Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2024

Silk Sonic
Track Records

Patrick Arzoumanian
Gustavo Chavez
Abraham Mookhoek
Ahmed Ullah
Spencer Whitehead

Revision History

Revision	Date	Author(s)	Description
0.1	1.22.2024	PA, GC, AM, AU, SW	document creation
0.2	2.11.2024	PA, GC, AM, AU, SW	complete draft

CONTENTS

List of Figures

List of Tables

1 Introduction

The concept of Track Records, a music analytics web application, is to offer a variety of data analytics tools in an easy and intuitive form. Users will be able to utilize these tools in order to look back on their Spotify listening habits. The data in question relates to each user's provided music streaming history. Users will be able to understand their listening habits by utilizing our application's data visualization as well as keep a detailed journal of their musical history. To make this possible, some key requirements include a calendar interface to visualize the user's listening history, a data visualization interface to provide another visual representation of other fringe statistics, and a Journal interface to provide users a way to document their thoughts on a given day of listening. To achieve these requirements, the scope of our project will be primarily based in popular web-development technologies such as Firebase databases and next.js. Since our web-app is user focused, we will also be handling the authentication of their Spotify accounts as well as a database to store their data.

2 System Overview

Track Records will be structured with three layers, Client-Side, Business, and Data. Each of these layers combine to create a cohesive system capable of providing an intuitive and responsive experience for the user.

The user interface of a website that allows users to communicate with our web server via a browser is known as the Client-Side Layer. In order to show the data that the client has requested, the client-side talks with the server to send HTTP requests and receive HTTP responses to display the requested feature back to the user. Our web-app has many sub-components such as the login view, calendar view, journal view, and the settings view.

The business logic of the web application is managed by our Business Layer. It is made up of models, controllers, and services that carry out the functions for user requests. To retrieve or modify data as needed, the business layer communicates with our Data layer. It also houses our web service API which is primarily responsible for communicating back and forth between our Client-Side and Data Layer.

Our Data Layer houses the storing, querying, and requesting of data from either 3rd party APIs or the database service itself. While the 3rd party APIs are queried via our database connection housed in our back-end, our data is stored securely server-side. This layer can also house our web-app's project files.

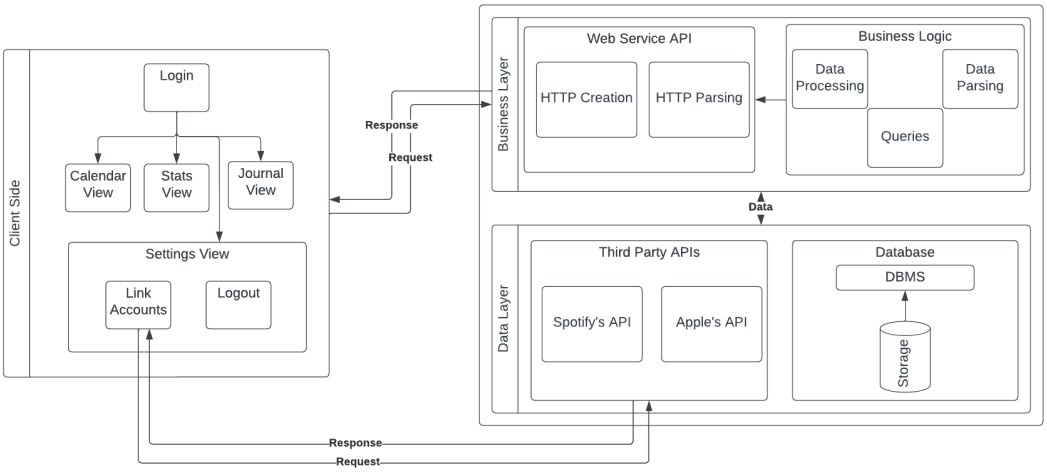

Figure 1: System architecture

3 Client Side Layer Subsystems

In this section, the layer is described in terms of the hardware and software design. Specific implementation details, such as hardware components, programming languages, software dependencies, operating systems, etc.

3.1 Layer Software Dependencies

Dependency/Version	Description
eslint-config-next@14.0.4	Javascript parsing and pattern recognition
eslint@8.56.0	Javascript parsing and pattern recognition
next@14.0.4	React framework
prettier@3.2.4	Formatting
react@18.2.0	Building interfaces
tailwindcss@3.4.1	Extra convenience for building interfaces
@types/node@20.10.7	Extra structures for node
@types/react@18.2.47	Extra structures for react
prettier-plugin-tailwindcss@0.5.11	Formatting for Tailwind
autoprefixer@10.4.16	Parsing CSS
postcss@8.4.33	Dependency for AutoPrefixer
react-dom@18.2.0	Entry for React to DOM
react-firebase-hooks@5.1.1	React Hooks for Firebase
@mui/icons-material@5.15.7	Material user interface library
@mui/material-nextjs@5.15.7tailwindcss	Material user interface library
@mui/material@5.15.7	Material user interface library
@emotion/cache@11.11.0	Incorporate CSS in JS
@emotion/react@11.11.3	Incorporate CSS in JS
@emotion/styled@11.11.0	Incorporate CSS in JS
zustand@4.5.0	State management in React

Table 2: Client-side layer software dependencies

3.2 Login Subsystem

The login view will be a section of the web page including the splash page introducing the site and its functionality along with a login button that will allow the user to create an account with our service. In doing so, we will prompt the user to connect their Spotify account. The application will need a valid email and password to successfully identify and create a unique user.

3.2.1 Subsystem Software Dependencies

This subsystem relies on the Next.js, Next-Auth.js, and Tailwind CSS frameworks.

3.2.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

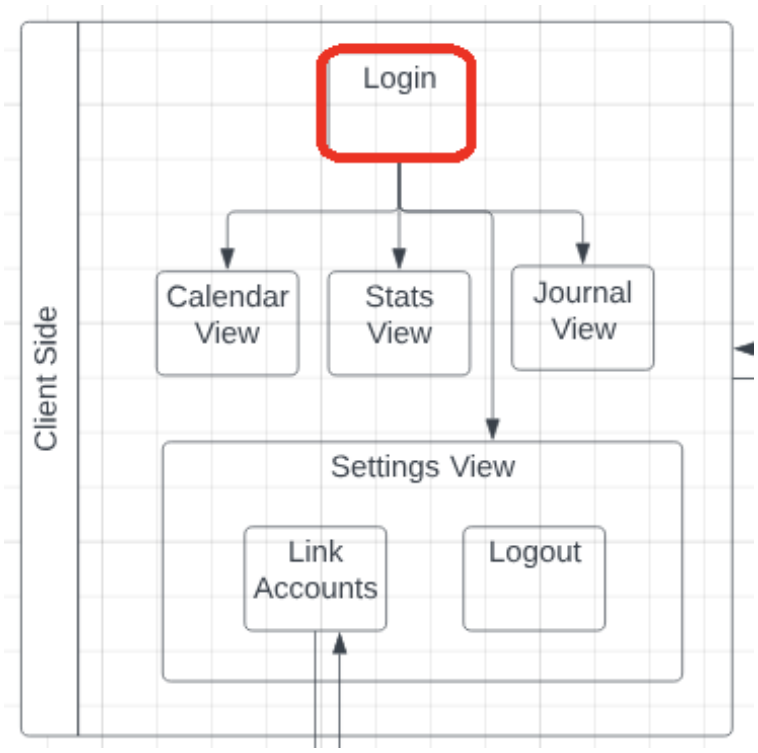

Figure 2: Login Subsystem

3.2.3 Subsystem Data Structures

This subsystem does not introduce any new data structures but utilizes created object types that allow us to store user-session information. This includes the name, email, access token and refresh tokens.

3.2.4 Subsystem Data Processing

One of the protocols used for login is OAuth 2.0. Used for authorization, OAuth 2.0 allows Track Records to query and update user information with Spotify and Google Firebase on behalf of the user. The logic is implemented through the use of Next-Auth.js, a library that handles login for Spotify and browser sessions.

3.3 Calendar View Subsystem

Track Record's default interface will be a calendar view of the userâs listening history. This will require queries to the database for the purposes of visualizing the userâs account history. Additionally, the user will be able to select what information is displayed in their calendar view, such as liked songs or listening history.

3.3.1 Subsystem Software Dependencies

@fullcalendar/daygrid@6.1.10
@fullcalendar/react@6.1.10
Rendering full-sized calendar

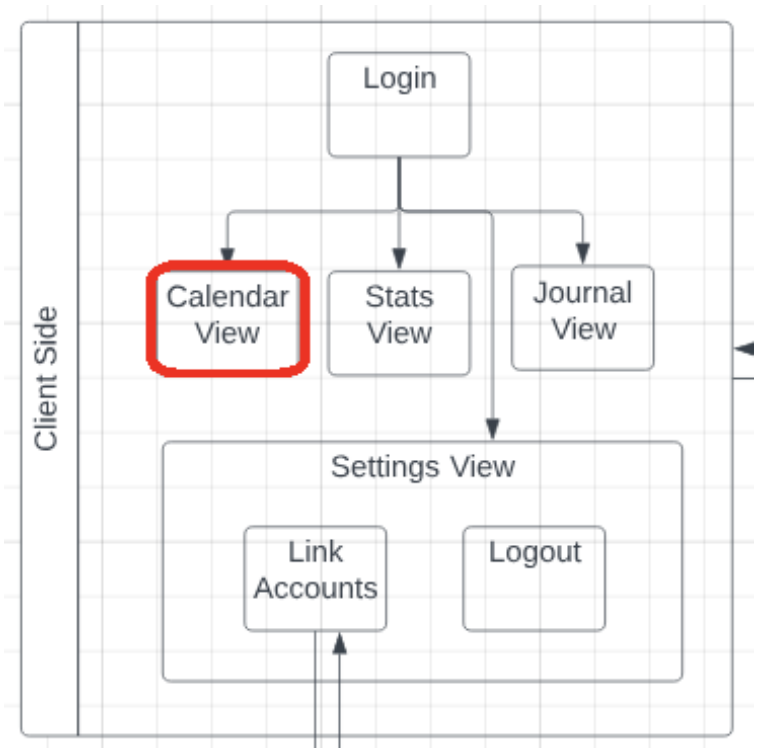

Figure 3: Calendar View Subsystem

3.3.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

3.3.3 Subsystem Data Structures

The Calendar component uses the FullCalendar npm module. The FullCalendar module has a few fundamental components we plan to utilize such as Events, for showing number of tracks added and listened, and the Day Selection Hook, used to allow the user to select the day. Upon selection of a day, the Day sub-component will give us data such as the date and the events listed under that day. This data structure will be key to our interactivity and allow us to conditionally render the "Today's Tracks" pane. To populate this pane, we will be using a "TrackCard" custom component that populates it's data via "Track" object queried by the front end.

3.3.4 Subsystem Data Processing

The calendar view subsystem will process data queried from firebase. The returned track objects will be organized into two different front-end objects. One group to represent which songs were added to the user's library and another to represent when a song was listened to. These two json objects will be formatted and saved into our global state management tool, zustand. Once these variables have been set in zustand, our Calendar component can populate with event's listing showing how many songs have been added and listened to. Then, when a user clicks on a day, we will already have the tracks sorted and can just populate the "Today's Tracks" pane without the need for added filtering.

3.4 Stats View Subsystem

The stats view will allow the user to view information about their listening history and patterns represented in different formats, with customizable parameters. The information will be grabbed from the

database and presented to the user through pre-selected view types.

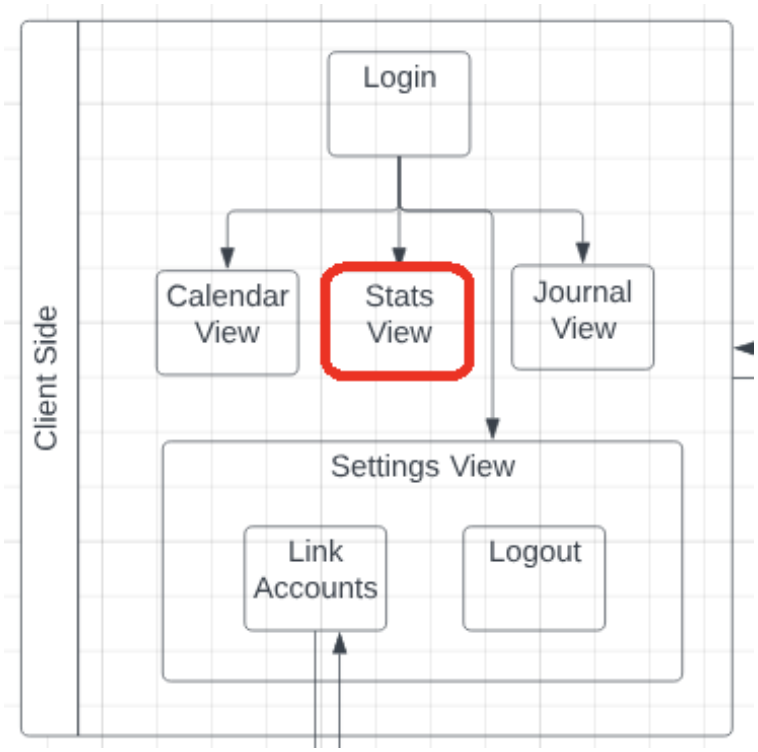

Figure 4: Stats View subsystem

3.4.1 Subsystem Software Dependencies

plotly@1.0.6
Simple node.js wrapper for stats visualization

3.4.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

3.4.3 Subsystem Data Structures

The stats view subsystem will implement the "Track" custom class object to easily access information on each track from a user's account. The Track object will have specific data only used in the stats view such as the "dance-ability" metric given to us by Spotify. We will need to format the data in preparation for the specific input the react-plotly graphs. Therefore, the specific parameters the various plotly graphs we plan to use must be adheared to when formatting our data.

3.4.4 Subsystem Data Processing

The stats view subsystem will process the tracks returned from firebase queries. Depending on the properties selected by the user, statistics will be calculated on all applicable tracks and displayed in some form of graph. Furthermore, any in house stats we plan to compute must be done on the server side as to not slow down our client. Once the user opens the app, we plan to start computing these customs in-house stats ourselves and save them to the zustand store. Then, once the user renders in the stats component, we will query our server with the processed stats and display them to the user.

3.5 Journal View Subsystem

The journal view will function as an interface that centralizes access to every journal entry created by the user. The user will then be free to create, edit, and delete entries as they please. In order for these changes to remain persistent, the application will need to save this information in the database.

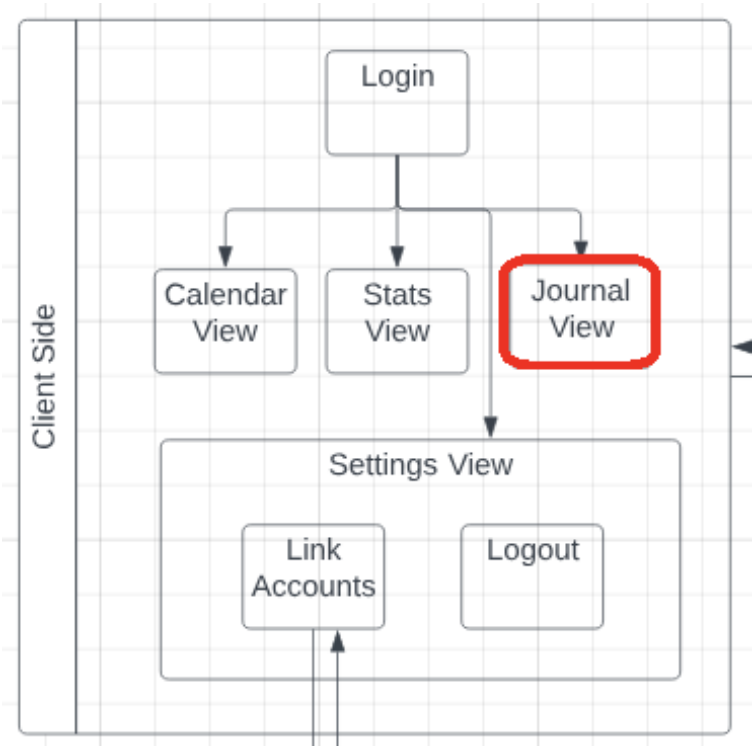

Figure 5: Journal View subsystem

3.5.1 Subsystem Software Dependencies

This subsystem has no unique software dependencies.

3.5.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

3.5.3 Subsystem Data Structures

The journal view subsystem will implement the "Journal" custom class object to easily access information on each journal entry a user has made. The main data structure for the journal will the our custom "Entry" component. This component will have the date, the tracks listened to on that day, and a string of their entry saved all into one object.

3.5.4 Subsystem Data Processing

The Entry data will only be queried when the user enters journal view, as the extra computing load on log-in would slow down the app heavily. The entries will be put in an array of "Entry" components. Due to the nature of dates and journal entries, the sorting and displaying will be very simple. We plan to place all the entries in chronological order, hence, no extra data processing will be needed other than a simple chronological sort based off the entry date.

3.6 Settings Subsystem

The settings view will offer the user a variety of customizable account and website parameters. Additionally, this is where the user can link accounts and logout. As with everything else, these settings must be saved in the database to ensure that the changes are persistent.

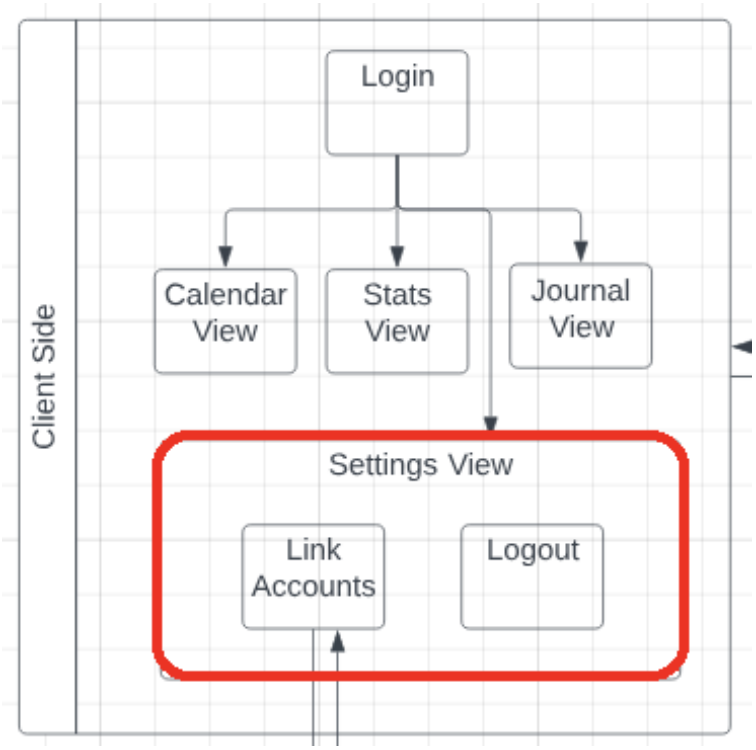

Figure 6: Settings View subsystem

3.6.1 Subsystem Software Dependencies

This subsystem has no unique software dependencies.

3.6.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

3.6.3 Subsystem Data Structures

The settings view subsystem will contain json objects to allow users to specify account details and preferences, which will be stored in firebase. Additionally, the user may submit their extended listening history as a zip file. This file's contents will be stored into multiple "Track" custom class objects. There are no other data structures being utilized in this subsystem.

3.6.4 Subsystem Data Processing

The settings view subsystem will process the zip file provided by users if they want to see statistics on their previous listening history. The data in the zip file will be parsed and placed into "Track" objects before being stored in firebase. Beyond this, there is no need to process any data. It will simply display the values contained in the json objects returned from a firebase query among other static objects.

4 Business Layer Subsystems

In this section, the layer is described in terms of the hardware and software design. Specific implementation details, such as hardware components, programming languages, software dependencies, operating systems, etc. should be discussed. Any unnecessary items can be ommitted (for example, a pure software module without any specific hardware should not include a hardware subsection). The organization, titles, and content of the sections below can be modified as necessary for the project.

4.1 Layer Software Dependencies

eslint-config-next@14.0.4
eslint@8.56.0
Javascript parsing and pattern recognition

next@14.0.4
React framework

prettier@3.2.4
Formatting

react@18.2.0
Building interfaces, etc

@types/node@20.10.7
@types/react@18.2.47
Extra structures for node and react

4.2 Web Service API Subsystem

The Web Service API serves as the gateway between the user interface and the business logic. It is responsible for handling HTTP requests and responses, serving as the external interface to the system. This subsystem communicates directly with the client-side components, including web browsers, mobile applications, or any user interface that makes use of the web app.

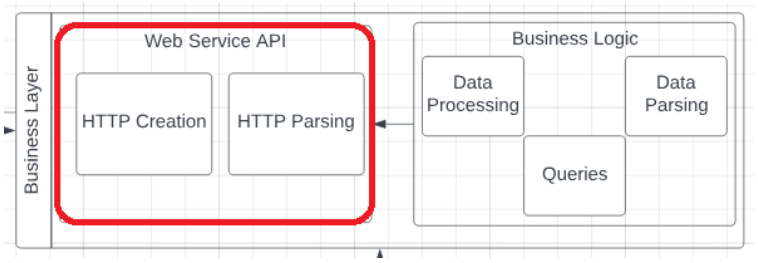

Figure 7: Web Service API subsystem

4.2.1 Subsystem Software Dependencies

This subsystem has no unique software dependencies.

4.2.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

4.2.3 Subsystem Data Structures

This subsystem does not introduce new data structures, but leverages standard HTTP requests and responses to facilitate communication between the server-side and client-side components.

4.2.4 Subsystem Data Processing

The API handles requests by validating and parsing incoming data and then routes these requests to appropriate business logic functions. The responses are formatted using JSON and returned to the client, ensuring smooth data flow.

4.3 Business Logic Subsystem

The Business Logic Subsystem forms the core of the app, responsible for processing, analyzing, and managing data retrieved from Spotify. It communicates with the database for processing data and sends the processed data to the Web Service API to display back to the user.

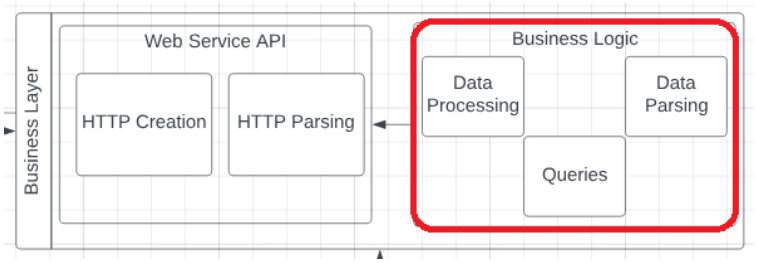

Figure 8: Business Logic subsystem

4.3.1 Subsystem Software Dependencies

This subsystem has no unique software dependencies.

4.3.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

4.3.3 Subsystem Data Structures

Functional modules within this subsystem process and manage data without specific new data structures, focusing only on operations like data retrieval, analysis, and manipulation.

4.3.4 Subsystem Data Processing

This subsystem involves analyzing user data to generate insights and manage application logic, such as recommendations and content personalizations. This ensures efficient processing of data to meet Track Records' functional requirements.

5 Data Layer Subsystems

In this section, the layer is described in terms of the hardware and software design. Specific implementation details, such as hardware components, programming languages, software dependencies, operating systems, etc. should be discussed. Any unnecessary items can be ommitted (for example, a pure software module without any specific hardware should not include a hardware subsection). The organization, titles, and content of the sections below can be modified as necessary for the project.

5.1 Layer Software Dependencies

eslint-config-next@14.0.4
eslint@8.56.0
Javascript parsing and pattern recognition

next@14.0.4
React framework

prettier@3.2.4
Formatting

react@18.2.0
Building interfaces

@types/node@20.10.7
@types/react@18.2.47
Extra structures for node and react

5.2 Third Party APIs Subsystem

The Third Party APIs subsystem allows Track Records to communicate with external music streaming services such as Spotify and Apple Music. This subsystem retrieves user-specific data about their listening habits, ensuring that Track Records is current and relevant to the user's current music listening trends.

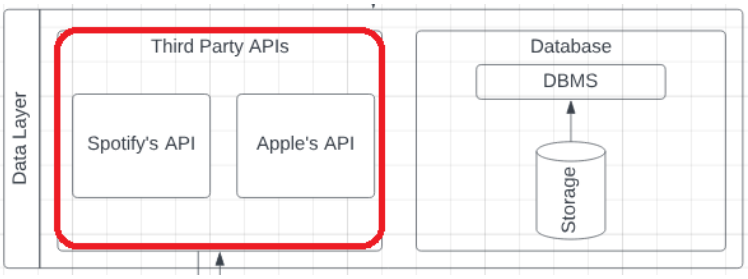

Figure 9: Third Party APIs subsystem (note: Apple's API can be ommitted for the time being as implementation of Apple music is a stretch goal)

5.2.1 Subsystem Software Dependencies

spotify-web-api-node@5.0.2
Node.js wrapper for Spotify's Web API

5.2.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

5.2.3 Subsystem Data Structures

Integration with Spotify is managed through wrapper functions and objects that standardize interactions with Spotify's API, ensuring seamless integration with Track Records' JavaScript and Next.js frameworks.

5.2.4 Subsystem Data Processing

Data from Spotify's public API is retrieved, processed, and formatted to fit Track Records' data structures and UI requirements. As well as modifications on behalf of the user can be made to modify their playlists, songs, user information, etc.

5.3 Database Subsystem

The Database subsystem stands as the backbone for the secure storage, retrieval, and modification of user data. This encompasses their musical listening history, journal entries, and user account details. The subsystem not only ensures data integrity but also provides efficient querying mechanisms to cater to user requests.

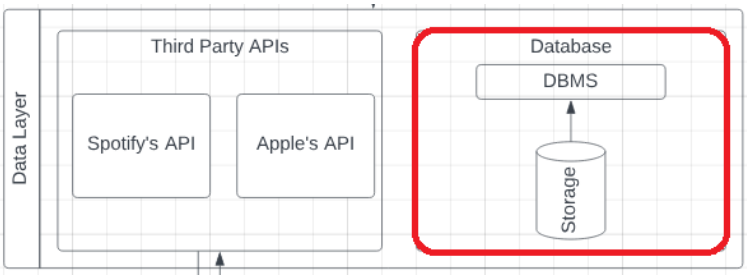

Figure 10: Database subsystem

5.3.1 Subsystem Software Dependencies

react-firebase-hooks@5.1.1
React Hooks for Firebase

@auth/firebase-adapter@1.0.13
Firebase adapter for authentication

firebase-admin@11.11.1
Firebase admin SDK for Node.js

firebase@10.7.1
Database JavaScript library for web and Node.js

next-auth@4.24.5
Authentication for Next.js

5.3.2 Subsystem Programming Languages

This subsystem is written in Javascript using Next.js.

5.3.3 Subsystem Data Structures

Firebase is used to structure and store data in collections such as Users, Listening History, and Journal Entries. These collections are designed for efficient access and manipulation for data handling strategies.

5.3.4 Subsystem Data Processing

Firebase's capabilities are leveraged for real-time data synchronization and storage. This subsystem ensures data integrity and availability to support dynamic content and user interactions.

6 Appendix A

Include any additional documents (CAD design, circuit schematics, etc) as an appendix as necessary.

References