CA400 Technical Guide

ArtistConnect

An app that allows artists and fans to connect and collaborate

Student 1 - Karl Doherty 19413086

Student 2 - Yury Chupahin 19416764

Project Supervisor: Dr Gareth Jones

Table of Contents

1. Motivation

1.1. Purpose of the Project

ArtistConnect is a social media application designed to connect music enthusiasts and provide a platform to share their passion for music. With ArtistConnect, users can post songs and events they are interested in, share their thoughts and opinions, and connect with friends and other users with similar music tastes. The main purpose of this project is to create a user-friendly and engaging social media platform that caters specifically to music enthusiasts.

The idea behind ArtistConnect is to provide a platform for users to discover new music and connect with like-minded individuals, while also allowing them to share their own musical experiences. By creating a community of music lovers, ArtistConnect aims to bring people together and foster a sense of belonging among its users.

1.2. Significance of the Project to the Field of Study

The field of social media has experienced significant growth in recent years, with new platforms being developed every day. However, most of these platforms cater to a general audience and do not provide specific features for users with specialised interests, such as music lovers. ArtistConnect aims to bridge this gap by providing a social media platform that caters specifically to music enthusiasts, offering features that are tailored to their needs.
By creating a social media platform that is dedicated to music, the project also contributes to the field of study by providing insights into the development of specialised social media applications and how they can be designed to cater to specific audiences.

1.3. Benefits of the Project for the Target Users

Connect with like-minded people: ArtistConnect allows users to connect with people who share similar music tastes, enabling them to discover new music, artists and concerts that they might have missed otherwise.
Discover new music: Users can explore new music based on their interests, as well as recommendations from their friends and the application's recommendation system.
Find events: The application enables users to view events and concerts happening in their area, as well as share their experiences with their friends.

Express themselves: Users can share their music and thoughts with a community of like-minded people, allowing them to express themselves freely and receive feedback.
Personalised recommendations: The application's recommendation system uses machine learning from Spotify's developer API to provide personalised recommendations based on a number of configurable parameters, enabling users to discover new artists and music that they may enjoy.

1.4. Objectives of the Project

To create a social media platform for music enthusiasts: The primary objective of the project is to design and develop a social media platform specifically tailored for music lovers, where they can connect, share their music interests and find out where to purchase tickets to events.
To implement a recommendation system based on song provided by user: Another objective of the project is to allow users to use a recommendation system that will suggest new music and artists based on a given song which is configurable from a number of different parameters.
To implement a sentiment analysis feature: The project also aims to implement a sentiment analysis feature that will allow users to view feedback and comments from other users
To design a user-friendly interface: The project's interface should be intuitive and easy to use, making it accessible for all types of users.

2. Research

2.1. Backend Research

During the backend research phase, we explored different server-side technologies that could be used to build a scalable and robust backend for the ArtistConnect application. Our research included examining various databases, programming languages, and frameworks that could be used to develop the backend.
After considering several options, we decided to use Firebase as our backend server. Firebase is a cloud-based platform that provides developers with tools to develop, manage, and deploy mobile and web applications. Firebase offers several services, including firestore database, authentication, and storage, making it an ideal choice for building the ArtistConnect backend.
Firebase's real-time database provides a flexible and scalable backend solution that can handle the large amount of user-generated content expected on a social media application. The firestore database can also be integrated

with Firebase's authentication and storage services to ensure data privacy and security.

2.2. Frontend Research

During the frontend research phase, we explored various frontend technologies that could be used to build an intuitive and responsive user interface for the ArtistConnect application. Our research included examining different JavaScript frameworks, libraries, and UI toolkits that could be used to develop the frontend.
After evaluating several options, we decided to use Angular as our frontend framework. Angular is a robust and powerful JavaScript framework developed by Google that provides developers with a set of tools and features to build dynamic and responsive web applications.
Angular's component-based architecture provides a clean and organised way to build the ArtistConnect user interface. Angular also offers several features, including data binding, dependency injection, and built-in directives, making it easy to build and manage complex user interfaces.

2.3. API Research

During the API research phase, we explored several APIs that could be integrated into the ArtistConnect application to provide users with rich and meaningful data. Our research included examining different APIs from various providers that could be used to fetch data related to concerts, events, and music.
After evaluating several options, we decided to integrate the following APIs into the ArtistConnect application:

- **Spotify API**: This API provides access to the Spotify music catalogue, allowing users to search for and play songs, albums, and playlists.
- **TicketMaster API**: This API provides access to the TicketMaster event database, allowing users to search for and view upcoming concerts and events.
- **EventBrite API**: This API provides access to the EventBrite event database, allowing users to search for and view upcoming events and festivals.
- **Google Maps API:** This API provides data using longitude and latitude of events as well as the users location to determine how far away events are taking place from their location

Unfortunately, the SoundCloud API was deprecated and no longer available for integration into the ArtistConnect application.

2.4. Angular

Angular is a popular front-end framework developed and maintained by Google. It is an open-source framework that enables developers to build dynamic and scalable web applications. During our research phase, we explored the features and capabilities of Angular to determine whether it was the best fit for our project.
Our Angular research involved the following activities:

- **Learning the basics:** We started by learning the basics of Angular, including its architecture, components, and directives.
- **Exploring Angular features:** We explored Angular's features, including dependency injection, data binding, and routing.
- **Comparing with other frameworks:** We compared Angular with other popular front-end frameworks like React and Vue.js to understand their strengths and weaknesses.
- **Evaluating community support:** We evaluated the level of community support for Angular, including the availability of online resources and developer communities, since we had not previously used angular as part of our time in DCU.

2.5. Firebase

Firebase is a comprehensive mobile and web app development platform developed by Google. It offers a wide range of tools and services to make building applications easier and faster. During our research phase, we explored the features and capabilities of Firebase to determine whether it was the best fit for our project.
Our Firebase research involved the following activities:

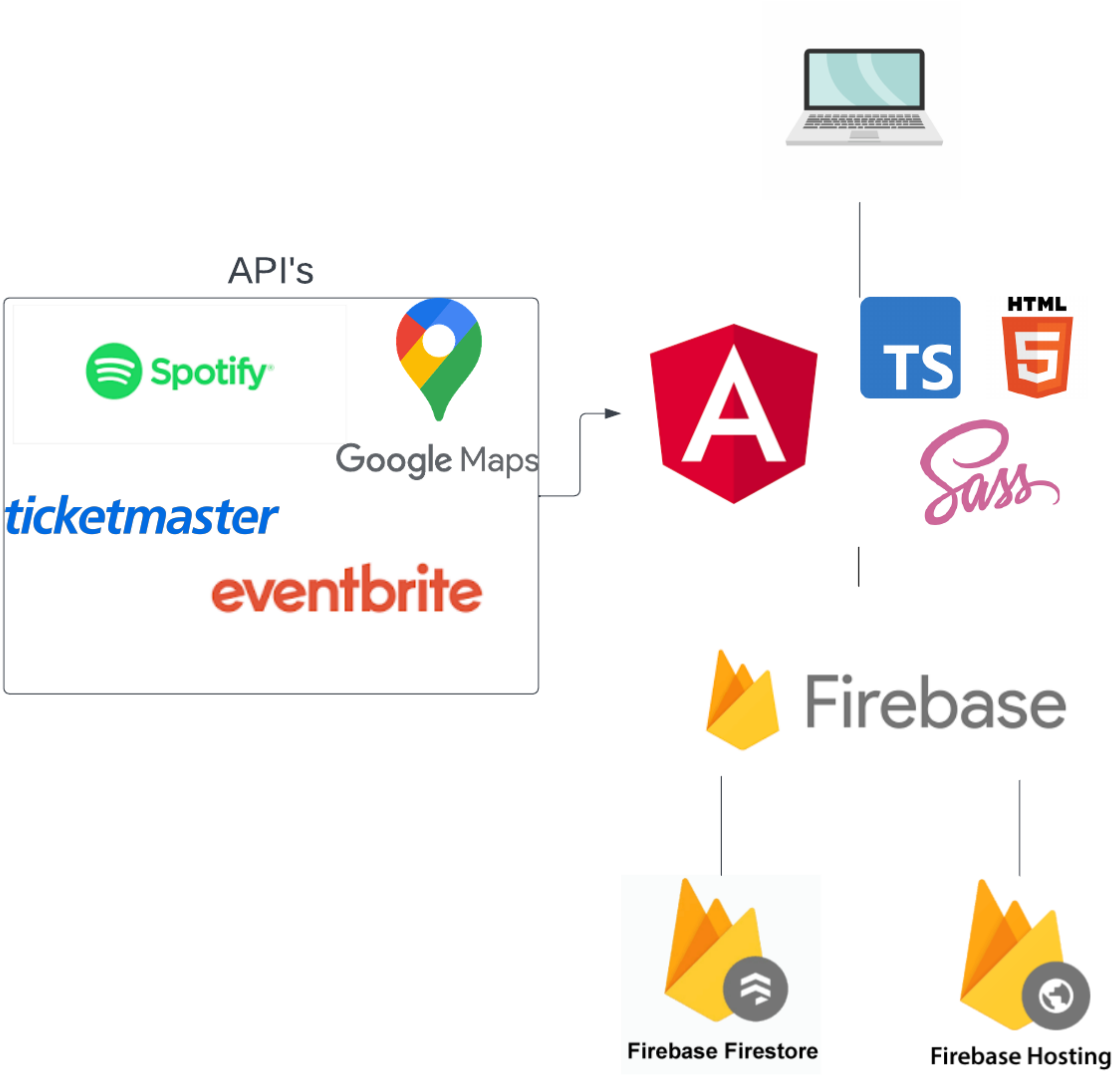
- **Exploring Firebase features**: We explored Firebase's features, including real-time database, cloud functions, and cloud messaging.
- **Comparing with other platforms**: We compared Firebase with other popular back-end platforms like AWS, MongoDB and Azure to understand their strengths and weaknesses.

As a result of our research, we concluded that Firebase was a suitable platform for our project due to its excellent features, easy integration, and wide community support.

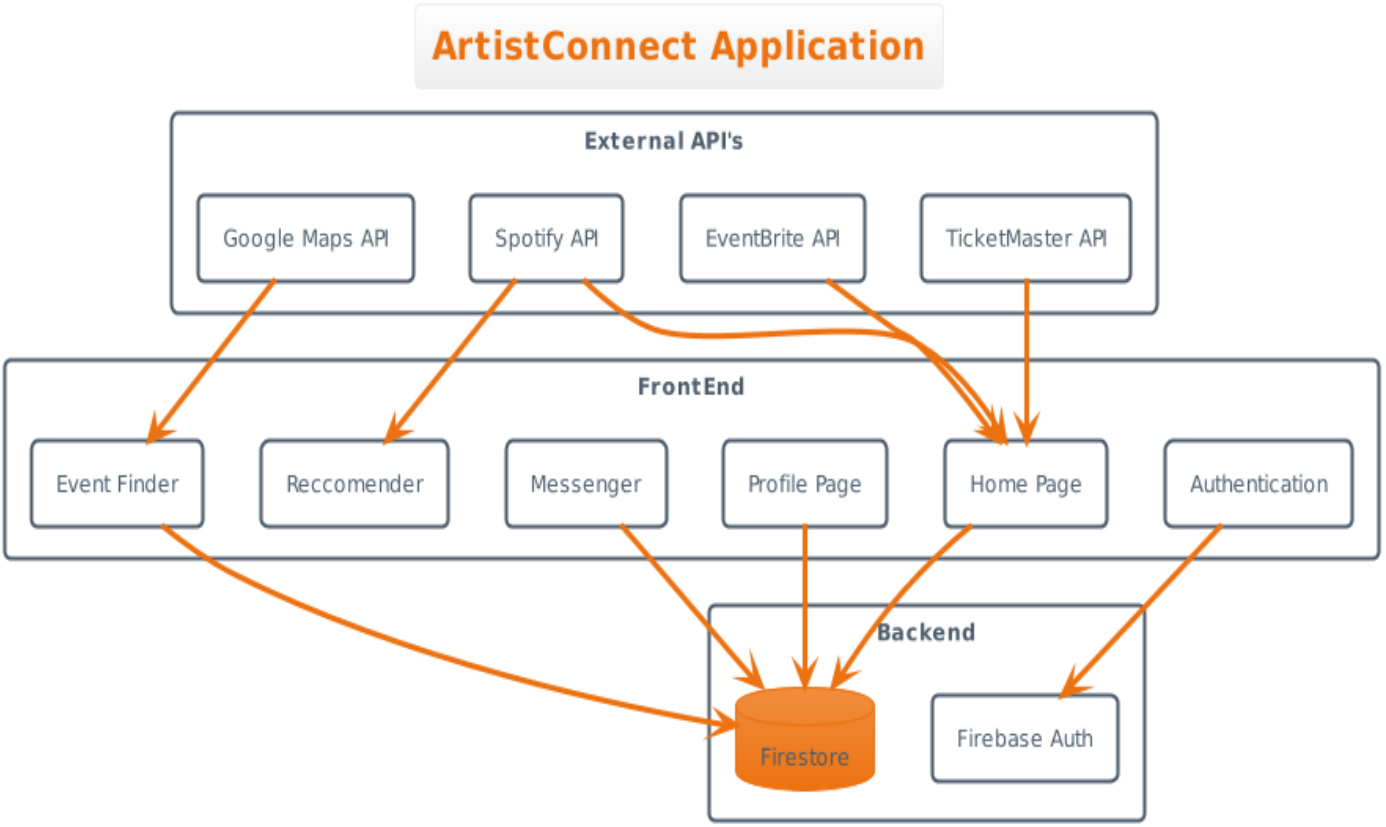
3. Design

3.1. System Architecture

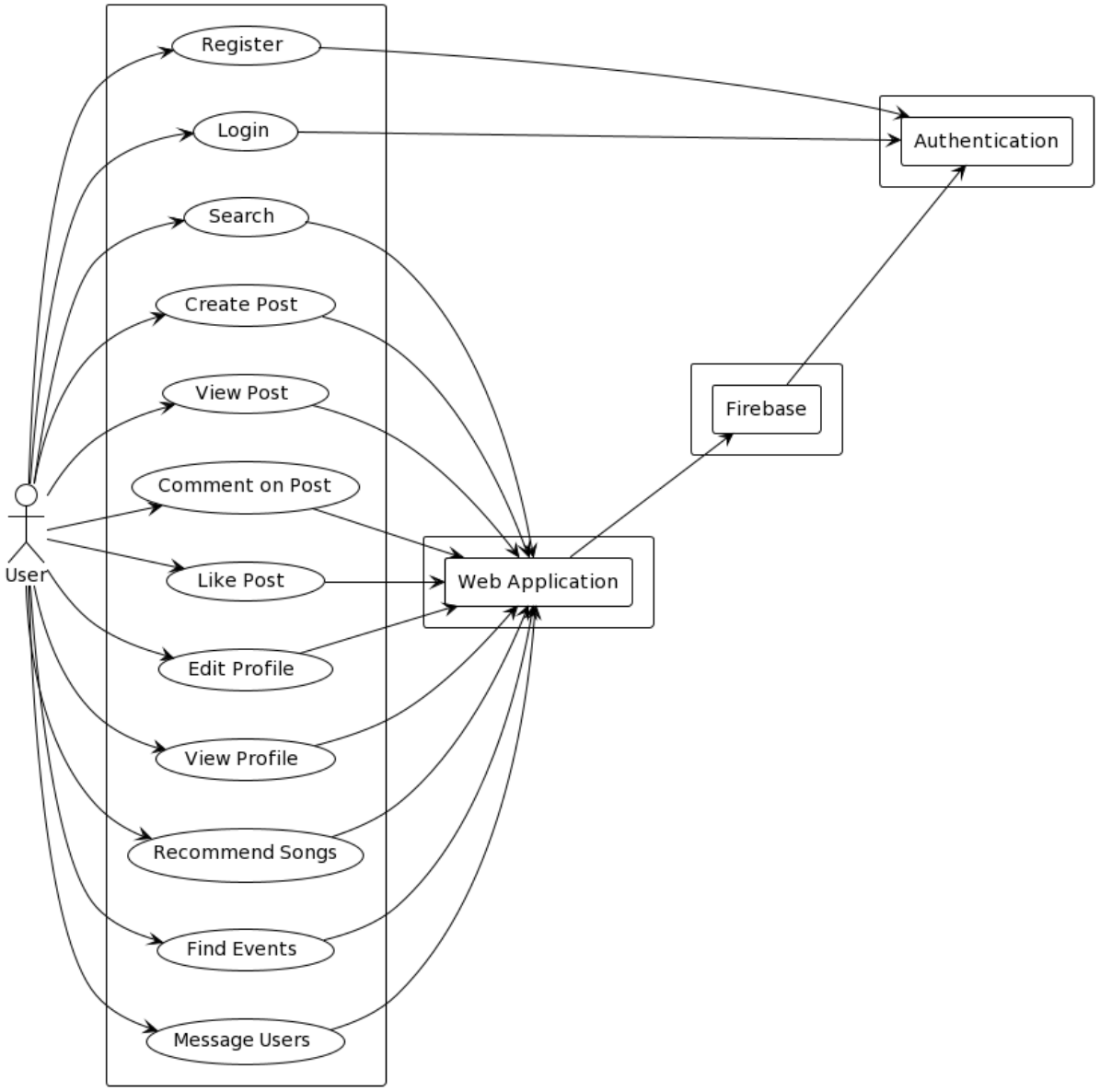
Our system architecture consists of 4 main components: The firebase backend, the angular frontend, the third party API's and the client side application. All information is pulled, posted to or edited from the Firebase database, and then either displayed on the application



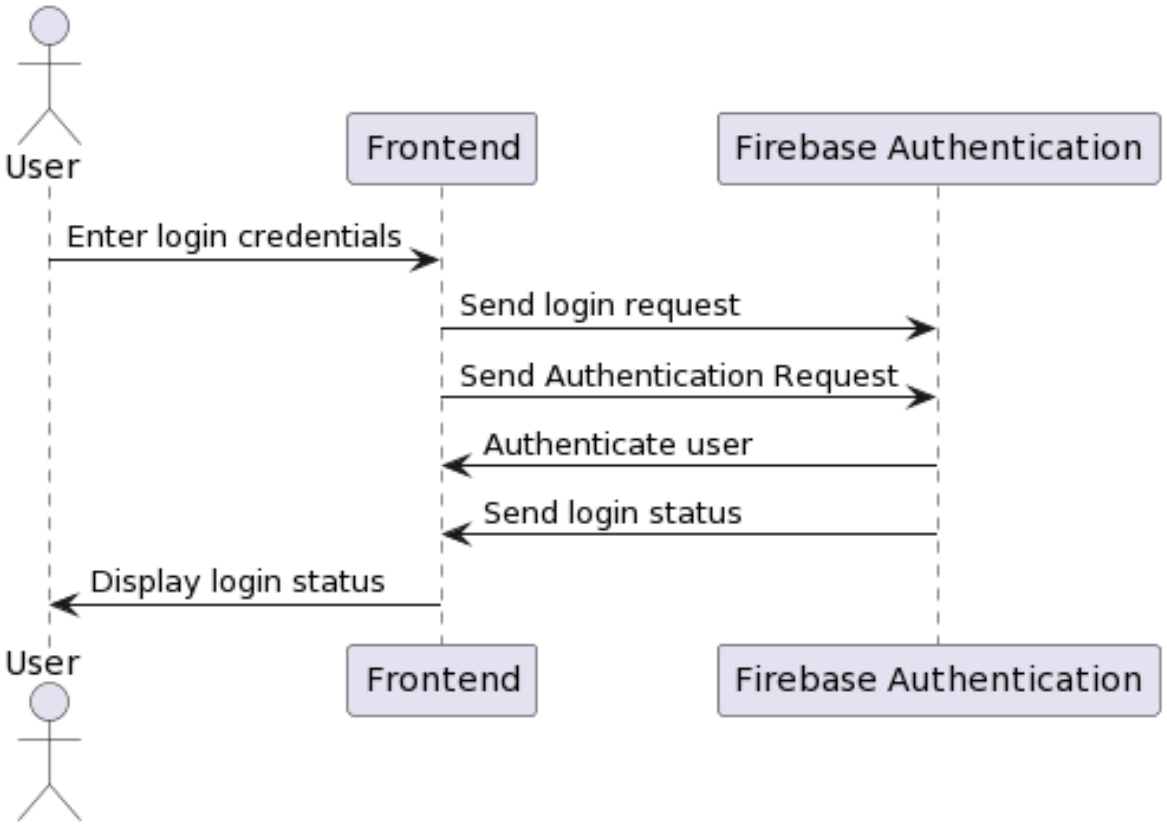
3.2. Context Diagram



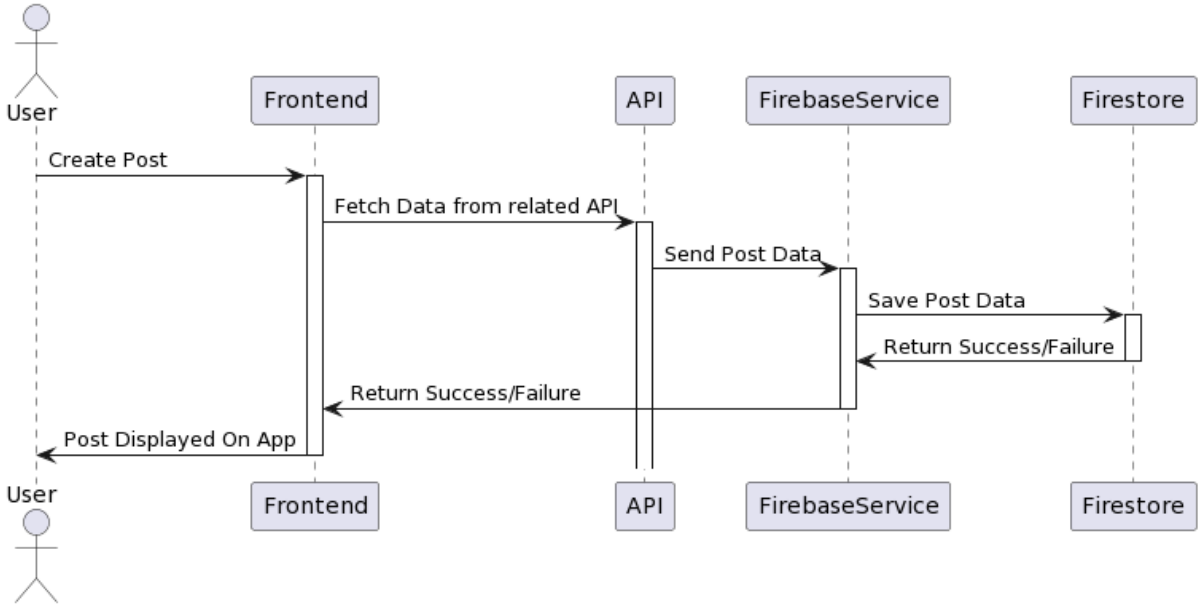
3.3.Use Case Diagram



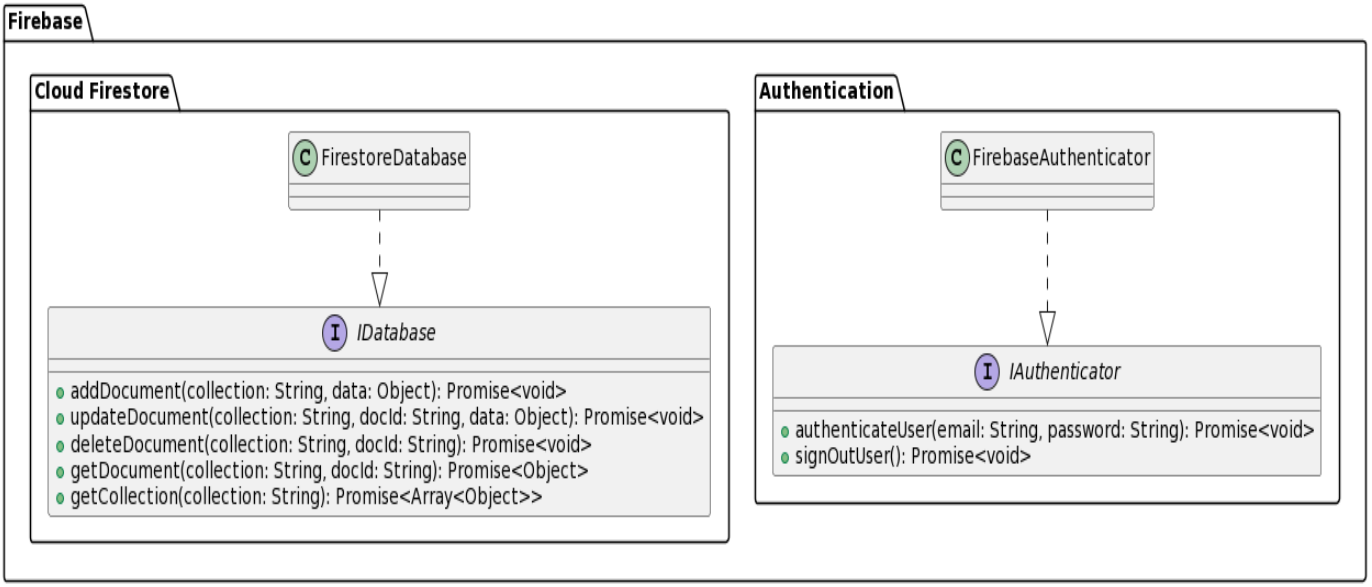
3.4. Sequence Diagram (User Login)



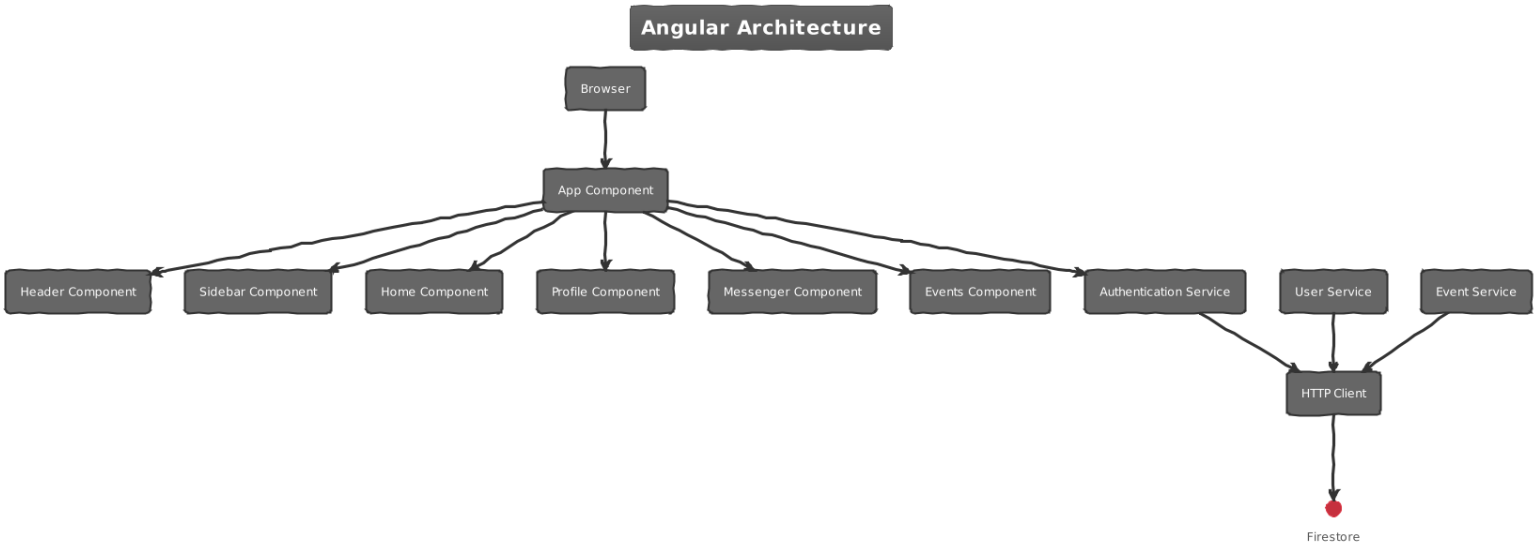
3.5. Sequence Diagram (User Makes a Post)



3.6. Firebase Architecture Diagram



3.7. Angular Architecture



4. Implementation

4.1. Backend Architecture

The backend architecture of ArtistConnect is based on Firebase, which provides a scalable and reliable cloud-based backend solution for mobile and web applications. Firebase is a real-time NoSQL database that allows for efficient data storage and retrieval. It also provides secure authentication, file storage, and cloud messaging services, which we integrated into our application.

Firebase Firestore Database: We decided to use firestore as our database. After doing some research on the different options available to us for database options, we thought firebase was the best choice for our application. Firestore stores data in documents, which are similar to JSON objects. Each document contains a set of key-value pairs, and can contain nested sub-collections.

Authentication: Firebase Authentication is used to authenticate and authorise users to access our application. We provide users with the option to sign up using their email address and password, or using their Google or Facebook account.

Database structure: In our database structure, we have several collections that store different types of data. We experimented with many different structures and document collection types for our application, as well as constant refactoring and updating as we went.

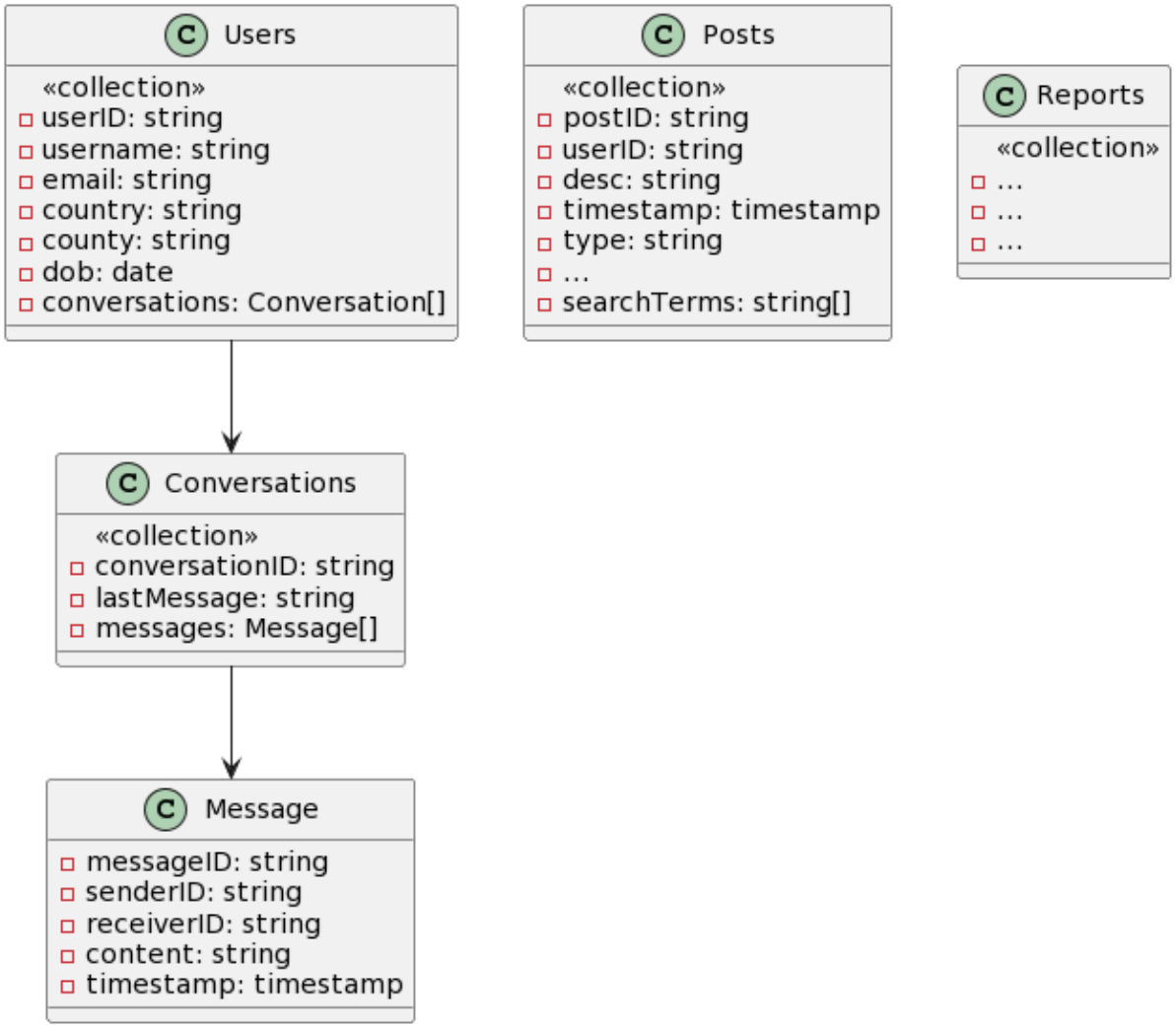
A high level description of the database collections and fields can be found below:

The "users" collection stores user account information, including user IDs, usernames, email addresses, and profile data like country, county and date-of-birth. It also contains a list of conversations the user is included in for the message centre.

The "posts" collection stores all the posts created by users, including the post content, timestamp, and user ID. The post content will differ depending on whether the post is a song or an event. All posts also have a “searchTerms” field which is a list containing words contained in the description, artist, song name etc. which is for search purposes.

The "reports" collection stores reports on posts that are submitted by users, including the report content, timestamp, and user ID.

The "conversations" collection stores all the messages sent between users in private conversations, including the message content, timestamp, and user IDs of the sender and receiver.



4.2. Frontend Architecture

Our frontend architecture follows a component-based design pattern, with each feature of the application encapsulated in a separate component. This allows for better code organisation and reusability, as well as easier debugging and testing.

The components we use in our application are:
Register: Allows users to create an account and sign up for the platform. Register features a HTML page where users can enter their own data into multiple input fields, and when they submit it will set them up with an account on firebase and redirect them to the homepage

```
register() {
   if (this.newUserData.email == '' || this.password == '' ||
this.newUserData.country == '' || this.newUserData.county == '' ||
this.newUserData.displayName == '') {
     alert("Please enter valid data into all fields to successfully complete
profile registration")
     return;
   }
   this.firebase.register(this.newUserData.email, this.password,
this.newUserData.displayName.trim().toLowerCase(), this.newUserData.dob,
this.newUserData.country, this.newUserData.county);
 }
```

Login: Allows registered users to log in to the platform. Users login with their email and password they used to register on the application. Firebase has their own functions to handle logins, logouts and registration, so the main aspects of these components are error handling to ensure there are no blank entries

```
async login() {
  if (this.email == '' || this.password == '') {
    alert("Please enter valid data into both fields")
    return;
  }
  await this.firebase.login(this.email, this.password, "");
  if (this.firebase.isLoggedIn) {
    this.isSignedIn = true;
  }
  this.email = '';
  this.password = '';
}
```

Home: Displays a feed of posts from users that the current user follows. The homepage is where the user is brought after a user logs in or registers. It displays the posts of users the logged in user follows, and will display all posts if the user does not follow anyone. Posts are retrieved from the firestore database using the getPosts() function. A filter option is present at the top of the homepage where users can choose between only songs, only events or all posts being present on the homepage.

```
getPosts(filter: string) {
  let id = this.firebase.userData.uid;
  this.firestore.collection('users', (ref) => ref.where('uid', '!=',
id)).valueChanges()
    .pipe(
      map((users) => users.map((user: any) => user.uid)),
      take(1)
    ).subscribe((uids) => {
      this.firestore.collection('posts', ref => ref
        .where('uid', 'in', uids)
        .where('type', '==', filter)
        .orderBy('timestamp', 'desc')
      )
      .snapshotChanges()
      .pipe(
       map(actions => actions.map(a => {
        const data = a.payload.doc.data() as PostData;
        const did = a.payload.doc.id;
        return { ...data, did };
      })),
      take(1)
    )
    .subscribe(postsData => {
      this.feedPosts = postsData;
    });
  });
}
```

Post: The post component contains functions for liking, deleting, commenting and general interactions with the post. The template for a post is also implemented here to reduce the size of the homepage component html file.

```
likePost(postId: string) {
  this.fauth.authState.subscribe(user => {
    if (user) {
      let likeRef =
this.firestore.collection(`posts/${postId}/likes`).doc(user.uid);
      likeRef.snapshotChanges().pipe(
        take(1),
        map(action => {
          if (action.payload.exists) {
            likeRef.delete();
          } else {
            likeRef.set({});
          }
        })
      ).subscribe();
      this.getLikeCount();
    }
  });
}
```

Create-post: Allows users to create a new post with content, tags and other relevant details. Opens up a dialog box where users can enter the details of the kind of post they want to make, either a song or an event. If the song is a song from Spotify, the user has the option to search from Spotify's collection of songs, which prevents the need to manually copy and paste the link of the song from Spotify. The post is posted to firebase and the information is grabbed from the relevant API whether it be Spotify,

TicketMaster or EventBrite and real information such as location, time, venue and link to purchase tickets are grabbed from the API.

```
 async onSpSearch() {
   this.spResults = [];
   const base64 = (value: string) => {
     return btoa(value);
   }
   //Generates OAuth token for api
   const auth =
base64('b6ccc6a683614eb49896a4fa30ed0815:a04caf9a809e49178a70755e84e29c4f');
   const response = await
axios.post("https://accounts.spotify.com/api/token",
"grant_type=client_credentials", {
     headers: {
       Authorization: `Basic ${auth}`,
       "Content-Type": "application/x-www-form-urlencoded",
     },
   });

   const formattedQuery = this.spSearch.replace(/\s/g, '%20');
   let authtoken = response.data.access_token;

this.http.get(`https://api.spotify.com/v1/search?q=${formattedQuery}&type=tra
ck&limit=10`, {
     headers: {
       Authorization: `Bearer ${authtoken}`
     }
   }).subscribe((data: any) => {
     const tracks = data.tracks.items;
     const songTitles = tracks.map((track: any) => track.name);
     this.spResults = [];
     for (const track of data.tracks.items) {
       const item: spotifySong = {
         id: track.id,
         name: track.name,
         artist: track.artists[0].name,
         url: track.external_urls.spotify,
         image: track.album.images[0].url,
       };
       this.spResults.push(item);
       console.log(item.image)
     }
   });
 }
```

Comment-view: Displays comments for a particular post. Uses a dialog which opens and shows all comments other users have made on a particular post. The logged in user can also post comments here.

```
addComment() {
  let ts = firebase.firestore.FieldValue.serverTimestamp();
  this.afAuth.authState.subscribe(user => {
    if (user) {
      this.fbase.getUser(user.uid).subscribe(name => {
        this.commenter = name;
        const comment: Comment = {
          userId: user.uid,
          name: this.commenter.displayName,
          timestamp: ts,
          input: this.commentInput
        };
        this.commentsCollection.add(comment);
        this.commentInput = '';
        this.comments = this.commentsCollection.valueChanges({ idField: 'id'
});
    });
    }
  });
}
```

I-player: A music player component that allows users to play songs on the platform. Users can click a button on any song post which will bring up an integrated IPlayer that gives users a taste of what the song sounds like.
Nav-bar: A navigation bar that allows users to navigate through different sections of the application. Allows users to easily visit some of the core areas of the application such as their own profile, home, the message centre, the event finder and the song recommender.
Header: A header component that displays the logo and other relevant information. Contains the search bar as well as a dropdown where users can view their own profile and logout of the application.

Search: Allows users to search for other users, posts, and events on the platform. Searches posts based on the searchTerms field which is an array of terms that are found in the post from the description, the song name, the song author and the displayName of the user who made the post. The functions are written in the firebase service.

```
getFilteredSearchResultsPost(searchValue: string): Observable<PostData[]> {
  return this.firestore.collection('posts', ref => ref
    .where('searchTerms', 'array-contains', searchValue.toLowerCase())
    .orderBy('desc')
  )
    .snapshotChanges()
    .pipe(
      map(actions => {
        return actions.map(a => {
          const data = a.payload.doc.data() as PostData;
          const did = a.payload.doc.id;
          return { ...data, did };
        });
      })
    );
}
```

Profile: Displays user's profile information and their posted content. Users can see the amount of posts, followers and how many users they follow here. They can also see their own posts.
Edit Profile: Allows users to edit their profile information. This is a dialog that opens and allows users to change their profile information like their location, date of birth and display name in case of an error when registering.

Song-Recommender: A recommender system that recommends songs to users based on a song that is entered by the user. The recommender uses Spotify's API to generate recommendations based on parameters such as energy, vocals and instrumental.

```
async onGenerateRecommendations() {
    const regex = /\/track\/(\w+)(?:\W|$)/;
    const match = regex.exec(this.trackUrl);
    if (match) {
      this.trackId = match[1]
    }
    const base64 = (value: string) => {
      return btoa(value);
    }
    //Generates OAuth token for api
    const auth =
base64('b6ccc6a683614eb49896a4fa30ed0815:a04caf9a809e49178a70755e8
4e29c4f');
    const response = await
axios.post("https://accounts.spotify.com/api/token",
"grant_type=client_credentials", {
      headers: {
        Authorization: `Basic ${auth}`,
        "Content-Type": "application/x-www-form-urlencoded",
      },
    });

    let authtoken = response.data.access_token;
    const url = 'https://api.spotify.com/v1/recommendations';
    const headers = {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${authtoken}`
    };
    const params = {
      seed_tracks: this.trackId,
      energy: this.energy,
      speechiness: this.speechiness,
      instrumentalness: this.instrumentalness
    };
    this.http.get<any>(url, { headers, params }).subscribe(data =>
{
      this.recommendedSongs = data.tracks;
    });
  }
```

Event-Finder: Allows users to search for and view upcoming music events in their area. The event finder uses the logged in users current location and finds event posts near to the users location.

```
  getDistance(evlat: string, evlong: string, userlat: number,
userlong: number): number {
    let eventlat = Number(evlat);
    let eventlong = Number(evlong);
    const R = 6371; // Earth's radius in kilometers
    const dLat = this.toRadians(eventlat - userlat);
    const dLon = this.toRadians(eventlong - userlong);
    const a =
      Math.sin(dLat / 2) * Math.sin(dLat / 2) +
      Math.cos(this.toRadians(userlat)) *
Math.cos(this.toRadians(eventlat)) *
      Math.sin(dLon / 2) * Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    const distance = R * c; // Distance in kilometers
    return distance
  }
```

Conversation: Displays a conversation between two users.
Message-Center: Shows the user's messages with other users, displays the last message sent between users and the name of the user the conversation exists.
Event-Map: Displays a map of music events near the user's location. Uses Google Maps API to display a map view of where an event is taking place.

```
  ngOnInit(): void {
    const venueValue = this.data.venue.replace(/\s/g,
"%20").replace(/&/g, "%26");
    const cityValue = this.data.city.replace(/\s/g, "%20");
    const locationTerms = `${venueValue}%20${cityValue}`;
    //Google Maps API Key: AIzaSyCkzjjQv5IUTC0yz1HTYDtP8KFvx2xuWwM
    let url =
`https://www.google.com/maps/embed/v1/place?q=${locationTerms}&key
=AIzaSyCkzjjQv5IUTC0yz1HTYDtP8KFvx2xuWwM`;
    this.link =
this.sanitizer.bypassSecurityTrustResourceUrl(url);
  }
```

Like View: A component that displays the users who have liked a particular post.

5. Problems Solved

5.1. Technical Challenges and Solutions

During the development of the ArtistConnect social media application, we encountered several technical challenges. One of the challenges we faced was integrating the various APIs we used, such as Spotify, TicketMaster, and EventBrite. The APIs had different data structures and authentication mechanisms, which made it difficult to fetch and display the data consistently in the application. To solve this we developed multiple error checks and try statements to ensure the data was loaded correctly and to ensure there were no errors in the loading of the data from the multiple API's

Another challenge we faced was optimising the application's performance. The application had several components and pages, each with its data sources and dependencies. This complexity made it challenging to load and render pages quickly. To solve this challenge, we implemented lazy loading, which loads the components and data sources only when they are required. We also implemented caching to reduce the number of network requests and improve the user experience.

5.2. Lessons Learned and Best Practices

During the development of the ArtistConnect project, we learned a number of valuable lessons and best practices that can be applied in future software development projects.

One important lesson we learned is the importance of planning and design before beginning implementation. By carefully considering our design and architecture, we were able to avoid many potential problems and ensure that our system was efficient, scalable, and easy to maintain.

Another best practice we discovered is the importance of using consistent coding conventions and following established industry standards. This makes the code easier to read and understand, which can save time and effort in the long run. We also learned the value of good documentation, which can be extremely helpful in ensuring that the project remains understandable and maintainable over time.

Finally, we discovered the importance of thorough testing and debugging throughout the development process. By thoroughly testing each component and ensuring that it functions correctly, we were able to identify and fix problems early on, which prevented them from becoming major issues later in the development process.

5.3. Bug Fixes and Troubleshooting

Issues with API integration: We experienced some challenges integrating some of the APIs we used, such as the TicketMaster and EventBrite APIs. We were able to fix this by carefully studying the documentation and seeking help from online communities.

Problems with data persistence: We noticed that some data was not persisting as expected in the Firebase database. We were able to identify the cause of this issue and resolved it by modifying our functions for posting and getting data and ensuring that everything was working as intended.

User authentication issues: We encountered issues with user authentication, especially during the login and signup process. We solved this by implementing a more robust authentication flow with the help of firebase and improving error handling.

UI inconsistencies: We discovered some inconsistencies in the user interface, such as incorrect font sizes, colour contrasts, and alignment issues. We were able to address these by conducting thorough testing and using appropriate tools to improve the design and layout of the application.

6. Results

6.1. What we achieved

Looking back at the past eight months of implementation, we have achieved most of what we wanted to when we were creating the functional specification. Our original idea was to create a social media application that allowed users to post songs and events they were interested in, as well as connect with other users, interact with posts and message other users as well as get personalised recommendations of songs based on their style. We planned to design a platform where music fans can connect and find like minded individuals, and users can also use our recommender system to find new songs based on their taste they may not have heard of. We believe we have achieved this with many different directions to head in the future if we decide to continue development on the application.

In a more technical aspect, we have grown our programming skills and used new technologies such as Angular and Firebase, as well as used different styles of programming especially using a component based framework such as Angular. Designing a social media application is a challenge no matter how

large or small, and with so many moving parts it is always certain to improve your ability as a programmer in every possible way.

6.2. Feedback and Reviews

For the ArtistConnect social media application, we gathered feedback and reviews from users through various channels such as surveys, user testing, and the advice of our supervisor. Overall, the feedback and reviews were positive, with users praising the ease of use and the unique features offered by the application.
Users particularly appreciated the song recommender and event finder functionalities, which allowed them to discover new music and events that they may not have otherwise found. The private messaging system was also praised for its reliability and ease of use.
However, some users did report minor bugs and glitches, such as duplication of posts and some responsiveness issues for users with smaller screens. We addressed these issues promptly and worked to improve the overall performance and user experience of the application.
Overall, the feedback and reviews from users were valuable in helping us to improve the application and provide a better user experience for all users.

7. Future Work

7.1. Feature Requests and Roadmap

Advanced search filters: Allow users to filter search results based on specific criteria such as genre, location, and popularity.
Enhanced messaging system: Enable users to send voice and video messages, add emoticons and stickers to messages, and access previous conversations easily.
Live streaming of concerts and events: Allow users to stream concerts and events live, and interact with other users in real-time during the event.
Integration with more APIs: Explore integration with more APIs such as YouTube, Instagram, and Facebook to allow users to share more content and information.
Personalised recommendations: Implement a recommender engine to provide personalised recommendations based on users' music taste, preferences, and behaviour.

Gamification: Add gamification features such as badges, rewards, and challenges to increase user engagement and loyalty.

7.2. Bug Fixes and Maintenance

In terms of future roadmap, it is important to allocate time for bug fixes and maintenance. This will ensure that the application is functioning as expected and is free of any critical bugs or issues. It is important to have a process in place for identifying and prioritising bugs, as well as a plan for addressing them in a timely manner.
In addition, it is important to schedule regular maintenance tasks such as database backups, security updates, and performance optimizations. This will help ensure the long-term stability and reliability of the application.

7.3. Technical Debt and Refactoring

The importance of addressing technical debt and refactoring in order to ensure the longevity and scalability of the application is critical when considering future work we could implement on the application. Even though it is a college project, it is crucial to consider the possibility of a growing user-base and the need to optimise the application for efficiency and speed.

One way to address technical debt would be to prioritise fixing any known bugs and inconsistencies in the codebase. This would require a thorough examination of the application's existing code and identifying any areas that could be improved upon. By doing this, we would be able to minimise the risk of unexpected issues arising in the future, and make it easier to maintain the codebase over time.

Another important aspect to consider would be refactoring the codebase to improve performance and scalability. As the user-base grows, the application would need to handle a larger amount of data and user requests, which could lead to slow loading times and decreased performance. Refactoring the codebase would involve restructuring and optimising the application's architecture and design to ensure it can handle the increased load in an efficient manner.

7.4. Platform and Device Support Expansion

Expanding platform and device support is another area for future development. Currently, our application only supports web-based platforms, but with the increasing demand for mobile application support, it would be beneficial to create a native mobile application to better serve our users. Additionally, expanding the platform support to include other web browsers such as Firefox, Safari, and Edge will increase the reach of our application to more users. Testing and optimising the application for different devices and screen sizes will also be crucial in ensuring a seamless user experience. With these expansions, we can ensure that our users have easy and convenient access to our application, regardless of their preferred platform or device.