Podify – An App to Always Remember your Podcasts.

By Mayank Agrawal

Undergraduate in Computer Science

101111218

Supervisor:

Dr. David McKenney

Abstract

Podify is a mobile application that is created to help individuals better remember information that they hear while listening to podcasts. Podcast listeners often face the harsh reality of struggling to remember the content that they have just spent the last hour listening to. Podify enables its users to automatically be notified when they begin listening to a podcast and gives them a sleek and effective platform to store notes on their thoughts regarding their favorite podcasts. Podify also enables its users to see what other users are listening to, and even see what others are thinking about certain podcasts. Podify uses Spotify as the podcast listening platform that the users must connect to give it access to the user's podcast listening information.

Acknowledgements

I would like to thank David McKenny for his support and guidance throughout my project. I can proudly say that he instilled a passion of web development and learning about new technologies through his teaching style for me.

Table of Contents

1.Introduction

1.1 Chapter Overview

This chapter serves as an introduction to Podify. Podify is a mobile app that allows users to connect their Spotify account to better enhance their experience in consuming podcasts. It uses Spotify's public application programming interface (API) to allow users to securely link their Spotify account. The users can take detailed notes regarding podcasts that they have listened and are even prompted with a push notification to begin taking notes once they start listening to their favourite podcasts. This section will also cover the benefits of note taking to consume information, the motivation behind the creation of this project, and the goals of the application.

1.2 Background

The podcast market has exploded over the last decade. There are over 5 million different podcasts to choose from with over 70 million across them [1]. As of 2023, there are an estimated 460 million global podcast listeners, and these numbers are only expected to increase as time goes on [1]. There are also podcasts in multiple different languages, genres, and cultures so is a podcast for every single different type of listener. Users that listen to podcasts have many different reasons for listening to podcasts ranging from comedy, autonomous sensory meridian response (ASMR), and education. Demographics of the United States show the three out of four adults in the United States listen to podcasts to learn something new [2]. The fact that an overwhelming majority of podcast listeners are looking to learn something new is very interesting since in the past much of learning was done through textbook reading and lectures. Additionally, many very successful people today create their own podcasts to share their own knowledge and experiences to others in an easier way than creating a textbook or autobiography. With the massive increase in consumer technology that is widely available in an individuals' house, they can easily get a basic podcast set up with a few hours and then they jus need to hit record to being their podcast.

Users have the choice between many different platforms that they can use to listen to their favourite podcasts. The largest podcast listening platforms are Spotify, Apple Podcasts, Google Podcasts, and Pandora. All of these platforms have been seeing increased growth in their users as the entire podcast industry continues to grow. Spotify is by far the most preferred

platform, and this can be seen in figure 1.2.1 which shows that 25% of the podcast listeners in the United States use Spotify [3]. Spotify additionally provides both premium and free services to users and thus can attract a larger user base in comparison to many of their competitors. With the free version of Spotify users do have limited access to the catalogue of music and do have to listen to advertisements when using the service. However, Spotify has been increasing their spending on getting exclusive podcasts on their platform which they make free to all user types.

Figure 1.2.1: Percentage of US Podcast Listeners by Platform [3]

1.3 Project Motivation

The main motivation for Podify comes from my own personal experience of being an avid podcast listener. Over the course of the last few years, I have been increasingly attracted towards podcasts as a medium to learn more about individuals' interpersonal experiences. Podcasts generally feel a little less formal than a lecture and allow the content to be easily digested as it is generally presented as a conversion between two or more people. However, when some users listen to podcasts, they can be distracted at certain points and forget some of the major points of the podcast, essentially wasting time by listening to the podcast. This section will serve as the rational for how users can better consume podcast information.

I personally found that when I listen to a podcast there is almost always one or two major anecdotes that really resonate with me, and these are often stories I like to tell others when speaking about the content of the podcast with others. There have been occasions where I try to reference the story and I really struggle to remember some of the details when put on the spot. Additionally, have even had experiences where I do not remember a single thing from an hour-long podcast. This started to become a pain point in my life and thus I wanted to create a solution for this.

Note taking is a very beneficial activity for both increasing memorization and is also a great way to apply techniques of active learning. The concept of actively learning is that the listener processes the information that they hear and then mentally filter out information that is not needed to get to the major take aways of the topic [4]. The act of taking a note adds emphasizes a thought that a person has. A study conducted at Stanford University found that humans have approximately 60,000 thoughts per day [5]. Thus, by physically taking a note of a single thought that a person has, the likelihood that person will remember that though significantly increases if they take a physical note of it. Furthermore, note taking helps remove a lot of the filler information from content by allowing the person to only write down pieces of information that they did not already know or information that they truly wanted to remember. If podcast listeners were able to have a platform where they could easily take and store their notes, I believe that it would be very beneficial for them.

Reminders can be very beneficial to help with memory and the learning of new ideas. Reminders serve as a physical queue to put thought into the subject of the reminder and thus can help an individual actively process their thoughts. Research conducted that the University College London suggests that getting reminders for certain activities can allow a person how better recall information regarding that specific reminder and will have an easier time remembering that information in the future [6]. In my university career I would always put in my due dates into my calendar with timed reminders that would notify me a few times before a deadline. This reminder would serve as a safety measure so that I would never miss a due date, and I can proudly say that I handed in all of my university assignments on time. Without the reminders I would have certainly missed some deadlines, and I feel that Podify should implement some type of notification system.

The main pain point of podcast listeners forgetting the content that they have just spent the last hour listening to is quite evident. Additionally, the fact that note taking, and reminders are beneficial activities for memorization and learning of content, it becomes imminent that a solution is needed to help address this problem. That is what Podify aims to do.

1.4 Goals

The main goal of Podify is to create a platform that can easily be used to digest podcast content and provide a pace for users to effectively take and share notes. This platform will be able to connect to the user's podcast service and automatically get the history of podcasts that the user has listened to. This platform will connect to a users Spotify account to get the podcast history, due to Spotify being the largest podcast platform. From the previous podcasts, the platform will allow users to take notes and rate how good the podcast episode was in their opinion. The platform will be a mobile app, since a majority of podcast listeners, use their smartphone to listen to podcasts [7]. With the creation of a mobile app, it is valuable to have a notification system that can remind the user to start to take notes of a podcast that they are currently listening to. Additionally, humans learn and achieve more by working together and thus the platform will allow users to share their notes and ratings with others if they choose to make their note public. The platform will also act as a place for users to view other podcasts that they are not listening to, and thus allow them to expand their horizons to learn new things.

1.5 Chapter Summary

In Summary, Podify looks to be a mobile platform for users to use to help remember the information that they have learned in a podcast. After considering the background information, motivation, and goals for Podfiy, the software development can be incorporated. Chapter 2 will serve as the section that document and discuss key aspects of the software implementation of Podify.

2.Project Design

2.1 Chapter Overview

Podify is a software application that uses many different technologies and has unique implementation decisions that can be discussed. The decisions around programing technologies, frameworks, and algorithms were made to best serve the motivations and backgrounds depicted in Chapter 1. This chapter aims to discuss and justify the decisions and provide a documentation for this project. Section 2.2 covers why Podify was created as a mobile application rather than a standard web application. Section 2.3 explains the server sided technologies that were used for the implementation and discusses the responsibilities that it has. Section 2.4 discusses the database technologies and data is structured within the application. Section 2.5 justifies the client sided technologies and explains some of the major features created with it. Section 2.6 covers the connection and use of Spotify's public API. Section 2.7 provides a deeper look at some of features and general architecture of the project.

2.2 Motivation for Creating a Mobile App

The main motivation to create Podify as a mobile app is convenience for the user, and integration with the primary device that users listen to podcasts with. 73% of podcast listeners listen to podcasts on their mobile device [7]. Due to this overwhelming percentage of podcasts listeners using their mobile device or smartphone to listen to podcast it would not seem very beneficial to create a standard web application as mobile users prefer to use apps rather than websites on their smartphones.

Research into user experiences comparing web application and mobile applications shows that users prefer to use the mobile applications more [8]. Mobile applications primarily provide convenience for the user as the application can be store on the user's home screen and do not require to have the user login multiple times as they would have to with a web application. Additionally, the general user experience has been documented to be better on mobile apps due to the integration into the hardware of the device, and also allowing mobile application to access device features such as push notifications or the camera. Lastly, users find that mobile applications run faster compared to web applications generally due to the integration of the navigation into the mobile application.

Although some users do say that the amount of mobiles application that they have on their devices is quite large and sometimes they stray away from downloading new mobile applications. The benefits of having a mobile application for Podify quite significantly outweigh the negatives. Thus, Podify has been created using a Cross-Platform Mobile Development tool called Expo, which will be further explained in Section 2.5.

2.3 Server-Side Technology

Firebase cloud functions were chosen as the server-side technology to prove data and functions to the client. Firebase cloud functions are part of Google's firebase which is backend as a service (Baas) [9]. Firebase cloud functions provide a suite of tools that simplify the backend development process to get the backend of an application running quickly and effectively. The firebase functions are written using JavaScript with a few syntaxes and configurations so that they can be integrated into the Firebase suite of technologies.

The advantages of using firebase functions is that they seamlessly integrate with the rest of the firebase suite of products which allows for easy integration of authentication, cloud storage, hosting, analytics, scalability. Firstly, authentication is often a pain point for web and mobile application developers. Firebase's authentication provides an end-to-end identity solution that allows for the handling of login process and account creating [10]. Additionally, Firebase authentication comes with comprehensive security as it uses the same safety measures used by Google themselves to provide developers with the same functionality [10]. Firebase hosting allows for the cloud functions that are created to be automatically hosted so that the backend endpoints are accessible for any authenticated clients. Firebase also provides seamless integration between their cloud functions and their NoSQL database. The details surrounding the database implementation will be covered in section 2.4. Lastly, the biggest reason to use firebase functions as the backend service is because of its autoscaling and serverless architecture. Firebase can scale the backend based on the number of users and demand without having to manually adjust anything from a developer perspective. The serverless architecture allows the developer to focus on developing the application rather than spend time on infrastructure management which is often a large time soak for backend development.

Alternative backend technologies that could have been used are AWS and NodeJS. NodeJS could have been used as the backend for Podify, as it has all the capabilities to create an

extensive backend. However, Firebase provides much more built-in features such as authentication connection which really speeds up the backend development process. Additionally, the fact that Firebase also has the database built into its backend allows for simple connection and create, read, update, and delete (CRUD) on the backend. AWS has a very similar set of products compared to Firebase that it offers and was considered as an alternative for a backend solution. However, after experimenting with AWS there was a steeper learning curve to use the platform compared to Firebase and the ease of use of Firebase in connection to the faster development ultimately led to the choice of Firebase as the backend technology.

Firebase uses Node and its own set of command line tools called Firebase CLI allow for the development and deployment of the backend functions. The firebase functions were API endpoints that allowed the client to interact with the backend and the database. A majority of the API's were either get or post requests from the client. Additionally, firebase allows for scheduled functions to run, which was needed to poll for the user's podcast data. This will be further explained in section 3. The cloud functions were written and test locally on the built in Firebase emulator, and then were easily deployed to the Firebase project created for Podify. Once the functions worked locally it only took about a minute to deploy the functions to firebase and this was a real advantage so that a local server did not have to be running to host the backend while in development. Furthermore, Firebase allows for standard JavaScript to be used to write the functionalities and thus a majority of the backend task were easy to implement, without having to gain new knowledge on the backend technologies.

2.4 Database Technology

For the Database technology, Firebase provides two options in Cloud Firestore and a Realtime Database. The Realtime Database is the older method of storing data on Firebase and stores data in the form of JSON, which at scale starts to become slower and harder to manage. The Cloud Firestore is the newer and recommended database provided by Firebase. Cloud Firestore stores data as document and these documents are stored as collections exactly the same as they are done in other common NoSQL databases such as MongoDB.

The biggest advantage of using Firestore is the integration with the rest of the google suite products and the seamless connection with the backend of Podify. The Firestore Database can easily be changed during the development and does not require a robust schema compared to

a SQL database. Since this application was created by a single developer with speed of development as a priority, the NoSQL Firestore was a great option to quickly get user data stored and connected to the client and backend technologies. Firebase provides simple retrieval options for data and has many examples that are easy to follow to interact with the database. Additionally, the data is nicely displayed on the Firebase console, so testing is made easy by allowing the developer to see the changes to the database in real time.

2.5 Client-Side Technology

Many mobile applications are created using individual programming languages depending on the device that the application is run on. For IOS applications on Apple devices Swift is the primary programing language that is used. For Android applications Kotlin or Java are the primary programming languages that are used. However, this divide of programming languages is quite intensive to create a mobile application for small teams and even more difficult for an individual. Thus, some mobile applications are being created using JavaScript and other web technologies through cross-platform mobile application frameworks such as React Native. The cross-platform mobile application frameworks allow the developer to create a single web application that can be emulated on a user's device regardless of the brand of the device. React Native is used by many Fortune 500 companies and even startups due to ease that comes with just creating a single application that can run across many different platforms [11]. Some of the most well-known applications that use React Native are Facebook, Microsoft, Shopify, and Wix.

Podify is built using React Native on the Expo platform. React Native allows developers to create mobile applications using React, which is a very popular JavaScript Library used to create user interfaces [12]. As mentioned above React Native has many existing libraries that developers can use which allows for faster development, and the fact that a single codebase can support both IOS and Android Applications is a real benefit.

Expo is an open-source platform that is used for developing mobile applications with React Native [13]. Expo provides tools and services that handle the deployment process of an application allowing developers to focus on the creating of their apps rather than the extensive setup of the configurations and infrastructure [13]. Expo also allows developers to test their mobile applications on their own devices without the need of an emulator to test their code.

Generally, emulators are required when developing mobile applications on the developer's computer, and for IOS applications the developer must own an Apple computer to get said emulator. Expo also provides many libraries for developers so that they can use pre-existing components rather than having to create everything from scratch [13].

For Podify almost all the components used were from the React Native library but had to be styled to fit the application based on size, color, and look of component. The ease of integrating React Native components really sped up the development process on the client side by allow the focus to be on the visual display of the components rather than managing the different elements on the screen. The individual screens for Podify were all created using the React Native components along with the create of some custom components which were primarily created to reduce on duplicate code. In terms of styling the screens that was done manually by creating stylesheets on each screen which would be applied to the components similar to how it is done in a basic HTML application. React also allows developers to send specific data to components so that variables can be sent to the components and fill in some of the components with user data retrieved from the database. In react this data is sent through the use of Props, which allows components to pass data to one another.

Navigation for the application was controlled using a top level React component called Navigation Container and a Stack.Navigator which would control which screens would be visited when certain actions were taken. A majority of the overhead with implementing the navigation is quite straightforward with the React Native Components, however, the client side does start to scale quite quickly in size making it difficult to find bugs when they appear.

The mobile application did need to make a connecting to the backend Firebase instance to access the endpoints created. The clients to have the required credentials for the Firebase application in order to make API calls and connect to the services to ensure security. Firebase provides great documentation for this and really makes the process easy, although versioning for the libraries is an important aspect to keep track of.

2.6 Spotify API

A key component for Podify is the connection of a users Spotify account to the platform. Without this Podify would not exist. Spotify has an excellent developer documentation to access their public API's and even have some examples of code that show their user authentication process.

The first step to connecting the Spotify API is to create an application on the Spotify developer console to allow access for Podify to get API authorization from Spotify [14]. In order to access Spotify data on a specific user, that user needs to authorize Podify to access their data. Spotify implements OAuth 2.0 authorization framework to allow a user or application to access Spotify data [14].

To make any API call to Spotify, a client key is required to connect to the app created on Spotify's developer dashboard. Once that is obtained Spotify has a web module that allows users to grant permissions for the desired scopes of Podify. During the permissions step the user will need to sign in to authorize access to their data. After the user successfully grants the permissions, Podify is returned an authorization token that is used to request the access and refresh tokens. The access and refresh tokens are used to make any subsequent requests on the users' behalf. Once that is done Podify has done all of its initial setup required for the access to the users Spotify information, and then can just simply make API calls to get that users information. See figure 2.6.1 to see a visual diagram on the authorization flow provided directly from Spotify's documentation.

Once the user's access token is obtained by Podify, it can poll to see if the user is currently listening to anything on Spotify, and if the user is listening to a podcast, then Podify can invoke its backend process to notify the user and create a note for that specific podcast episode. Spotify has a thorough documentation outlining how to make API calls for specific information, which can all be found in their developer documentation.

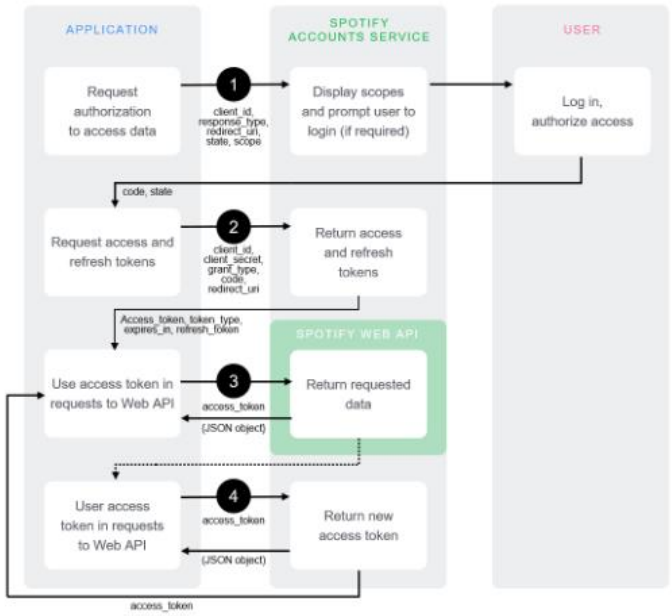

Figure 2.6.1: Spotify Authorization Code Flow

2.7 Architecture Strategies

Now that all the major technologies used for Podify have been covered, this section will cover the architecture of the overall program. The core functions of the application revolve around user authentication, polling for users' podcasts, notifying the user to take a note, allowing the user to take a note, and the community tab of the application. Additionally, connection of the users Spotify account to Podify a major feature, as discussed in section 2.6.

User authentication and user creation is handled by the built in authorization module provided by Firebase. When the mobile application for Podify is launched it checks to see if someone is logged in using a stored state variable. If they are not logged in, then they are served the login page where a user can enter their email and password. If a user attempts to login there is a function called "signInWithEmailAndPassword" from the authorization module from firebase and when that returns an authenticated user is set as a variable within the application, and then the logged in user is displayed the rest of the application. If the user is looking to signup there is a button on the login page where the user is directed to a screen to signup and the signup process is handled by a firebase authorization module function called "createUserWithEmailAndPassword".

Spotify does not have a web hook that would fire when a user starts listening to a podcast, and thus Podify needs to poll the Spotify API on a scheduled job in order to check to see if a user has begun listening to a podcast and then proceed from there. Ideally if there was a web hook then it would be the best method to get the listening data for a user, however Firebase has functionality to have scheduled functions and thus the polling for user data is run every minute, and the returned data is checked, and the appropriate next steps are taken once the results have been achieved. This was a major part of the project and will be further explained in section 3.

Notifying the user to start taking notes of a podcast is triggered when the polling function find that a user is currently listening to a podcast. When the user signs up for Podify, they are generated a push notification token linked to their device and this is stored as part of the user profile in the Firestore database. When it is time to send a notification Expo provides an API interface that requires the push notification token generated for the user and a message, and then sends a notification to that user's device with the given message. The fact that Expo provides an interface to send these notifications makes it a great choice as the client for Podify.

Allowing the user to take notes is handled by first getting users data from the database and backend. The user can then see what is currently stored in their note for a specific podcast and allows the user to edit the note though the use of a text input component in React Native. Once the user leaves the note then the data from the note is sent to the backend to save as an updated version of the users note. The community tab works similarly to how the users edit their notes however it is mainly just to view and not much is edited by a user on the community tab.

Those are the major structures of the application, and a more in-depth coverage of the applications flow will be covered in section 4 which covers the results.

2.8 Chapter Summary

This chapter covered much of the rational behind the use of the specific technologies to create Podify, from both a client and server perspective. Additionally, this chapter explained how Podify connects to the Spotify Public API and the general architecture breakdown of the application. Chapter 3 will discuss the major challenges that were faced during the development of Podify and break down, in depth, some of the features that needed to be created for Podify to work.

3.Challenges With Implementation of Project Design

3.1 Chapter Overview

All projects have unforeseen challenges occurring that were not accounted for in the planning stages. Podify is no different in this sense. The main challenges surrounded the configuration of Expo and some of the libraries that are use with the client side of the application, the design of the polling functionality from Spotify's API, working with React Native components, and designing the NoSQL Database.

3.2 Configuration Challenges

The Expo SDK is what allows developers to link their React Native application and provides all the tools necessary to get a mobile application running. When testing the viability of Expo, the version of the SDK that was used was Expo SDK 45.0.0 and the demo application compiled and ran properly. To run the application on a user's device they must download the Expo-Go app which servers the Podify application to the user. When the actual development for Podify began, a new requirement on the Expo-Go application said that the minimum version of the Expo SDK that must be used is Expo SDK 46.0.0. After the SDK version was updated, some small changes were required to be made in the configuration files for the Expo project, which were not documented on the Expo documentation guide. The issue was resolved by searching through various posts on Stack Overflow and GitHub Forums.

When setting up the connection of the client portion of Podify to the Firebase backend there was another issue with versioning. The newest version of Firebase that gets installed when using NPM is 9.19.1, which has implemented some changes on how the connection works and the structure of the Firebase modules. The Expo tutorial outlining the connection process to Firebase is around two years old and therefore uses an older version of the Firebase modules. The tutorial did not outline what exact version they were using, and this research was required to find what version was needed. The error messages when attempting to connect to Firebase were not helpful as they would just respond with a generic message stating that there was an error connecting to Firebase. This process took a substantial amount of time as there was little

information regarding this issue. Eventually though trial and error this issue was able to be resolved.

3.3 Polling for Podcasts

As previously mentioned, polling the Spotify API to check if a user is listening to a podcast is the biggest component of Podify. Spotify does not have a web hook that pushes data out to an endpoint when certain conditions are met. Thus, Podify was required to constantly poll Spotify's API to check for new data.

When the polling function is trigged, it starts by looping through each of the users that exist for Podify. The function then ensures that the user has set up their Spotify account so that the Spotify Public API can be used to get their listening data. If the user has a Spotify access token that can be used to query for their Spotify Data, then Podify must check if the access token is still valid. OAuth 2.0 sets a standard that access tokens must expire after a certain period of time, and for Spotify this time limit is one hour. Thus, when Podify initially gets the users access token it also stores an expiry time for the token which is 59 minutes in the future. If the token is expired, then another API call must be made using the users refresh token that was stored when the user initially signed up.

The actual API call to Spotify is made to the "https://api.spotify.com/v1/me/player?additional_types=episode" end point. This endpoint returns the podcast the user is currently listening to, and if they are not listening to anything then an empty body is returned. If the user is not listening to a podcast, then the polling function moves onto the next user. If the user is listening, then a default note is created for the user just a placeholder so they will be able to take notes in the application about the returned podcast. After the default note is created for the user, there is a function in the backend which fires the push notification to the Expo-Go application which notifies the user if they would like to start taking a note. Additionally, Podify stores all the unique podcasts that have been listened to so that storing reviews of certain podcasts is made simpler.

The polling function is set up as a scheduled function in Firebase. The scheduled function runs every minute to ensure that notifications to the user are not too delayed and so that Podify is as up to date as it can be. Podcasts are long forms of media and thus when a user is going to

listen to a podcast, they are going to be listening to it for upwards of 20 minutes. This ensures that Podify will not miss the podcast as it will have received at least 20 responses from the Spotify API that a user is listening to a specific podcast. To ensure that the user is not bombarded with notifications every minute the notifications only send to the user if it has been over 10 hours since the last time they listened to that specific podcast. To allow for Scheduled cloud functions to be hosted on Firebase, an upgrade to the Blaze pricing model was needed [15]. Fortunately, Google allows up to three scheduled function free of charge and thus running the polling function does not cost any money at a small scale.

3.4 React Native Challenges

React provides great functionality support through the use of React hooks. Hooks allow the use of stateful logic to be implemented across multiple react components without the need for class components in React. The useState hook is one of the most commonly used hook at it is used to manage state and when the state variable is set it triggers the component to be re-rendered. Generally, when loading variables into a component in React, a developer uses the useState hook to set the value of the variable after making any appropriate API calls. In the development of Podify, when a user is done editing their note the new data needs to be saved and sent to the backend. React proves a function that is called beforeRemove that basically allows certain code to run before the React component is removed from the DOM. When attempting to access the state variables in this stage, it appeared that sometimes the data was being set to the default values rather than the updated and changed values. After reading into the documentation, the useState hook being used with the beforeRemove function can lead the re-renders taking place because the component is still on the DOM, and thus resetting the value within the useState hook [16]. The useRef hook is better to keep track of the variables set by the user that need to be sent to the backend. The useRef hook is used to create a mutable reference that can be updated without triggering components to re-render [16]. After the useRef hook was used to store the user data that needed to be sent to the backend, the components were not being unnecessarily re-rendered and the functionality worked as it should have.

In React Native applications it is standard to separate the navigation components from the actual screen components that are seen by the user. When initially setting up the navigation portion of the application it can be easy to get confused with how the layering needs to be set up.

In React Native the common method to handle the navigation is to use a navigation stack with essentially "stacks" new screens onto the previous ones and then allows the user to go back following the same path of screens. Once the developer can understand this flow, they can use the navigation prop to navigate to the desired screen in the navigation stack.

3.5 Database Challenges

The main challenge with the database was to create models that both made sense and were not redundant. The power of NoSQL database creation is that it allows for collections to be directly created within another collection, rather than having to create a new separate collection that would have to be linked by id. For Podify the two main collections are Podcasts and Users.

The User collection has many user documents that are identified by the user's unique user id which is provided by the authorization module of Firebase. Each user document then has specific user information such as the users Spotify API access information and the expiry of their access tokens. Additionally, the user doc also has a collection of notes that allows for the storage of all of the notes that a specific user has created. The initial rational to create user's notes as a collection within the user doc was because it would result in fewer queries needed to get any information for a specific user and this could be served faster. Within the notes collection for the user information regarding some of the main details of the podcast and any of the users note details are also stored.

The Podcast collection has many podcast documents which store all the information about a Podcast that is received from Spotify when making a call to the Spotify API. This collection serves as a local database of podcasts that can grow as users use Podify and then in the future would be used to create a recommender system. Each of the podcast documents also has a reviews collection that is used to store user reviews of a specific podcast. This collection just keeps track of the user id of the review and their score for the podcast. The reviews collection is created to ensure that user reviews are linked to the podcast so that when calculating the average score of a podcast this can be done more efficiently with less queries needed.

Working with a NoSQL database is generally quite straightforward in terms of making queries as the backend is often given more data than it needs, which can easily be filtered out.

3.6 Chapter Summary

The major challenges for Podify have been discussed which stemmed from configuration, polling for Spotify Data, database design, React Native setup. The next chapter will cover what was implemented in creation of Podify and will show images of the screens that can be seen by users when using the application.

4.Project Results

4.1 Chapter Overview

This chapter will serve as a walkthrough of the features and flow of the Podify application. Section 4.2 will discuss the flow and walkthrough of the application from a user perspective. Section 4.3 will discuss application testing, and how that was done.

4.2 Flow and Walkthrough

This section will be further broken down into smaller sections so that specific functionality can be isolated for a better understanding of the application.

4.2.1 Registration and Logging In

The first step for any mobile application is to authenticate the user and allow them access to the rest of the application. Figure 4.2.1.1 shows the basic login screen where a user can enter their email and password. If the user enters the correct credentials, then they are directed to the home page for a user.

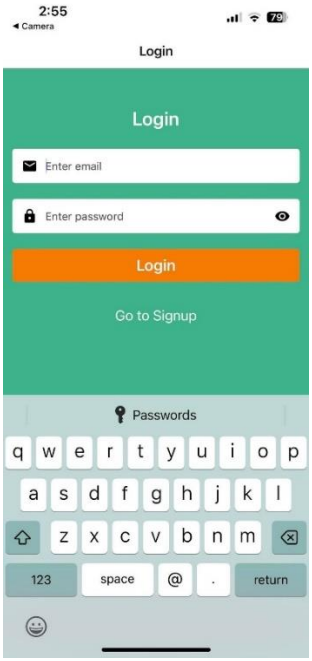

Figure 4.2.1.1: Basic Login Page for Existing Users.

If the user does not have an account, they can select the signup button and then are directed to the signup page seen in Figure 4.2.1.2. This looks almost identical to the login process to the user, however, on the backend these two functionalities are required to be handled differently. Once the user enters a valid email and password they are then directed to the main page of the application and will automatically remain signed in until the explicitly click on the sign out button.

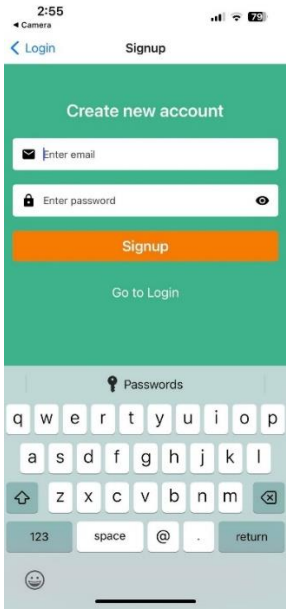

Figure 4.2.1.1: Basic Sign up for New Users.

4.2.2 Connecting Spotify

When a user first logs in there will be no podcasts listed for them since, Podify only adds support to take notes on new podcasts that the user listens to. Figure 4.2.2.1 shows the home page with a message telling the user to start listening to podcasts.

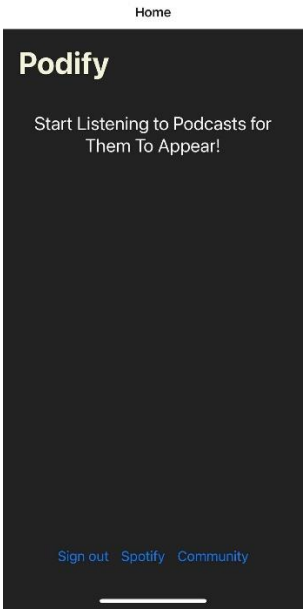

Figure 4.2.2.1: Podify home Screen with no podcasts.

To ensure that the user has the Spotify account linked the user must click on the Spotify button at the bottom of the home page. Once they click this the user is displayed one of two options. First if the user has not linked their Spotify account, then they are shown a button stating, “Connect Spotify Account”. Then they are shown a web browser allowing the user to login to their Spotify Account giving Podify access to their Spotify Data. See Figure 4.2.2.2. Once this happens the user is directed back to the home page shown in Figure 4.2.2.1. If the user has already connected their Spotify account, they are shown a message stating that Spotify has already been connected. See Figure 4.2.2.3.

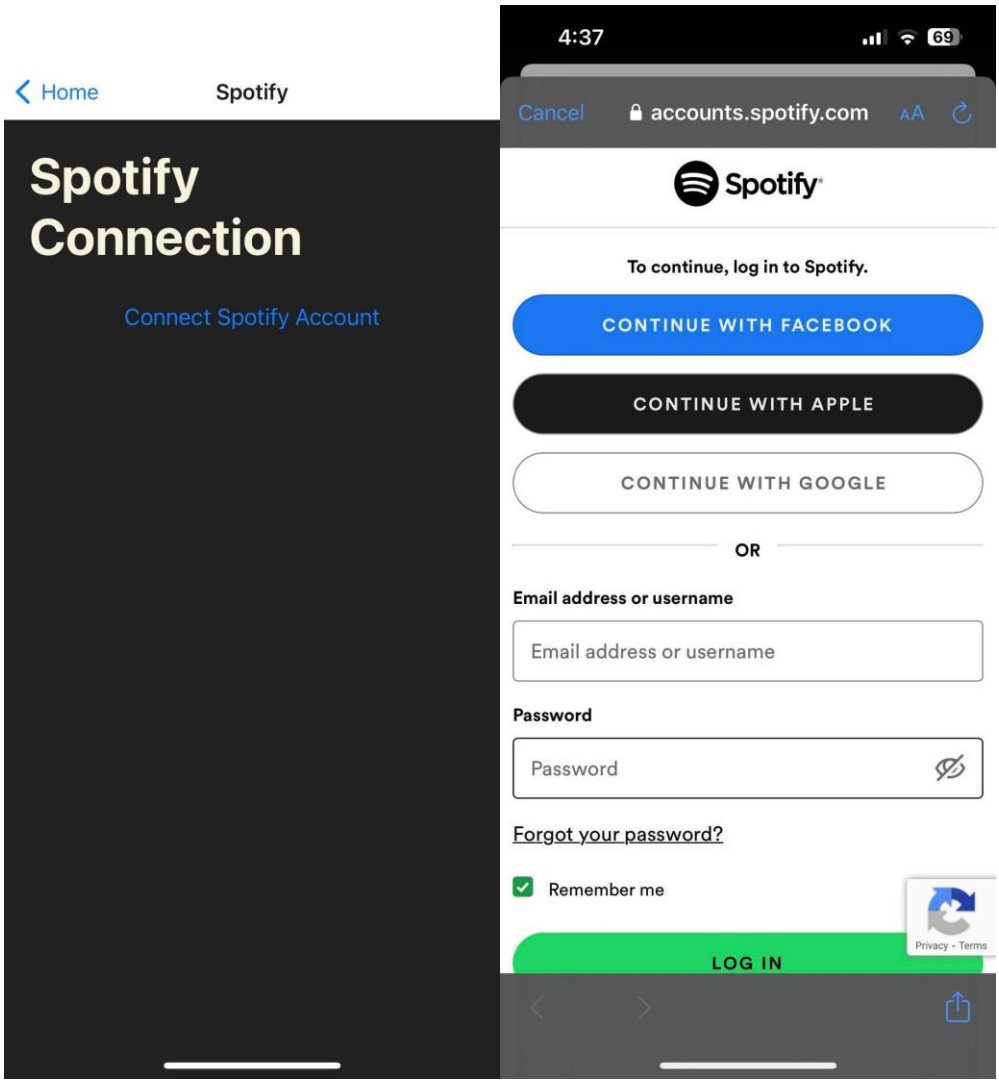

Figure 4.2.2.2: User Connecting their Spotify Account to Podify

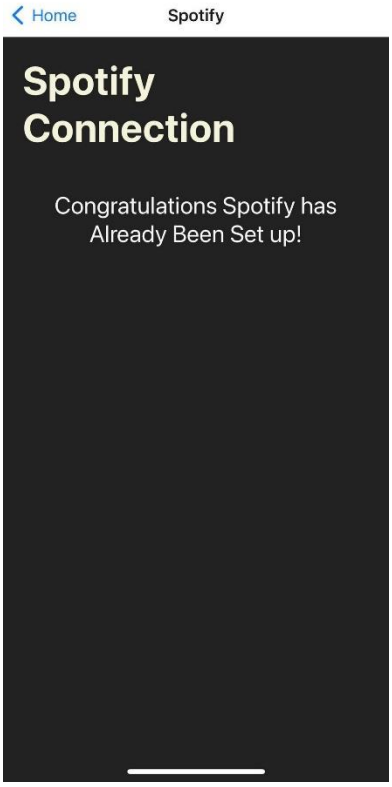

Figure 4.2.2.3: User that has already set up their Spotify Account with Podify

4.2.3 Push Notification

When a user begins to listen to a podcast, they are sent a push notification asking if they would like to take notes on the podcast that they are listening to. This can be seen in Figure 4.2.2.1. If the user clicks on the notification, then they are brought to the Podify App where they can see all podcasts that they have notes for. This can be seen in Figure 4.2.3.2.

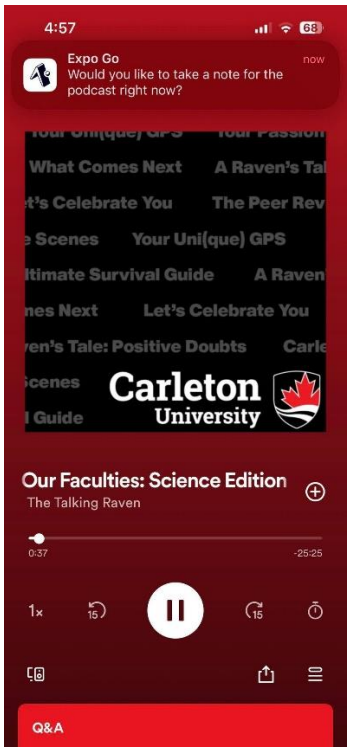

Figure 4.2.3.1 Push Notification Shown When User is Listening to Podcast.

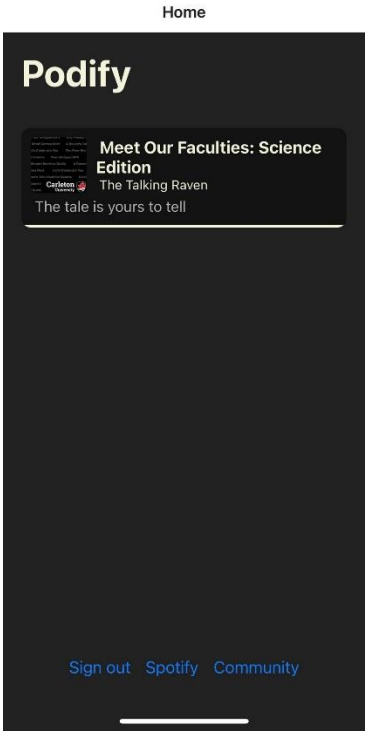

Figure 4.2.3.2: Podify Home Screen with a Note for the User.

4.2.4 Taking a Note

When a user clicks on a podcast note, they are then brought to the edit screen for that podcast episode. On this screen the user can choose to make their note public or private, rate the podcast on a scale of 1 through 5, and actually edit the text of the note. After the user is done, they simply just go back, and the note is automatically saved to the backend. See Figure 4.2.4.1.

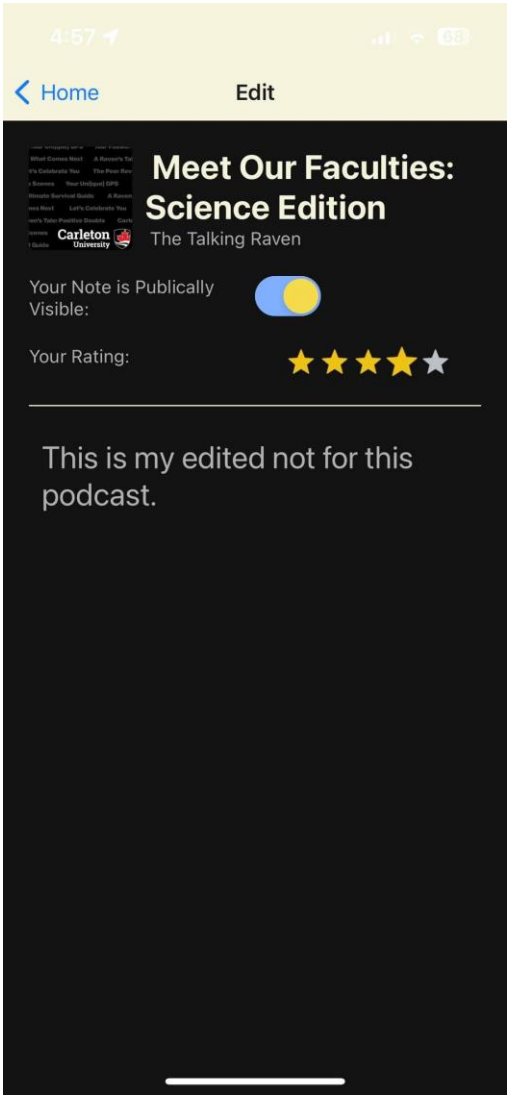

Figure 4.2.4.1: Note Taking Screen for a Podcast Episode

4.2.5 Viewing Community Notes

When the user clicks on the community tab, they are brought to a page showing the collection of podcasts that other users are listening to with their average rating calculated. The user can also see the podcast that they have listened to as part of this community. See Figure 4.2.5.1. If the user clicks on any of the podcasts, then they are brought to a page where they can see all of the public notes that users have left for certain podcasts. This can be seen in Figure 4.2.5.2. On this page the user can see the rating and the note preview. If the user clicks on any of the notes they can see the full version of the note.

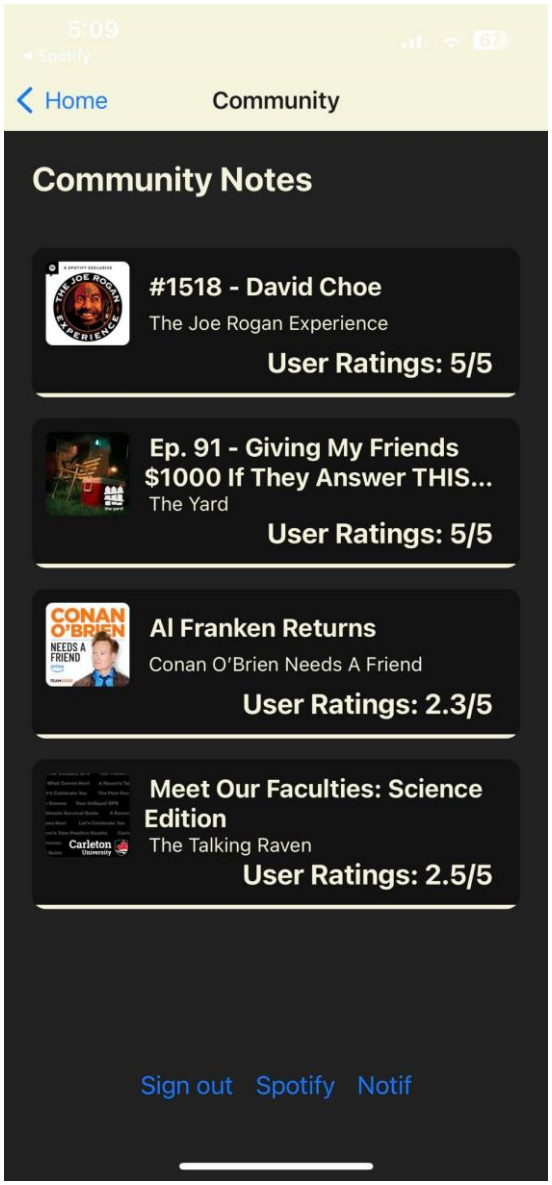

Figure 4.2.5.1: Viewing the Community Ratings for the Podcasts

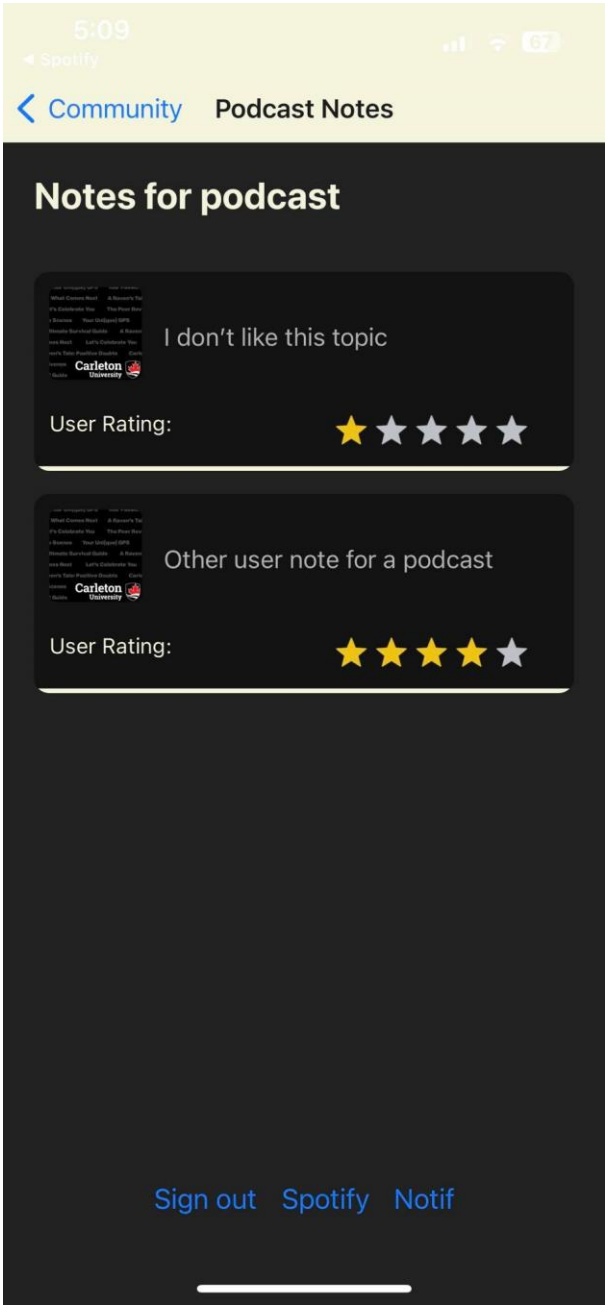

Figure 4.2.5.2: Viewing Other User's notes for a Specific Podcast

4.3 Application Testing

To ensure that Podify works correctly, many different manual tests were completed throughout the development of the application. This includes testing for both the client and server side of the application.

For the backend, every endpoint was tested with postman by providing a number of different inputs to the endpoints and then the returned results were validated. The benefit of testing the backend endpoints separately leads to less errors occurring when connecting the backend, and if an error occurs during the connection process it can be much harder to debug the issue.

On the client side, any time components were created they were tested to ensure that they act as they should. The benefit for the client side is that it is very visual and being able to see the physical changes helped speed up the debugging process. The client side was most prone to errors during navigation stages; thus, the application was tested extensively by attempting to navigate in all different ways.

4.4 Chapter Summary

This chapter covered the entire flow of the application from the client side and showed off what the end product looks like. This includes, walking through the user onboarding process and the actual use of the application. Additionally, the method of testing for both the server side and the client side were discussed.

5.Conclusion

The major goal of this project was to create a mobile application where users could take notes of the podcasts. This goal achieved to the fullest by ensuring that users were able to connect their Spotify Account to the application and then the data from Spotify was used within the application. The frontend was pleasing and easy to use for user, while the backend was functional to achieve the main goal.

Some features that did not get implemented due to the time constraint of this project are creating a recommender system for other podcasts and hosting the client. A recommender system would have been an interesting feature to add for users to discover new podcasts that fit their listening style. Since users can rate the podcasts that they have listened to, this data could have been used within the recommender system. This feature would have slotted nicely into the community tab of the application and would have enhanced the application as a whole.

The next step for Podify is to host the client of the Expo-app. This requires more research as there are standards that the app needs to meet to be published to the Expo app store. If Podify

were to gain significant traction it would be interesting to see if Spotify themselves would be interested in implementing the features of Podify directly into their own platform. Podify as an application is created to enhance the podcast listening experience and if the features could be natively available within Spotify it would be the best cast scenario.

In conclusion, Podify was able to meet its main goal of being a note taking platform for podcasts. With some fine tuning, Podify should be able to be published onto the Expo app store so that all users can access this great application.

6. References

[1] Ruby, D. (2023, April 7). *41+ podcast statistics for 2023 (Listeners & Market Size)*. Demand Sage. Retrieved April 12, 2023, from https://www.demandsage.com/podcast-statistics/

[2] Lin, Y. (2022, December 13). *10 powerful podcast statistics you need to know in 2023 [infographic]*. Oberlo. Retrieved April 12, 2023, from https://www.oberlo.com/blog/podcast-statistics

[3] Howarth, J. (2023, January 18). *Number of podcast listeners (2023)*. Exploding Topics. Retrieved April 12, 2023, from https://explodingtopics.com/blog/podcast-listeners

[4] Locke, R. (2015, August 19). *Why people who take notes all the time are more likely to be successful*. Why People Who Take Notes All the Time Are More Likely To Be Successful. Retrieved April 13, 2023, from https://www.lifehack.org/298313/10-reasons-why-note-takers-are-the-fast-track-success

[5] Comaford, C. (2013, November 7). *Got inner peace? 5 ways to get it now*. Forbes. Retrieved April 13, 2023, from https://www.forbes.com/sites/christinecomaford/2012/04/04/got-inner-peace-5-ways-to-get-it-now/?sh=171e2e1c6672

[6] Loh, T. (2022, August 5). *Prognosis: Reminder list can improve your memory*. Bloomberg.com. Retrieved April 15, 2023, from https://www.bloomberg.com/news/newsletters/2022-08-05/prognosis-reminder-list-can-improve-your-memory

[7] Buzzsprout. (n.d.). *Podcast statistics and data [March 2023]*. Buzzsprout. Retrieved April 15, 2023, from https://www.buzzsprout.com/blog/podcast-statistics

[8] Buck, A. (n.d.). *Mobile apps vs mobile websites: Which is best for 2023?* Mobiloud. Retrieved April 15, 2023, from https://www.mobiloud.com/blog/mobile-apps-vs-mobile-websites

[9] *What is Firebase?* Educative. (n.d.). Retrieved April 15, 2023, from https://www.educative.io/answers/what-is-firebase

[10] Google. (n.d.). *Firebase authentication | simple, no-cost multi-platform sign-in*. Google. Retrieved April 15, 2023, from https://firebase.google.com/products/auth

[11] *Showcase · REACT native*. React Native RSS. (n.d.). Retrieved April 15, 2023, from https://reactnative.dev/showcase.html

[12] *React native · learn once, write anywhere*. React Native RSS. (n.d.). Retrieved April 15, 2023, from https://reactnative.dev/

[13] Borozenets, M. (2022, October 13). *React native Init vs expo 2022: What are the differences?* Fulcrum. Retrieved April 15, 2023, from https://fulcrum.rocks/blog/react-native-init-vs-expo

[14] *Spotify for Developers*. Spotify, 2021, developer.spotify.com/documentation/web-api/.

[15] Google. (n.d.). *Schedule functions | cloud functions for Firebase*. Google. Retrieved April 16, 2023, from https://firebase.google.com/docs/functions/schedule-functions

[16] *USEREF*. React. (n.d.). Retrieved April 16, 2023, from https://react.dev/reference/react/useRef