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

Architectural Design Specification
CSE 4316: Senior Design I
Fall 2023

Silk Sonic
Track Records

Patrick Arzoumanian
Gustavo Chavez
Abraham Mookhoek
Ahmed Ullah
Spencer Whitehead

REVISION HISTORY

Revision	Date	Author(s)	Description
0.1	10.23.2023	AM	Document creation
0.1	10.24.2023	PA, GC, AM, AU, SW	Figures added
0.2	11.06.2023	PA, GC, AM, AU, SW	Complete first 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 or Apple 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 NoSQL databases and ReactJs. Since our web-app is user focused, we will also be handling the authentication of their Spotify/Apple 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.

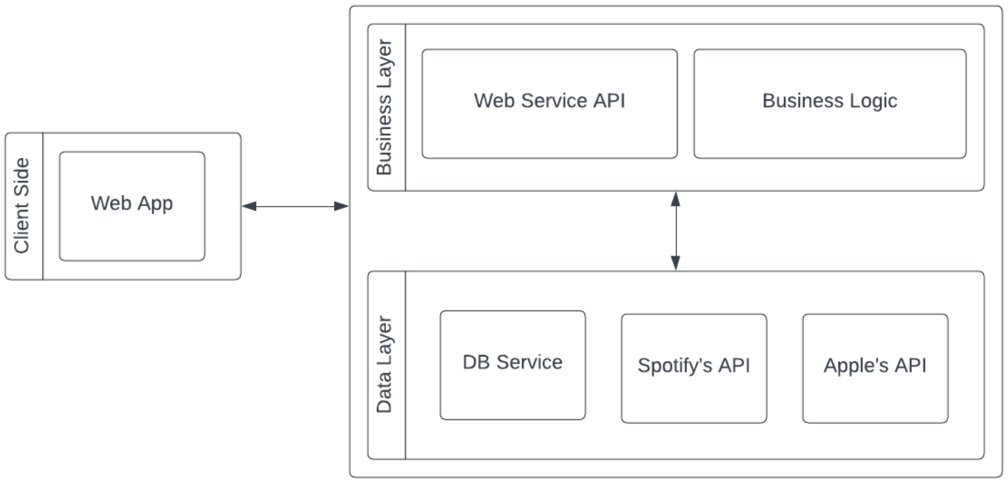

Figure 1: A simple architectural layer diagram

2.1 Client Side Layer

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.

2.2 Business Layer

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.

2.3 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.

3 Subsystem Definitions & Data Flow

This section offers an in-depth graphical representation of our design architecture, revealing the subsystems for each layer and the data flowing between them. As previously shown, Track Records consists of three layers: Client-Side, Business, and Data. The Client-Side layer consists of multiple subsystems pertaining to each major feature of the web application. The Business Layer contains two subsystems, one for the web service API and another for the business logic. Finally, the Data Layer contains two subsystems. One subsystem pertains to the third-party APIs, and the other represents the database management system (DBMS). The data flow going to and from the Client-Side Layer exists in the form of API requests and responses. The Business and Data Layer share a data flow in the form of information that will either be sent to or from the DBMS.

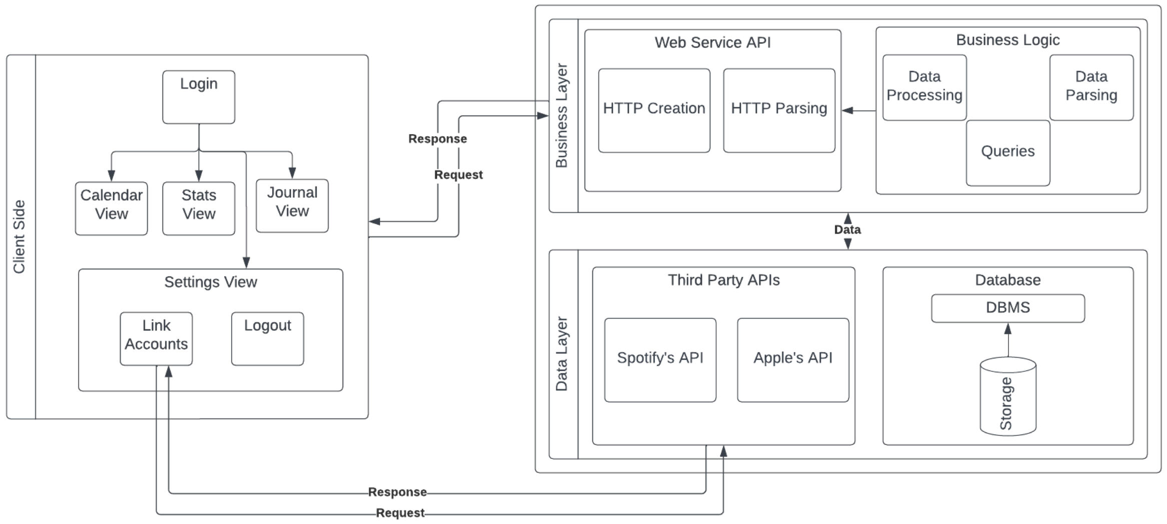

Figure 2: A simple data flow diagram

4 Client-Side Layer Subsystems

The Client-Side Layer subsystems account for the main functionalities that are available and visible to the client. Below is an in-depth overview for each of this layer's subsystems in addition to their respective assumptions, responsibilities, and interfaces.

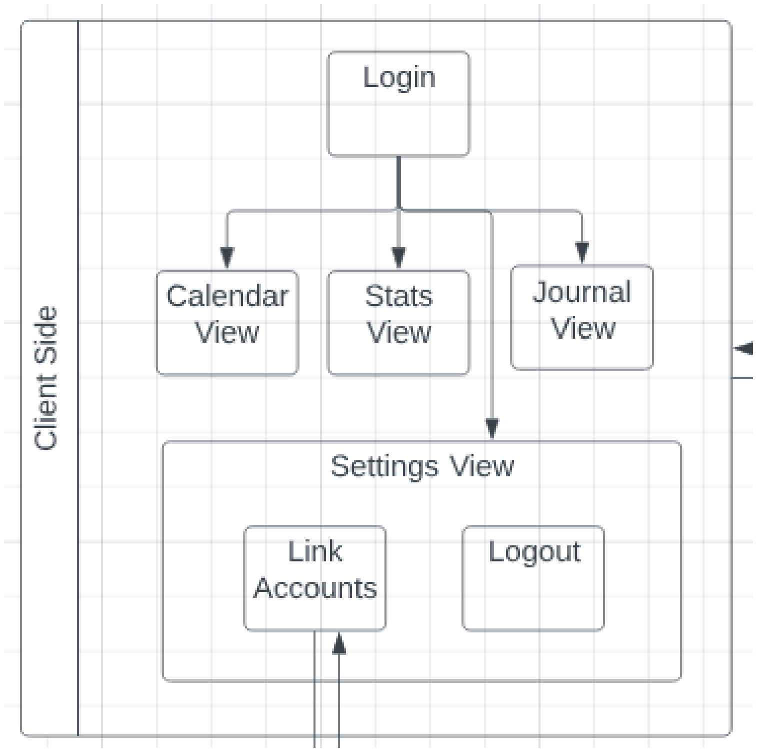

Figure 3: Client-Side Layer Subsystem

4.1 Login

The web application will allow the user to create an account with our service. In doing so, we will prompt the user with the option to connect either their Spotify or Apple Music account. The application will need a valid email and password to successfully identify and create a unique user.

4.1.1 Assumptions

Assumptions that come with login capabilities is that the email the user will use is entered correctly and the email is a valid address. This will allow us to properly send confirmation emails to the user and proceed with the account creation.

4.1.2 Responsibilities

The responsibility for the login system is that it will allow a user to create an account with the service and allow retrieval of that account through entering correct credentials. The accounts will allow us to keep track of concurrent users and to keep track of their music data. On top of that it will also allow us to store any sensitive credentials that were given to the application on behalf of the user. Which then also brings to the table the fact that any information that we keep must be protected at all costs.

4.1.3 Subsystem Interfaces

Table 2: Login Subsystem Interfaces

ID	Description	Inputs	Outputs
#001	Account Creation	User Email User Password	Account Creation Account Access
#002	Account Retrieval	Existing User Email Existing User Password	Access to Account

4.2 Calendar View

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

4.2.1 Assumptions

One particular assumption for the calendar view subsystem is that it will only display information that has been tied to the user's account. For example, if only a Spotify account is linked, then only Spotify listening history will be present in the calendar view. Furthermore, only journal entries that are tagged with a certain date will appear in the calendar view.

4.2.2 Responsibilities

The responsibility of the calendar view subsystem is to reliably present the user's listening history and journal entries. This should extend back to the earliest point in a user's music streaming history up to the present. Additionally, users should be able to apply filters to their calendar to change what information is displayed. This will require a request for the processing server to query all of the data pertaining to the visible time-frame of the calendar and any specified constraints. Any updates to the data must be accounted for when the calendar page next refreshes.

4.2.3 Subsystem Interfaces

Table 3: Calendar View Subsystem Interfaces

ID	Description	Inputs	Outputs
#003	Listening History	Specified Filters Database Query	Queried Listening History
#004	Journal Entries	Specified Filters Database Query	Queried Journal Entries

4.3 Stats View

The stats view will be used to perform a wide variety of data analytics on listening history. The user will specify the desired graph type, graph components, and data-points to use. Naturally, this functionality will require queries to the database in order to retrieve the user-specified information.

4.3.1 Assumptions

The stats view subsystem operates on the assumption that the data shown will not be entirely accurate if the user has not provided their full listening history. Additionally, it is assumed that users will face constraints when comparing data between different streaming services as Spotify and Apple Music store data unique to their platform.

4.3.2 Responsibilities

The stats view subsystem is responsible for displaying the user-specified data in the requested format. This will be accomplished by sending a request to the processing server that will then retrieve the requested data from the database. This is a core feature of the application, and as such, it must be able to reliably turn the queried data into sensible and visually appealing graphs.

4.3.3 Subsystem Interfaces

Table 4: Stats View Subsystem Interfaces

ID	Description	Inputs	Outputs
#005	Data Visualization Graphs	Graph Specifications Data Specifications	Requested Graph

4.4 Journal View

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.

4.4.1 Assumptions

The journal view subsystem operates on the assumption that users will tag their entries with all applicable information. Accurately tagged journals are essential to maintaining effective analysis.

4.4.2 Responsibilities

The journal view subsystem is responsible for storing every journal entry and allowing the user to edit each of them. This will require requests to store and retrieve the entry information from the database. Additionally, this subsystem might perform analysis on the entries to determine the user's mood in relation to the tagged song(s).

4.4.3 Subsystem Interfaces

Table 5: Journal View Subsystem Interfaces

ID	Description	Inputs	Outputs
#006	Journal Entry	Entry Title Entry Body	New/Updated Journal Entry
#007	Text Mood Analysis	Existing Entry	Mood Statistics

4.5 Settings View

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.

4.5.1 Assumptions

The settings view subsystem holds the assumption that users will enter valid music streaming service account information when attempting to link accounts. Additionally, it is assumed that the submitted lifetime history is not faulty.

4.5.2 Responsibilities

The settings view subsystem is responsible for maintaining users' preferred account and webpage settings. Additionally, it is responsible for letting the user select music streaming service accounts to link as well as add their lifetime listening history. This requires the settings to be stored in the database for persistence.

4.5.3 Subsystem Interfaces

Table 6: Settings View Subsystem Interfaces

ID	Description	Inputs	Outputs
#008	Music Account Linking	Account Email Account Password	Account Linked
#009	Lifetime Listening History	Zipped History	History Added to Database

5 Business Layer Subsystems

The Business Layer subsystems account for the back-end processes necessary to carry out user requests. Below is an in-depth overview for each of this layer's subsystems in addition to their respective assumptions, responsibilities, and interfaces.

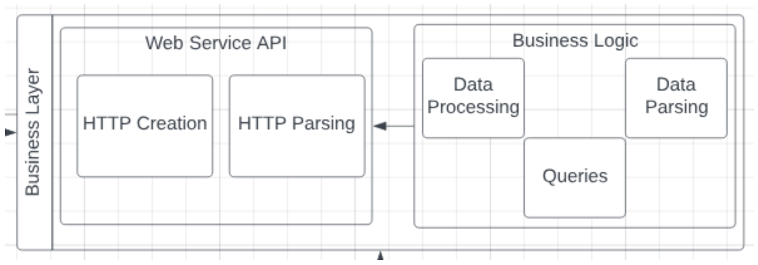

Figure 4: Business Layer Subsystem

5.1 Web Service API

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.

5.1.1 Assumptions

The Web Service API will work on the assumption that the HTTP requests it receives will conform to the expected format of its endpoints and methods. It will also be assumed that the requests it receives are valid and authenticated, along with the data it will receive from the buginess logic.

5.1.2 Responsibilities

The Web Service API subsystem will primarily be responsible for generating and sending HTTP responses to client requests, as well as handling the reception and parsing of incoming HTTP requests, extracting relevant data from headers, query parameters, and request bodies. It should also have sufficient error handling, to detect and handle errors gracefully.

5.1.3 Subsystem Interfaces

Table 7: Web Service API Subsystem Interfaces

ID	Description	Inputs	Outputs
#001	Data Display	Processed Data	Data Views
#002	HTTP Routing	HTTP Requests	Valid HTTP Responses

5.2 Business Logic

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

5.2.1 Assumptions

It will be assumed that the data received by the business logic will be valid and authenticated, and that the data will be retrieved and processed in a timely manner.

5.2.2 Responsibilities

The business logic will be in charge of retrieving data from the database, processing and transforming raw data from Spotify and Apple Music into formats suitable for analysis, as well as parsing the data by formatting, structuring, and generating reports or visualizations to be sent to the Web Service API for displaying to the user.

5.2.3 Subsystem Interfaces

Table 8: Business Logic Subsystem Interfaces

ID	Description	Inputs	Outputs
#003	Data Retrieval	Data Queries	User Data
#004	Data Processing	User Data	Processed Data Data Analytics

6 Data Layer Subsystems

The Data Layer is pivotal in ensuring that Track Records provides users with a seamless experience. This layer is responsible for fetching data from third-party APIs, securely storing user data, and serving project files. In this section, we delve into the specifics of the subsystems within the Data Layer.

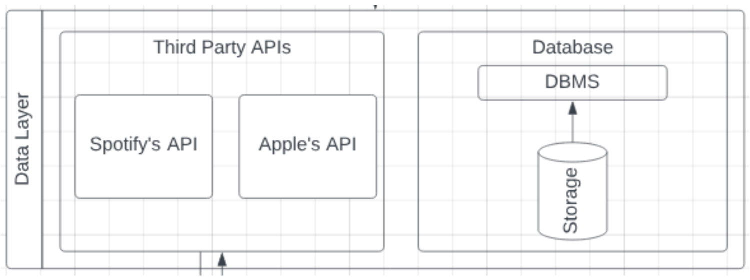

Figure 5: Data Layer Subsystem

6.1 Third Party APIs

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.

6.1.1 Assumptions

In developing the Third Party APIs subsystem, several assumptions were made. It is presumed that the APIs provided by Spotify and Apple Music are stable with a high uptime. Additionally, it's assumed that users have already authenticated their Spotify/Apple Music accounts with Track Records. The expected response structure from these APIs is also anticipated to remain consistent, which is crucial for the seamless operation of our platform.

6.1.2 Responsibilities

The core responsibilities of this subsystem encompass querying Spotify and Apple Music APIs to fetch user listening data. Alongside data retrieval, it's imperative for the subsystem to handle errors and unexpected responses from the APIs gracefully. The data fetched is of utmost importance, and therefore, the subsystem ensures its consistency when integrating with the database.

6.1.3 Subsystem Interfaces

Table 9: Third Party APIs Subsystem Interfaces

ID	Description	Inputs	Outputs
#001	Spotify Data Retrieval	User Spotify Token	User's Spotify Listening Data
#002	Apple Music Data Retrieval	User Apple Music Token	User's Apple Music Listening Data

6.2 Database

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.

6.2.1 Assumptions

When considering the Database subsystem, it is assumed that the database is scalable enough to handle the expected user load. Regular backups are believed to be in place, ensuring that user data is always protected against potential losses. Furthermore, in accordance with global data privacy norms, the database is expected to follow all necessary regulations, ensuring user data is encrypted and stored securely.

6.2.2 Responsibilities

The Database subsystem's primary responsibility is to store critical user data, such as listening history, journal entries, and account details. It retrieves user-specific data on demand, ensuring that the data served is always current and relevant. Another important responsibility is to back up data on a regular basis to protect against potential data loss scenarios. Above all, the subsystem ensures the data's integrity and consistency at all times.

6.2.3 Subsystem Interfaces

Table 10: Database Subsystem Interfaces

ID	Description	Inputs	Outputs
#003	Data Storage	User Data (Listening history, journal entries, account details)	Confirmation of Data Storage
#004	Data Retrieval	User ID/Request Parameters	Requested User Data

REFERENCES