two cards that contain forms for logging in and signing up for a GeoMelody account. Each form includes two input fields for the user's email and password, along with a submit button. Overall, this implementation of the Sign In page for the front end is fairly straightforward, using standard HTML and CSS styles along with importing a JavaScript file which provides the interactivity and functionality for the page.
2. **SignIn.js** → this file uses Firebase Authentication and Firestore to create a simple sign in page for GeoMelody which gets imported to the SignIn.html page for use. This includes all of the imported Firebase modules which links our Firebase project to the page and provides methods that can be applied to the database.
    a. **getAuth** → After initializing the database with the configuration object, the getAuth function is used to create an auth instance to manage user authentication.
    b. **getFirestore** → A function that creates an instance of the database.
    c. **Button event listener** → event listener to the sign up and login buttons.
    d. **CreateUserWithEmailAndPassword** → When the user submits the sign up form, it gets the email and password values and uses CreateUserWithEmailAndPassword function to create a new user in the Firebase Authentication. If the sign up process is successful, it logs a new user object to the database and resets the form. If there is an error, however, it logs the error message to the console.
    e. **signInWithEmailAndPassword** → Once the user inputs their details into the email and password fields of the login form, it uses the signInWithEmailAndPassword function to authenticate the user. If authentication is successful, it logs the user object and redirects the user to a specified user and allows them access to the home page of GeoMelody, otherwise an error message gets logged.
    f. **onAuthStateChanged** → used to track any changes to the authentication state of the user. If there is a user, it displays a welcome message on the page, otherwise, it clears the message.

Home Page

1. **SignIn..html** → This code contains the contents of the home page of the PWA and it includes several scripts to enable functionality such as navigation, music recognition and Google Maps integration. This code also defines a button for music recognition which, once clicked, will begin to record and then identify the song inputted by the

user. Beneath that lies the implementation of a Google Map which gets initialized by the initMap() function of the GoogleMaps API.

2. **Index.js** → This code implements the Firebase services Authentication and Firestore, as well as the Google Maps API and the Aud-D API. This also includes the functionality of the button defined in the html page. Once the button is clicked, the code retrieves the user's location and address using the Google Maps API. It then retrieves the audio input from the user and sends a POST request to the Aud-D API endpoint with the address parameter. When the response is received, the artist and title of the recognised song are extracted and added to the Firestore subcollection "data" for the current user with a timestamp as well as the user's current location found by the Google Map API. All of this communication is accommodated through the Axios library
    a. **firebaseConfig** → Initializes the Firebase services.
    b. **getFirestore** → Sets up a Firebase instance to get called on
    c. **onAuthStateChanged** → Gets the current ID and checks if the new user already exists in the Firestore user collection If this is the user's first time using the app with a new account, a new document is created for the user with a subcollection named "data" where the user can store their own specific search results. The code then creates a new collection reference to "songs" in the subcollection of the user and sets up a query to order the collection by creation time.

Search History Page

1. **History.html** → This code is the html implementation of my search history page. The main content of the page is a table that displays the user's search history. The table includes four columns for the artist, title, time and location of the recognised song and where and when it was recorded. The table is initially empty for a new user, and gets populated over time as the user uses the app more. The javascript code in this file imports the necessary Firebase SDKs for Firestore and Authentication. It listens to see if the user is logged in and when it sees that they are, the user is authenticated and the code retrieves the user's search history from Firestore and populates the table with the results.
    a. **AddItemToTable** → takes four parameters representing the artist, title, time and location of a recognised song and appends a new row to the table with the corresponding values.
    b. **AddAllItemsTotable** → clears the table and populates it with all the recognised songs in the user's search history by calling AddItemToTable for each item.

About Page

1. **About.html** → This page is a simple static page that contains information about the PWA and its functionality for the user.

Css

1. **Styles.css / Style2.css / Style3.css**→ This page contains styles that are applied to the elements of my PWA project. The styles here are responsible for defining the layout and appearance of various different components of GeoMelody including the navigation bar, side-navigation bar, buttons, forms, tables, cards, etc. More specifically, it uses classes and selectors from the Materialize CSS library as well as some custom css variables defined by us. The purpose of this page as well as the implementation of Materialize CSS is to maintain consistency in the PWA in terms of colour, style and overall feel.

Service Worker

1. Sw.js → This service worker file is a key component of our PWA and it provides several benefits to the application including loading content both online and offline, which improves the user experience and reliability of the application. This is done by using a cache with specific assets including all of the html and css files, images, etc. With these assets, the cache can intercept fetch requests made by the user and decide whether to return cached content or to make network requests. This helps to reduce the amount of network traffic, speed up the application and decrease data usage.

Webpack Bundler

1. Webpack.config.js → This file is a configuration file for wepack, which is a popular module bundler for JavaScript applications. In short, a module bundler is a tool that takes multiple separate JavaScript files and their dependencies and packages them together into an optimised and minified bundle file. The purpose of this file is to reduce the number of requests that the browser needs to make to the server to download JavaScript files. Instead of loading multiple separate files, the browser can load one or more bundle files that contain all of the necessary code. This plays a role in optimising the code for faster loading times and overall better performance.

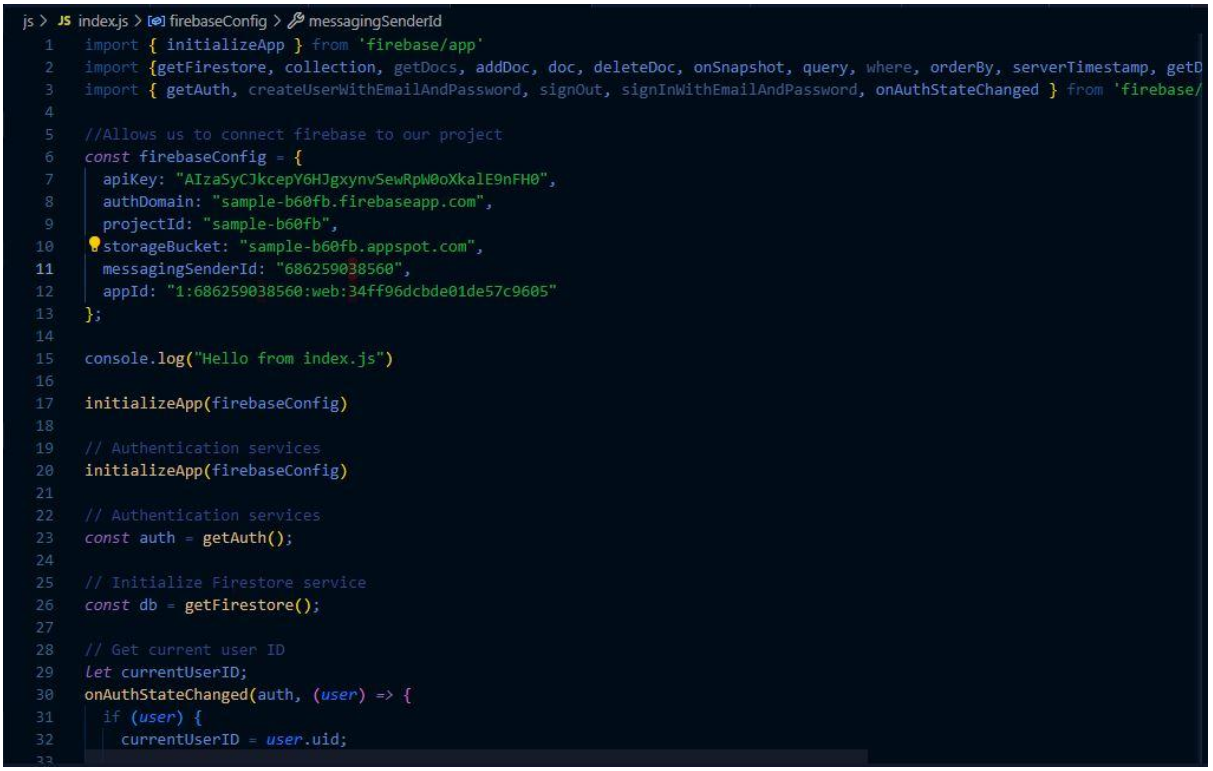
VI. Sample Code

Index.js:

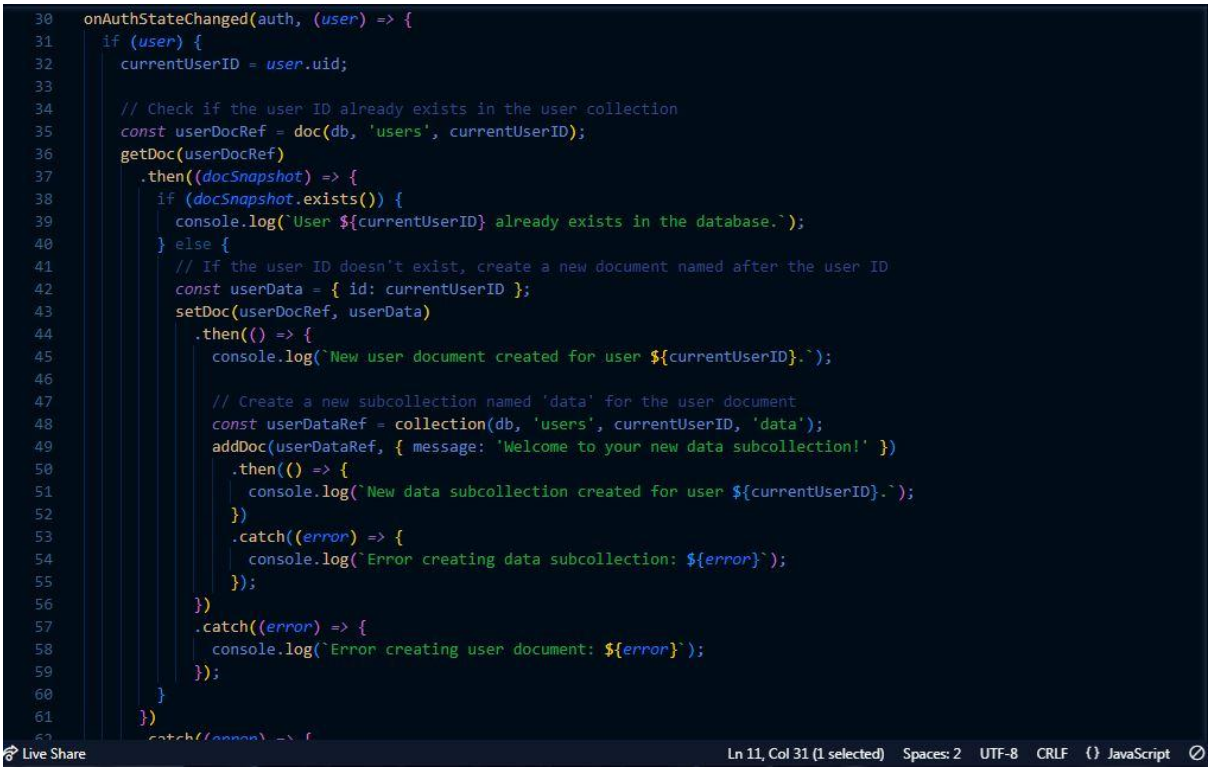
This file contains the code for the home page of the application where the song identification process takes place. The basic functionality of this page includes linking it to the Firestore database as well as the Aud-D API and the Google Maps API so that when the user clicks the record button, the song playing will be used as input for the application, get sent to Aud-D where it will be processed and eventually a response will be returned to the user with all of

the relevant information. The code then extracts the Artist's name and the name of the song from this information and adds it to the Firestore database where it will be displayed to the user in the search history page.

This screenshot of the code below is the beginning of the index.js file, where the required Firebase libraries are imported and initialised with the Firebase configuration object. This includes the project's unique key, authentication domain, project ID, etc.



This next part of the code sets up a listener using Firebase's onAuthStateChanged method to detect any changes in the authentication state of the application. This is so if it is a new user using the PWA, a new user ID is created and appended to the database so that whatever data that they add to the database through using the app will be user specific and only accessible to them. If the user already exists then this step is skipped.



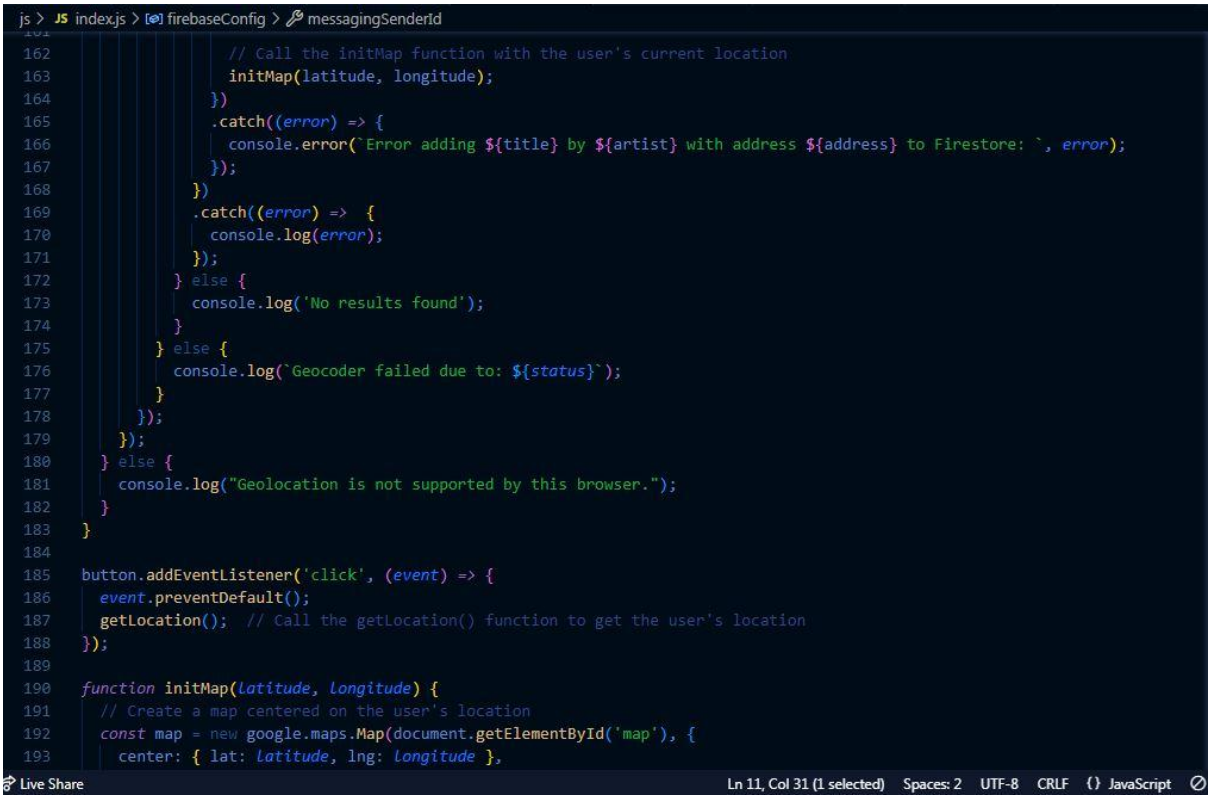
This section of the index.js file is where the Aud-D API and the Google Maps API are implemented into the code. As seen below, we first had to import the axios library to allow for HTTP requests to be made, then inside the getLocation() function the code retrieves the user's current location based on their longitude and latitude coordinates and this is added to the data object that will be displayed in the search history page along with the Aud-D response. A POST request is then sent to the API endpoint and once the song is processed, the response is added to the Firestore database using the addDoc() method. Finally, the code calls the initMap() function with the user's location and displays this on the corresponding html page once the song identification has been accomplished, showing the location of the user when they recorded the song. This address is also appended to the data object of the song in the database to be displayed in the search history page.

```
96
97  ////////////////////////////////////////////////////////////////////////////
98  //////////////////////////// AUD-D API / Google Maps API ///////////////////
99  ////////////////////////////////////////////////////////////////////////////
100
101 const button = document.getElementById('audD');
102 var axios = require("axios");
103
104 function getLocation() {
105   if (navigator.geolocation) {
106     navigator.geolocation.getCurrentPosition((position) => {
107       const latitude = position.coords.latitude;
108       const longitude = position.coords.longitude;
109       console.log(`Latitude: ${latitude}, Longitude: ${longitude}`);
110
111       const geocoder = new google.maps.Geocoder();
112       const latlng = { lat: latitude, lng: longitude };
113
114       geocoder.geocode({ location: latlng, language: 'en'  }, (results, status) => {
115         if (status === "OK") {
116           if (results[0]) {
117             const address = results[0].formatted_address;
118             console.log(`Address: ${address}`);
119
120             const url = document.getElementById('audioUrl').value;
121             const data = {
122               'api_token': 'ea9ce5f98f4ac6388733c8efe213c884',
123               'url': url,
124               'accurate_offsets': 'true',
125               'skip': '3',
126               'every': '1',
127               'address': address // Add address parameter to API request
```

Live Share Ln 11, Col 31 (1 selected) Spaces: 2 UTF-8 CRLF {} JavaScript

```
js > JS index.js > [@] firebaseConfig > messagingSenderId
130             axios({
131               method: 'post',
132               url: 'https://enterprise.audd.io/',
133               data: data,
134               headers: { 'Content-Type': 'multipart/form-data' },
135             })
136             .then((response) => {
137               const artist = response.data.result[0].songs[0].artist;
138               const title = response.data.result[0].songs[0].title;
139               const userDocRef = doc(db, "users", currentUserID);
140               addDoc(collection(userDocRef, "data"), {
141                 Artist: artist,
142                 Title: title,
143                 Address: address,
144                 createdAt: serverTimestamp()
145               })
146               .then(() => {
147                 // rest of the code
148               })
149               .catch((error) => {
150                 console.error(`Error adding ${title} by ${artist} with address ${address} to Firestore: `, error);
151               })
152               .then(() => {
153                 //console.log(`Added ${title} by ${artist} with address ${address} to Firestore`);
154                 const notification = document.createElement('div');
155                 notification.classList.add('notification');
156                 notification.textContent = `This song is ${title} by ${artist}`;
157                 document.body.appendChild(notification);
158                 setTimeout(() => {
159                   notification.remove();
160                 }, 5000);
161
162                 // Call the initMap function with the user's current location
```

Live Share Ln 11, Col 31 (1 selected) Spaces: 2 UTF-8 CRLF {} JavaScript



VII. Problems Solved

While developing this project, there were a number of problems and implications that had to be dealt with along the way. This includes complications arising with the API, creating the bundle files and having the cache work as intended. Thankfully, through research as well as trial and error, we were able to solve these issues:

API Complications:

When we first began to implement the API for our project, we decided on using RapidAPI's Shazam API as it was free and according to our research, and user feedback, the API is a reliable one. However, after attempting to implement this we encountered some difficulties:

1. Technical Issues → there were issues with the RapidAPI Shazam API in terms of integration. We were still unsure as to what the actual problem was but as far as we were aware we were implementing the SDK correctly and we followed all of the documentation. Despite this however, we were unsuccessful with this API and after some time we decided to change our approach and experiment with a new API.
2. Cost → There was evidence online that Aud-D is a reliable and efficient choice, but due to the pricing plan we initially overlooked it. However, due to the circumstances,

we decided to subscribe to it and shortly afterwards we had the API responding successfully.

Asynchronous loading of scripts

When designing the record button of the project, our aim was to have the button working in a way that sent the POST request to Aud-D, have the current location of the user through the Google Maps API found as well as adding the responses to the firestore database upon completion. Our initial approach to this however didn't work as intended due to the asynchronous loading of scripts. As far as we could tell, the database wouldn't log the responses of these two APIs simultaneously. As a way of solving this problem, we did some research and found that using a bundle file would correct this. Bundling is the process of taking multiple Javascript files and combining them into a single file. This helped us to have the database logging the API responses in the same function which sent the requests.

VIII. Results

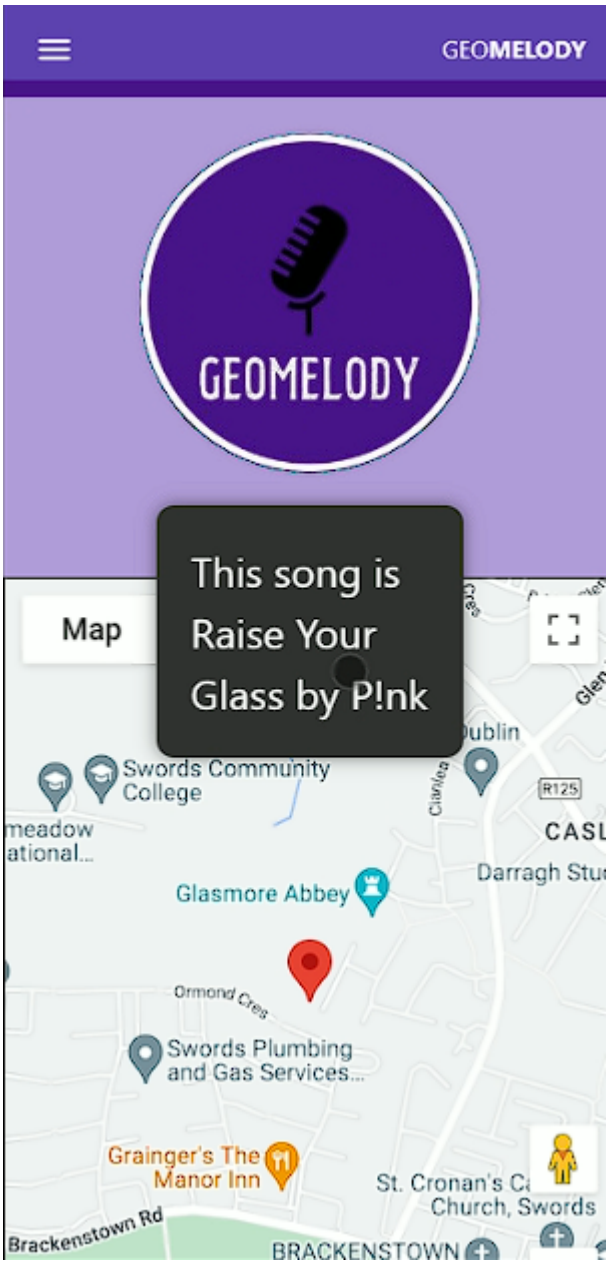
The index.js file comprises the song identification logic for the application, which involves integrating it with the Firestore database, the Aud-D API, and the Google Maps API. When the user clicks on the record button, the currently playing song is used as input for the application. It is then sent to the Aud-D API for processing, and a response is returned to the user with all the pertinent information. The code extracts the song's name and the artist's name from this data and stores it in the Firestore database. The user can view this information, along with their location data obtained via the Google Maps API, in the search history page.

To ensure that user-specific data is saved, the application generates a new user ID and appends it to the database using Firebase's onAuthStateChanged method. Combining multiple JavaScript files into a single file enabled us to implement the APIs and log responses to the database within the same function.

During the project's development, we faced some challenges, such as technical problems with the RapidAPI Shazam API and asynchronous script loading. We addressed these challenges by switching to a new API, Aud-D, and bundling multiple JavaScript files into a single file, allowing us to log responses to the database in the same function.

Overall, the implementation of the song identification process using the Aud-D API and the Google Maps API was successful. The database logging of responses in the same function provided a seamless user experience.

Here is a screenshot of the successful scanning of a song:



As you can see it gives the users current location along with the name of the song and the artist.

IX. Future Work

Despite the project's success in identifying the recorded song and logging this information to the database, as well as the user's current location and a corresponding timestamp, there are some additional features that we would have ideally liked to have implemented had we more time:

1. Improved User Interface → While we believe the current UI fulfils our aim for the project to appear clean, simple and concise, we believe that with more time we could add to this to create an even more engaging user experience. For example, implementing a more interactive interface with the inclusion of more animations could improve the overall user experience and draw more users back to the application for future use.
2. Implementing more advanced search capabilities → As demonstrated by the SoundHound music recognition survive, the possibility of humming or singing a song into the microphone for recognition could have been explored more and with more time to develop GeoMelody we would like to achieve this level of search capability too.

X. References

[1] → https://www.intrasonics.com/news/2020-11-09-what-is-audio-fingerprinting-when-where-and-why-you-should-use-it

[2] → https://www.javatpoint.com/what-is-vanilla-javascript

[3] →https://firebase.google.com/

[4] →https://docs.audd.io/

[5] →https://developers.google.com/maps/documentation/elevation/cloud-setup

[6] →https://nodejs.org/en/docs