UNIVERSITY OF SOUTHAMPTON
Faculty of Physical Sciences and Engineering

A project report submitted for the award of
MEng Computer Science

Supervisor: David Millard
Examiner: Zehor Belkhatir

Project Audyssey - A Platform for Multidimensional Journeys through Large Song Libraries

by Kathirvelan Arounassalam

April 10, 2025

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

A project report submitted for the award of MEng Computer Science

by Kathirvelan Arounassalam

Music Streaming Services have made it significantly easier to listen to and collect any amount of songs, leading to much larger song libraries. Whilst there has been heavy research into improving how to recommend songs, there has been a lack of improving the organisation of these large song libraries and better ways to listen across them.

The only method of organisation is the hierarchical use of playlists, however these have a mental tax attached to the creation and mantainance of them. This project will investigate two different approaches to viewing and organising songs: static cartesian graphs and a dynamic similarity graph. These graphs will then be used as a basis for novel visualisations of a user's listening history and song queue. This project will also investigate how the listening behaviours of Spotify users can be improved with novel software features.

Attributes originally extracted from the Echo Nest (now part of Spotify) will be used to map songs in the static graphs and compute the similarity for the dynamic graph. These features will be implemented using a desktop application (using web technologies) that interacts with the Spotify API.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

The JavaScript React library `ag-grid-react` was used to implement the table fiew for the application.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/-code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem

As technology has advanced over the years, people's personal music collections have become more and more digital, with "around 2 in 3 people listening to music using a streaming service" [1].

Initially, if a listener wanted to listen to a song or some music, it would have to be performed live, usually by the creator of said music. As such music started to be performed in concerts, where many people could listen to songs from one or more artists.

Radio allowed people to listen to music together and in the comfort of their own home. However, there wasn't full control over what to listen to.

Vinyls and CDs allowed for listening to any song in any place, with the right equipment. This made it possible for people to start treating music as a collectible physical item and build personal music collections. However these physical collections were limited by purchase cost and storage space.

As technology progressed further, we entered the streaming era, where physical collections were replaced by digital collections. These digital collections are significantly cheaper, and are much less constrained by storage space.

As such music collections have the capability to be significantly larger than their physical counterpart. The process of adding songs is much simpler digitally, allowing these digital collections to grow at a significantly faster rate than physical collections.

To navigate the landscape of a personal music collection the predominant method employed by streaming applications is to simply arrange the items in a list. This method works well on a small scale, but falls apart with the large scale that digital music collections can reach.

As such, as a workaround, most users split their mental collections into sub-collections (playlist)s, folders that allow for a lesser mental load and better maintainability.

Whilst playlists help manage the scale of songs, they themselves' have the issue "when a user's playlists become overwhelmingly numerous, streaming services begin to appear inefficient and unmanageable as collection systems"[2].

This hierarchical system allows for the better handling of large scale collections but if these playlists grow too large, then they require reorganising which is a mentally taxing activity. As such the organisational benefit provided by these playlists is lost due to the 'perceived' high mental load required. Whilst the capability and format of personal music collections have changed dramatically with technological advancements, the structure of these collections has stagnated in the streaming era.

This project will investigate different structures to see how they can improve either of the following key aspects for personal music collections:

- **Complete Knowledge of the Collection** - understanding the entire contents of the collection (to not lose/forget about songs in the collection over time) without exacting a heavy mental load
- **Replayability/Queue Building** - being able to quickly and frictionlessly create song queues to listen to (where the order of the song queue exists on a spectrum between fixed and random)

1.2 Method

First we will try to understand people's mental organisational models for their personal music collections and how they create song queues using their organisational model.

Then we will see the relationship between their mental understanding and their digital music collection.

1.2.1 Complete Knowledge over the Collection

For song organisation, the current method employed by streaming applications is a folder-based list structure (using playlists as folders).

Other methods using graphical spatial methods have been researched

- there has been research into creating 2d and 3d visualisations of song Libraries

- although there hasn't been research into making these usable from a software application point of view

This project will employ two different organisational models:

- **Static Cartesian Graphs** → each song is mapped onto a cartesian grid (both 2D and 3D) where the co-ordinates of the song is determined by the numerical attribute for that axis
- **Dynamic Similarity Graph** → each song is a node in a graph with edges between each song for each attribute and metadata. As such songs will be 'pulled' closer to together if they have a high overall similarity.

The aforementioned attributes are audio features analysed by the Echo Nest group. These include: instrumentalness, loudness, energy, etc.

1.2.2 Replayability/Queue Building

When listening to songs, the next song to be played is chosen somewhere on a spectrum between fully deterministically (manually selected) or fully non-determinstically (randomly selected):

- fixed deterministic choices occur when the listener knows which song they want to listen to next
- when the listener doesn't know which song to listen to next then the next song should be randomly chosen (such as with Spotify's shuffle feature on a playlist)

With the advent of software now being the medium with which songs are listened to, song queues can now be generated randomly from a set of songs, such as a listener's playlist.

However, there exists no framework for creating song queues where their order is elsewhere on the spectrum than near the two ends. Essentially there is no way to control this randomness without significant overhead (such as having to create a new playlists with the desired songs).

By using the graph-based views as the foundation, this project will investigate a song-queue building algorithm which allows for more of a guided randomness, that is allowing for songs to be randomly selected under the constraints of song metadata and attributes.

This project will use a software application to act as a vehicle for testing and evaluating the aforementioned new organisational structures.

1.3 Report Structure

Chapter 2 dives into the approaches towards visualisations of music related items (songs, artists, etc.) and related literature towards song attributes. Chapter 3 details the design of the software application, with Chapter 4 detailing the implementation/development process of this design. Chapter 5 presents the testing methods and results on the application and the novel concepts. Chapter 6 discusses the results, new understandings and addresses the technical improvements. Chapter 7 is a retrospective on the project as a whole, including the limitations. Finally Chapter 8 will conclude the paper, summarize the findings, and explore the potential avenues for further work using this research as a foundation.

The problem is that I want to be able to see my full collection, not just see it in bits, even if it has been broken down structurally.

Every time i see the collection (rather part of it) my mental experience and idea of it is influenced by what I saw. I want full knowledge over the whole collection, not just vague notion of the parts

When listening to my songs I want to be able to have easier dynamic movement on the spectrum (too much friction currently) - fully ordered/active - manually choose songs - guided random - listen randomly within a radius of a song - fully random/passive - just hit shuffle

Main issue is sometimes I want to only listen to a certain region of my collection - if I know the region - either this is already a playlist (issues of creating and maintaining this region) - can draw a region/create radius - if don't know the region - should be able to guide trajectory to song that I'm currently vibing with

Chapter 2

Background

2.1 Song Similarity

There are two significantly successful attempts at algorthmically computing the similarity between songs for streaming services. These streaming services mainly utilise this song similarity metric as a way to provide meaningful suggestions.

2.1.1 Pandora's Music Genome Project

One approach to copmuting song similarity is the manual approach, undertaken by Pandora. Using a team of many human musical experts, they analysed and categorised every song in their database. They have then patented this into "an n-dimensional database vector in which each element corresponding to one of n musical characteristics of the song."[3]. Whilst this system has no benefit for the organisation of songs for the user, it allows for recommendation through sonic distance - a distance calculated using the distance between the vectors of two songs. This database is also used to create radios where the next song played is guaranteed to be similar.

2.1.2 The Echo Nest Attributes

The other approach to computing song similarity is to use machine learning to extract low, mid and high level features. One such company that employed this method is the Echo Nest company who created again created an n-dimensional database of musical characteristics. These were publicly accessible through their API until Spotify bought the Echo Nest in 2014.

Spotify has used this (along with their own research into programmatically learning music characteristics [4]) to empower their internal tools[5], but they have not made these attributes public to users. The attributes and audio analysisi are only availible through their developer API which is now deprecated.

2.1.2.1 Exportify

Exportify is a tool that allows for the exporting of a Spotify user's library as `.csv` files. This `.csv` file contains all the metadata for a song and also all the Echo Nest attribute values. These files can then be inputted into a Jupyter notebook(found here) that provides data visualisations using the Echo Nest attributes, providing insights into the distributions of attributes using histograms.

2.1.2.2 Chosic

Chosic is a website that uses the EchoNest attributes via the Spotify API to provide insights into a Spotify user's collection as a whole, by finding things such as the top genre and top decade for songs. It also shows the different distributions of all the attribtutes and metadata over the entire collection.

2.2 Spatial Visualisation of Music

Streaming services predominantly think of music in terms of distinct playlists, where the songs are interactable purely as a scrollable list. There have been many other approaches to representing these music collections using spatial dimensions, where the position of the song directly maps to a characteristic of it.

2.2.1 Using the EchoNest/Spotify Attributes

2.2.1.1 Exportify

As mentioned above Exportify can provide insights using data visualisation techniques, but it also has a way to collapse the attributes into 2 dimensions. This is done using a dimensional reduction technique known as t-SNE. Whilst this is effective at creating a now organisation of songs, the resulting landscape is very difficult to understand and is limited to 2 dimensions.

2.2.1.2 Organize Your Music

This is a project built by a Spotify Employee, Paul Lamere, which plots songs from a Spotify user's library on a 2-dimensional graph using the Spotify API itself. Both the x and y axis can be set to any of the Echo Nest attributes. Unfortunately many of the features of the graph are unavailable (e.g. zoom in not working) with development having stopped 4 years ago[1]. The graphs are effective at displaying songs, however the main interactivity with these graphs is purely rooted in creating and manipulating playlists and nothing else.

2.2.2 Using Artists

2.2.2.1 MusicGalaxy

Music Galaxy is an interactive 3D visualisation of the relationships between over 70,000 artists using a graph-based approach to spatially layout these artists. To build these relationships, Spotify's API was used as it provides similar artists when one is queried. However, these graph edges are opaque to the user and are only visible when fully zoomed out.

2.2.2.2 MusicSun

MusicSun[6] is another approach at visually laying out artists based on their similarity. This interface is primarily focused on helping recommend artists to users, based on their existing preferences.

2.2.3 Ambif

The ambif[7] is an attempt at spatially rendering songs along three axis, where each axis is a music characteristic that has been extracted from song audio files using machine learning. These songs are rendered as 3D spheres and are explorable using a first person camera and WASD keyboard controls. However, this approach only implemented this concept in a game engine and was treated as a standalone proof-of-concept.

[1] Organise Your Music GitHub: https://github.com/plamere/OrganizeYourMusic

Chapter 3

Design/Method

3.1 Non-Functional Requirements

The software application was designed with the following guidelines in mind:

- **Synergy with existing software** → the features in the application should be able to be inserted in existing music streaming applications without issue (such as Spotify, Apple Music, etc.)
- **Desktop only** → to simplify the development process, the application was only developed for use on desktop
- Users can access their library in 3 clicks or less
- All controls should be intuitive and comfortable to use

3.2 Functional Requirements

Note: the below list of functional requirements have their MoSCoW prioritisations in bold.

Auth1 Users **must** be able to access their library by logging in with the credentials of the platform that library is stored in.

Auth2 The system **must** be able to interact with a user's library by using a user-specific API access token.

Lib1 Users **must** be able to view all songs from a singular playlist

Lib2 Users **should** be able to view view multiple/all playlists in their collection at once

Attr Each song in the user's library **must** have the appropriate Echo Nest attributes attached

Play Users **should** be able to control playback on another device that is playing their music

Table Users **must** be able to see the metadata and attributes for all their songs in a table

SG1 Users **should** be able to see their songs mapped along one dimension for each attribute

SG2 Users **must** be able to see their songs mapped along 2 dimensions for each combination of 2 continuous attributes

SG3 Users **must** be able to see their songs mapped along 3 dimensions for each combination of 3 continuous attributes

DeSo1 Users **must** be able to see the attributes and metadata for a song when they click on it.

DeSo2 Users **should** be able to see the most similar songs to the currently selected song

DG1 The system **must** create a logical similarity graph of all the songs currently being viewed

DG2 The system **must** be able to render logical dynamic graph in 2D

DG3 The system **must** be able to render logical dynamic graph in 3D

DG4 Users **must** be able to toggle which attributes and metadata are currently affecting the similarity graph

Fil1 Users **should** be able to filter songs on any view using continuous attributes/metadata

Fil2 Users **should** be able to filter songs on any view using discrete attributes/metadata

Fil3 Users **should** be able to toggle which playlists are currently being shown

VLJ1 Users **must** be able to see their currently playing song as a distinct node in the graph views.

VLJ2 Users **must** be able to see their queue as a directed line through the relevant songs

VLJ3 Users **should** be able to see their history rendered as a fading line (up to different preset lengths(of either number of songs or length of time))

CLJ1 Users **should** be able to set a target song for the listening journey to go to

CLJ2 Users **must** be able to select a song to randomly listen around

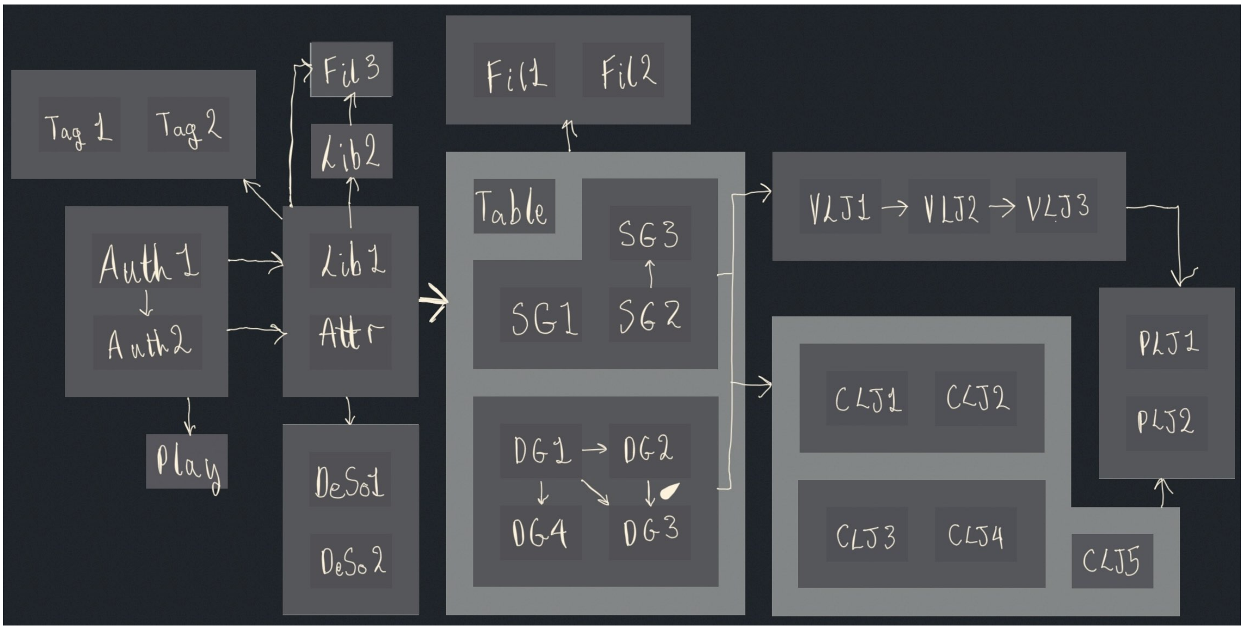

Figure 3.1: Activity Network Diagram of Functional Dependencies

CLJ3 Users **must** be able to create a segment of the listening journey where the songs are played through in a fixed order

CLJ4 Users **must** be able to create a segment of the listening journey where songs are played through randomly

CLJ5 Users **should** be able to listen to a song and then return to their original listening journey trajectory (effectively a temporary diversion)

PLJ1 Users **should** be able to view past audio journeys (segments of their full audio history), possibly as a sped up line

PLJ2 Users **should** be able to quickly re-listen to an old audio journey

Tag1 Users **should** be able to add custom tags to their songs (these can then be used to make the song similarity more informative)

Tag2 The system **should** treat these tags in a similar fashion to genres, in that they are hierarchical and not mutually exclusive

3.3 Static Cartesian Graphs

As can be seen in figure 3.2 the coordinates for the songs directly map to the values for the metric for the axis. Clicking on a song will then render a line to the axis to show the exact value on the axis.

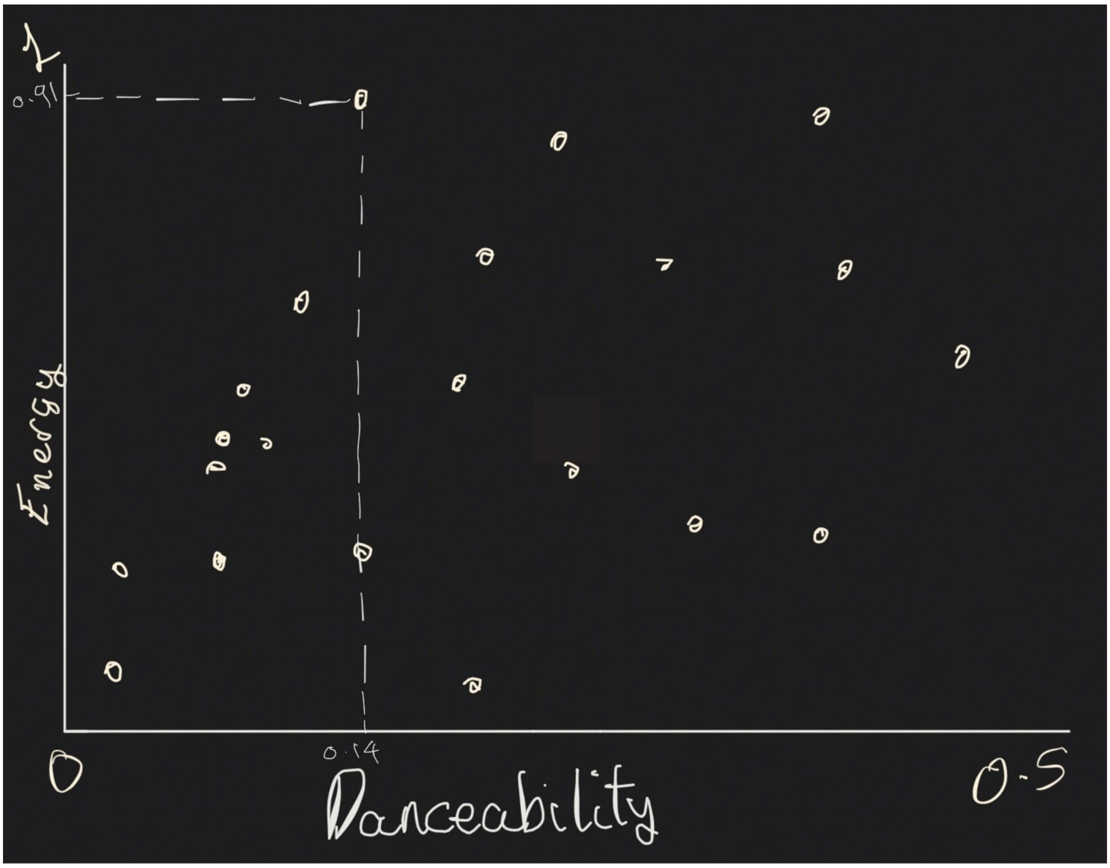

Figure 3.2: Static Cartesian Graphs UI Mockup

3.4 Dynamic Similarity Graph

3.4.1 Building the Logical Dynamic Graph

3.4.1.1 Discrete Metrics

For each discrete metric in tables 4.1 and 4.2 a query is made. All songs that have the same value for this metric have the relationship component `SameMetric(100)` added to each other (`Metric` is replaced by the metric that has been queried, such as key or album). A weight of 100 is added to make it easier to interact with the continuous edges mentioned below.

3.4.1.2 Continuous Metrics

Each continuous metric is queried for. Then for each song, all other songs are looped through performing the following:

- given song `A` and song `B`, find the difference between their values for the current continuous metric,

- divide this difference by the maximum possible difference for the current continuous metric to get a percentage similarity
- add a relation component, such as `A` –SimilarAcousticness(57.4%)– `B` (the weight is the similarity percentage)

3.4.2 Rendering the Dynamic Graph

Firstly a song is selected at random. Then for each discrete relation component on this song, the target for the relation component is also rendered (along with the edge itself). Once all songs have been rendered with their discrete edges, each song adds its continuous edges to the graph.

Songs are pulled closer together depending on the strength of the weight for each edge. A weight of 100% would pull the songs as close together as possible. If there are multiple weights between two songs then their overall similarity is used to decide how close they are to each other.

Each metric can be toggled to be inactive, meaning all edges for that metric are unrendered, with the song spheres moved to their new appropriate location based on the new overall weights of their edges.

Figure 3.3: Dynamic Similarity Graph UI Mockup

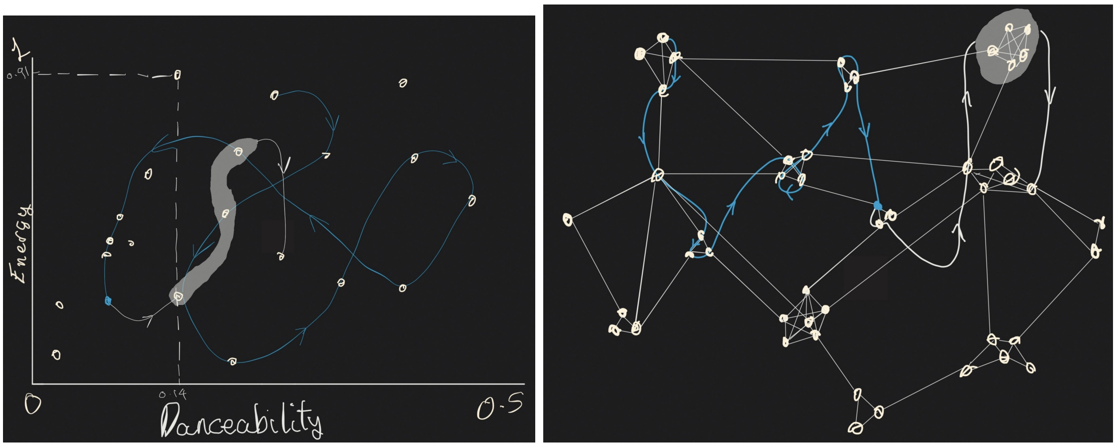

Figure 3.4: Audio Journey on both Static and Dynamic Graph UI Mockup

3.5 Audio Journeys

Audio journeys are simply the listening history, present and future for a user. These are normally shown in a list format but can also be shown as a directional line as shown in figure 3.4:

- Blue directional line: listening history of the user (limited either by number of songs or length of time)
- Blue song sphere: the current song being listened to
- Grey line: the upcoming songs in the queue
- Grey shaded region: subset of queue that will be played in a random order

The upcoming line can also be moved by the user to easily go through different songs or clusters of songs. To listen to a subset of songs in a random order, they can be marked with a random region, where those songs will all be played, but in a random order that can be shuffled by the user. The line is more curved to help distinguish it from the pre-existing straight lines in the graphs.

Chapter 4

Development

4.1 Chosen Techonology Stack

4.1.1 Application Framework: Tauri vs Electron

Tauri and Electron are the two main frameworks for building native applications using web technologies. They both create an executable than can be run on any operating system. Electron bundles a version of Chromium into the executable, increasing the size of the application compared to Tauri, but resulting in a consistent experience across any OS. Tauri uses the default browser of the operating system, which means it is smaller in size, but more inconsistent. As the application will mainly be tested using Windows machines, this inconsistency is not an issue.

The main difference between Tauri and Electron resides in the backend; Electron is JavaScript-based, whilst Tauri is Rust-based. Due to the type-safety and prior experience in Rust, Tauri is the more optimal choice for the backend. Tauri also integrates with any frontend framework much simpler than Electron does (Electron requires more manual setup).

Tauri is less widely-used than Electron however, meaning there is less community support. However, Tauri is still well-documented and has very open forums for support.

As both frameworks use web technologies, accessing a user's music collection via API is trivial.

4.1.2 Frontend Library

React.js was chosen as the frontend framework for this project for the following reasons:

+ Breaks down the code into reusable components that can be inserted and removed easily.
+ Virtual DOM: only the parts of the UI that have changed are re-rendered, meaning the entire UI doesn't have to re-render for any update
+ Widespread adoption (highest community and industry use), lots of support and excellent documentation
+ Full compatibility with three.js, an excellent 3D data visualisation library.
+ Highly effective at managing state.

4.1.2.1 Data Visualisation

Whilst using a specialised data visualisation library would be easier to use and intepret, it would be too restrictive in making the views interactive. As such three.js a 'lower-level' library was chosen even though it requires more work to create the static cartesian graphs.

Three.js was chosen partly due to its synergy with React (via the `react-three-fiber` library) and mostly for its more flexible nature. This library allows the user to build complex 3D scenes using primitive 2D and 3D shapes. As such it works well for creating the static and dynamic graphs, polar charts and ridge plots.

4.1.3 Backend

4.1.3.1 Database: Fast and Lightweight Entity Component System (FLECS)

To store a user's songs, collection structure and the song attributes a local storage approach was taken. Cloud-based would be effective at allowing for access from multiple devices, however this was considered out of scope and not worth the extra development.

To efficiently store and query the data locally, the `flecs` library (which has Rust bindings) was chosen:

+ Entity Component System - alternative paradigm to object-oriented programming, execellent at storing and querying large amounts of data efficiently, very good at parallelising complex logic. Mainly used for game engines.
+ Rust-bindings and very lightweight → slots easily into the project
+ Rich support for relationships → easy to create and query graphs using the data.

This heavy support for relationships helps with storing and accessing the structure of a user's collection. It also makes it easy to create the dynamic graph view.

Bevy, another ECS library written in Rust was considered, however it has significantly worse support for relationships and is less featureful in general.

4.2 Accessing a user's Digital Music Collection

There are many applications that allow for streaming music and creating music collections. Most of these applications have an API to easily work with, however these vary in their effectiveness. The chosen streaming application for this project was Spotify:

- + Extensive and reliable API
- + Free API access (Premium account required for controlling playback and song queues)
- +/- endpoints for detailed attributes for any song (now deprecated)
- - Risk of API deprecation or major breaking changes to the API

Whilst there have been major changes to the API that significantly affected the project, this API is still the most feature complete and easiest to use. Apple Music was considered due to also having a considerable market share and extensive API, but was not chosen due to not wanting to be locked to iOS users. YouTube music, another popular streaming service, was considered however they have no official API.

Pandora is a US-based streaming service that also has attributes on their songs in their Music Genome project, however their API is paywalled and region locked to the USA. Amazon music also has a high market share but their API is only in a closed beta so it was not considered.

4.3 Sourcing the Echo Nest Attributes

Initially Spotify was the source for fetching song attributes as Spotify was already the chosen API for accessing a user's music collection.

However, due to Spotify's deprecation of these endpoints (`Get Track's Audio Features` and `Get Track's Audio Analysis`) an alternative method was required. For most of the development process SoundCharts's API was used. This was then switched to Exportify instead due to reasons explained in the project retrospective. Where relevant both workflows will be shown in this chapter.

Note that the more ideal approach is using Exportify.

4.3.1 SoundCharts

SoundCharts has an API that allows for fetching songs with the Echo Nest attributes attached. However, these attribute values are to a significantly lesser precision. Another issue with the API is that it is paywalled. There is a free trial, however this was only for 500 API calls which was only sufficient for development and not the final evaluation.

4.3.2 Exportify

Exportify allows for the exporting of one's Spotify collection as csv files with all associated metada and attributes. These attributes are also the original precision of Spotify's endpoints.

A minor bug/issue with the Exportify process is that the explicit field was not correctly filled in, meaning that all values in the csv were left empty. How this affected the project is also detailed in the Project Retrospective chapter.

4.4 Architecture

The main application flow worked as follows

- Frontend is interacted with, function calls `invoke("rust_function")`
- The Tauri API then sends a request to the backend to run the function that was invoked
- If access to the databse is required then a query is created and run. Then for each entity within this query an event can be sent to the frontend or the data can be packed up and sent after the query has finished. Both methods use JSON to send data.
- Frontend can now either use the data to or progress the UI

4.4.1 Entity-Component Database Structure

Flecs has a concept of a world which is the data structure that stores all the entities and the components (amongst other things). To ensure efficient querying of these entities using their components, the entities are organised into multiple tables based on their archetype.

Archetypes are the groupings of entities that have the same components. This decomposition allows for high query efficiency even when there are lost of different groups.

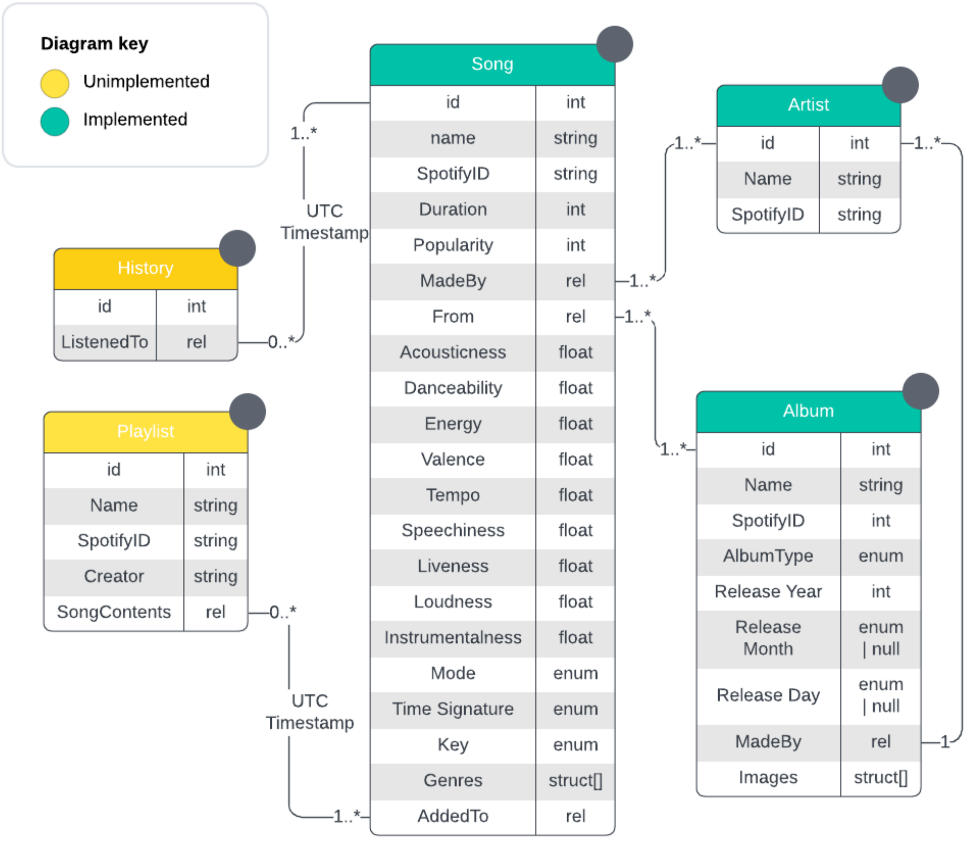

Figure 4.1: The Audyssey's Entities, Components and Relationships

Figure 4.1 shows the different archetypes that the data was organised under. To help make these archetypes easier to query, the headers of each of these archetypes directly map to a tag component, that is a component without any data attached. Also note that whilst these archetypes have many entities, the history archetype only has one entity as it is specific to the user.

4.4.2 Echo Nest Attributes & Spotify Metadata

Table 4.1: The Echo Nest Attributes

Attribute	Definition	Datatype	Possible Values	Continuous /Discrete
Acousticness	A confidence measure of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.	`float`	`0-1`	Continuous

Danceability	Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and verall regularity. A value of 1.0 is the most danceable.	`float`	`0-1`	Continuous
Energy	Represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.	`float`	`0-1`	Continuous
Instrumentalness	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.	`float`	`0-1`	Continuous
Liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.	`float`	`0-1`	Continuous

Loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.	`float`	`-60-0`	Continuous
Speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.	`float`	`0-1`	
Valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).	`float`	`0-1`	Continuous
1-5 **Tempo**	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.	`float`	$\geq$`0`	Continuous

Key	The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, and so on. If no key was detected, the value is -1.	`integer`	`None/C/C#/D/D#/E/F/F#/G/G#/A/A#/B`	Discrete
Mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.	`boolean`	`true/false`	
Time Signature	An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of "3/4", to "7/4".	`integer`	`3/4/5/6/7`	

Table 4.2: Spotify's Metadata

Metadata	**Definition**	**Datatype**	**Possible Values**	**Continuous /Discrete**
Title	The full name of the track	`String`	Unicode	Discrete
Album	What this track belongs to	`String`	Unicode	
Artist(s)	The artist(s) that contributed to this track	`String[]`	Unicode	
Explicit	Whether the track contains explicit words	`bool`	`true/false`	
Duration	Length of the track (in ms)	`Integer`	$\geq$ `0`	Continuous

Popularity	How 'popular' a track is, based on recent streams and other factors	`Integer`	`0-100`	
Added At	UTC timestamp this track was added to a playlist	`String`	Unknown-Now	
Playlist(s)	All (of a user's) playlists containing this song	`String[]`	Unicode	Discrete
Genre(s)	List of genres and sub-genres	`root: String,` `sub: String[]`	Unknown	

4.5 Application State Flow

4.5.1 Spotify Authorisation

To gain a user-specific API access token, Spotify's OAuth authorisation flow was implemented based on this official guide. This API token expires after an hour and requires sending another request to refresh this, however this mechanism was not implemented due to being needed. Once this access token has been retrieved and stored, the user's library can be loaded.

4.5.2 Downloading User's Library (with Echo Nest Attributes)

4.5.2.1 SoundCharts API

A full breakdown of this process can be found in .3, but here are the high-level steps:

- Request Spotify for all songs in a user's library
- Read serialized songs from a file into memory
- Fetch User's Spotify Library (only Liked Songs due to time constrains)
- Update songs missing attributes by requesting the SoundCharts API

4.5.2.2 Exportify

A full breakdown of this process can be found in .3, but here are the high-level steps:

- Login to Exportify.net
- Export a chosen playlist as a `.csv` file
- Convert the `csv` entries into flecs entities
- Fill missing fields (this was not implemented during this setup stage, but was done on-demand when a user clicked on a song)

4.5.3 Main Application

The `ViewSelector` UI component acts as the way to switch between views: Table, Static and Dynamic.

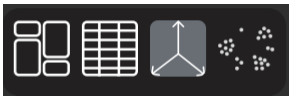

Figure 4.2: The UI Component for switching between views

4.5.3.1 Table

To make the development process easier, a library was used to implement the table UI: ag-grid. This library was chosen due to its extensive feature set and easy assimilation into the existing codebase.

Name	Artists	Acousticness	Danceability	Energy	Instrumentalness	Key	Liveness	Loudness	Mode	Speechiness	Tempo
Stoked	Weston Estate	0.149	0.778	0.642	0	10	0.106	-6.435	Major	0.137	133.04
CRUSH	Zamir, Chevy	0.0251	0.808	0.72	0.00165	6	0.315	-3.928	Major	0.0688	104.989
Heartbreak, Heartbreak	INOHA	0.0244	0.532	0.843	0.00000114	8	0.488	-5.799	Major	0.0492	161.936
Here, Tomorrow	League of Legends, Kevin...	0.108	0.383	0.603	0.0000484	2	0.138	-5.887	Minor	0.037	155.867
Entammede Jimikki Kammal	Shaan Rahman, Vineeth S...	0.141	0.567	0.488	0.000593	7	0.0645	-8.298	Major	0.314	120.017
宮ツインズ	Asami Tachibana	0.151	0.592	0.519	0.908	2	0.115	-10.734	Minor	0.0308	96.051
Dragons Rising Main Theme	Ninjago Music	0.00908	0.324	0.633	0.249	0	0.195	-7.957	Minor	0.0315	82.275
LEGO Ninjago WEEKEND WHIP - The Temporal Whip Remix	Ninjago Music, The Fold	0.000318	0.485	0.912	0	11	0.105	-6.039	Minor	0.0483	155.001
Calling After Me	Wallows	0.00598	0.636	0.797	0.00000426	8	0.114	-4.644	Major	0.0296	139.963
Remember Me (from the series Arcane League of Legends)	Arcane, d4vd	0.132	0.481	0.587	0.394	0	0.246	-7.572	Minor	0.0349	131.221
Remember Me (from the series Arcane League of Legends)	d4vd, Arcane, League of ...	0.132	0.481	0.587	0.394	0	0.246	-7.572	Minor	0.0349	131.221
Feel It - From "Invincible"	d4vd	0.0288	0.565	0.593	0	8	0.215	-5.864	Major	0.0883	120.509
Need You Around	grentperez	0.0926	0.639	0.842	0.00203	1	0.118	-5.473	Minor	0.0834	175.978
part time thing	Nafeesisboujee	0.192	0.612	0.651	0.00000347	5	0.316	-10.618	Minor	0.127	148.958
Headspace	grentperez	0.391	0.5	0.823	0.00000116	4	0.283	-6.043	Minor	0.0425	167.01

Page Size: 100 1 to 100 of 2,039 Page 1 of 21

Figure 4.3: Implementation of the Table View

4.5.3.2 Static Graphs

three.js Instanced Points Each song has their attribute values fetched based on the current attributes assigned to the axes. These values are then normalised and scaled to the size of the axis so that their spatial position directly represents their attribute values. The songs are represented by 3D spheres with an arbitrary colour. Their opacity is not 100% to make it easier to see the density of songs.

Draggable Attribute Range Sliders These range sliders allow for zooming into the graph. Only the songs within the range of the left and right sliders are shown on the graph, as all other songs are greyed out and moved outside of the bounds of the axis. The sliders that are not attached to an axis can be dragged over to replace them: e.g. valence could be dragged to replace _ in the y-axis slot.

Clicking any of the X, Y or Z buttons will result in that axis being turned off and and camera being locked to view the reamining two axes. If the Y-axis was turned off whilst the X-axis was inactive then the graph would switch to showing only the X and Z axes. Clicking an inactive axis will turn it back on, switching the graph view back to 3D.

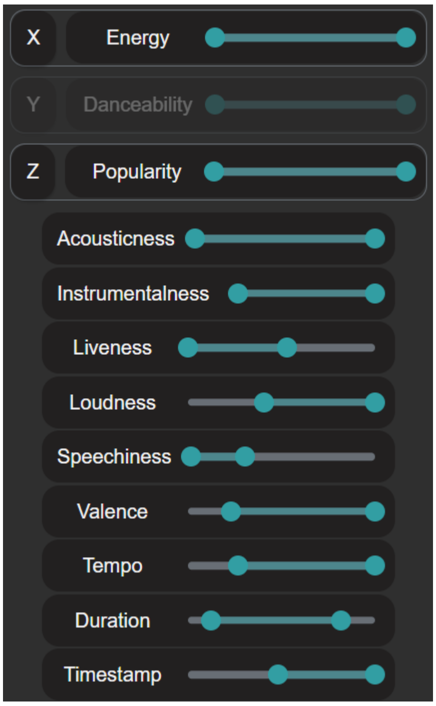

Figure 4.4: Implementation of the Draggable Attribute Range Sliders

Detailed Song Component When a user clicks on a song sphere, a request is sent to the backend to get all the components for that song. These are then rendered as

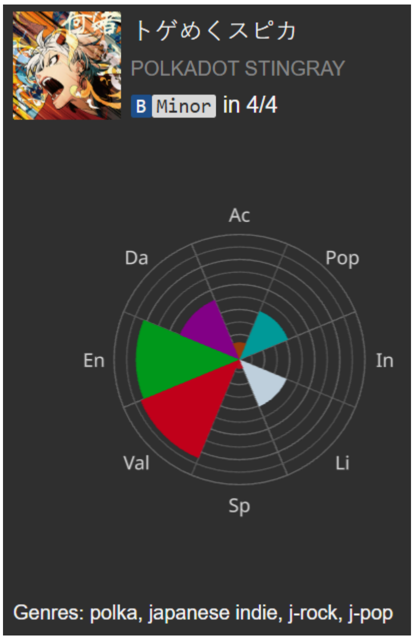

Figure 4.5: Implementation of the Detailed Song Component

shown in figure 3.3. The continuous metrics are rendered using a polar chart. Both key and mode have a specific colour background for each possible value.

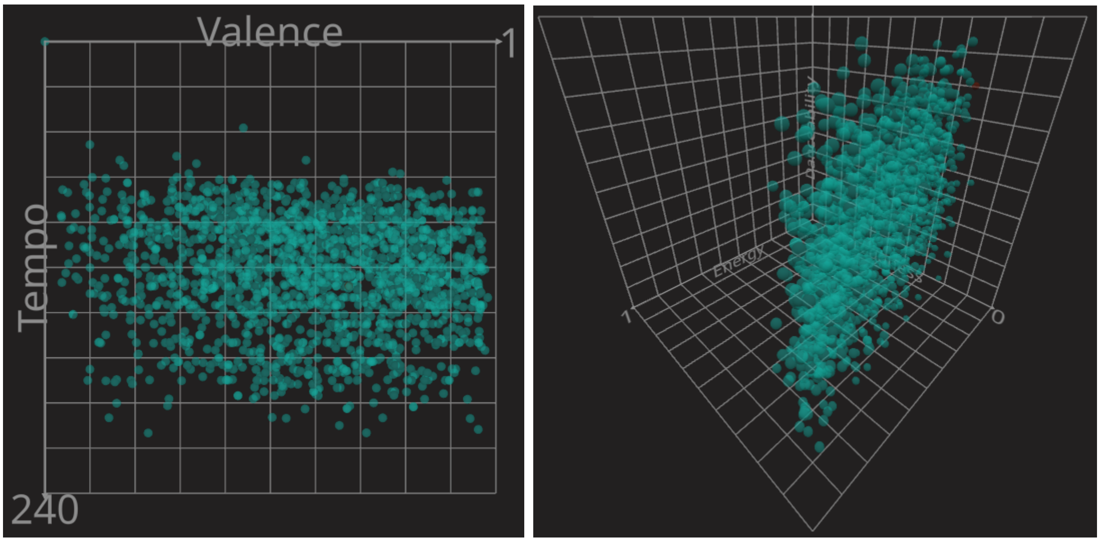

Figure 4.6: The Static Cartesian Graph implementation for 2 and 3 axes

4.6 Development Methodology/Process

A participatory approach was taken for the development process of this application (the Audyssey). Two music streaming service users who were asked help the development of the Audyssey as a review team (by acting as potential customers).

The chosen members also both represented both ends of the listening queue control spectrum. [Dhruv] is a fully active listener, they know the next song that they want to listen to, whereas [Josh] is significantly more passive, as they just pick a playlist and hit shuffle to always give them a randomly ordered queue.

4.6.1 Design Review Meeting

The initial design elements (including requirements, UI) were presented to the review team to get their feedback. Each requirement and its associated UI design were discussed with the team with any significant feedback noted down. The team were quite happy with the design resources, however they did have comments on reducing scope and new features.

4.6.1.1 Reducing Scope

After discussing the requirements during the meeting, the key takeaway was that the core features and novel approachs were the graph views and as such these should be prioritised over the other requirements.

4.6.1.2 Feature: Mixed Reality as a Medium for the Audyssey

One proposed feature/future work by the review team was the adaptation of the 3D graph views to a mixed reality device. This would make the graphs "easier to navigate and explore" but would be a lot harder to develop. As such this feature was classified as out of scope but a good potential avenue for future work.

4.6.1.3 Feature: Creating Playlists

Similar to Organise Your Music, one member of of project noted that they would like the ability to create playlists by drawing a circle around a set of songs in the graph views. However, the main reason for this was so that they could easily re-listen to those songs again. As such this desire was adapted into a new requirement of being able to store and replay audio journeys.

ReqID Users **should** be able to listen to past audio journeys.

4.6.2 Prototype Review Meeting

As shown in the project plan, this meeting was planned to occur after an MVP was developed. This MVP would be showcased to the review team and any feedback would then be implemented in time for the final evaluation. Due to time constraints detailed in the Project Retrospective chapter, this meeting did not occur. As such, this feedback was instead discussed during the final evaluation itself. These design improvements are detailed in the Design Improvements section in the Evaluation Chapter.

Chapter 5

Testing and Results

5.1 Data Collection

As the project was aiming to initially understand how people manage and use their digital music collections, then to see how these could be transformed using new concepts implemented with software, a more qualitative approach was chosen. Students from the University of Southampton with an active Spotify Account were invited to take part in a participant study after development had finished. This study was performed and complied with the ethics standards set out in `ERGO78677.A2`.

The study consisted of 1-on-1 ethnographic semi-structured interviews, typically lasting 45-60 minutes. The process was as follows:

- Provide the participant with an information sheet and consent form
- They were then asked how they build song queues - from fully random to fully ordered
- The following questions were asked:
  - How is your collection organised?
  - How do you maintain your collection?
  - How do you create queues to listen to? (expanding on their previous answer of random vs ordered)
  - How do you interact with your listening queues after creation (excluding the listening aspect itself)?
- Then the participant loaded a playlist into the software application:
  - Export a chosen playlist as a `.csv` file from the Exportify web page

  –
  – Open the Audyssey application with the aforementioned `.csv` file

- Then the user was walked through the application and then they gave their thoughts on the application and concepts within it.

Due to the semi-structured nature, some questions and topics were explored to different degrees depending on the different answers and behaviours of the participants. During the interviews, notes were taken to be used in an inductive coding process.

This process was done iteratively, modifying and combing codes after each interview. Then these codes were grouped to form a hierarchy of themes and sub-themes, along with the occurrence count for each code. This process follows from the Thematic Analysis process set out by Joy, Braun and Clarke [8].

5.2 Thematic Analysis of Interview Pre-application

Table 5.1: Hierarchical Table of Themes with Counts.

Theme	Sub-Theme	Code	Count
Collection Organisation	Method	Singular Playlist	2
		Multiple Playlists	4
	Full Collection Understanding	Vague	1
		Strong	5
	Mental Model	Distinctly Unique	2
	Familiar Structure	Easy to Use	4
	Dark Spots	Forgotten/Unfamiliar Songs	3
	Growing Song Count	Always Adding	6
		Doesn't Remove Songs	5
Playlist Management	Unique Identity	Per Artist	1
		Per Time Period	2
		Per Genre	2
		Per Mood/Vibe	3
		Activity	2
	Friction/High Mental Load	Creating Desired Playlists	2
		Switching between Playlists	2

Theme	Sub-Theme	Code	Count
		Infrequent Playlist Creation	3
		Cleaning Playlist Contents	1
	Absolute/Fixed Ordering	Playlist Contents	3
		Specific Audio Experiences	3
Listening Journeys (Queue)	Creation Process	Passive: Shuffle	5
		Active: Manually Create Queue	2
	Source	Individual Playlist	4
		Full Collection	4
		Affected by Recency Bias	3
	Shuffle	Unplayed Songs	2
		Fixing the Queue	4
		Close Enough	2
	New Song Buffer	Forgetting to Add	3
		Desired	2
	Listening as a Trajectory	Listening Rendered as a Line	4
		Boundary Songs	2

5.2.1 Collection Organisation

All the participants had a specific way that they digitally organised their Spotify music collections. Mainly they could be split into two groups: those who placed all their songs into one singular box (the Liked Songs folder) or those who split their collection over multiple playlists.

Only 1 participant felt like they had a vague understanding of their entire collection, whilst the other 5 felt like they had quite a good grasp on their collection.

Contradicting this however, 3 of the aforementioned 5, upon exploration of their collection in the software application realised there were forgotten or unfamiliar songs to them in their collection.

3 of the 4 participants who utilised multiple playlists to organise their collection also attributed benefits to this:

- knows where a song would be found in, doesn't have to know the exact location, taking off mental load
- muscle memory, familiarity

The participant who didn't attribute benfits to the structure of their playlists also had an issue with their playlists converging to the same identity[Riya: "over time, my playlists all sort of converge to the same type of song, though they're not meant to"]

Common across all 6 participants is that their collections were continuously growing over time, with only 1 participant stating that they removed songs[Riya] although rarely.

5.2.2 Playlist Management

The 4 participants who organised their collection using playlists also had common concepts and behaviours.

All 4 participants created their playlists with a distinct identity in mind whether this be for a specific artist, a time period, a genre, a mood or vibe or an activity like the gym.

Many of the participants however noted common cases where they felt friction in interacting with their playlists. 2 participants felt that it was "too much effort" to create playlists with distinct identities that they felt were missing from their collection as there are "too many songs" and "it would take too much time".

2 participants also noted that when switching between playlists to create queues, they felt that there was a significant amount of mental effort required that put them off from doing so, even though they mentally felt that they needed to create the queue with a specific order of songs.

3 of the participants also mentioned that they make playlists very infrequently, with 2 of them only making them for each new time period. The other participant remarked that "making playlists [was] more effort than it was worth".

Something that didn't cause friction was the absolute or fixed ordering of the contents of their songs. 3 participants ordered by time or alphabetical order, all remarking that this consistnent order provided "muscle memory" and helped them better remember their collection. 3 of the participants also ordered songs in the collection to elicit a specific listening experience. 1 participant only listened to albums in their original order as they exclusively enjoyed that way of listening to them. The other two participants ordered them so that when the didn't shuffle, they could listen through that experience they set out for themselves.

5.2.3 Listening Queues

There were two modes of creating listening queues for the participants: 5 had a passive mode where they created their queues using the shuffle features and 2 created their queues by manually placing songs in the queue.

4 participants built these queues from an individual playlist and 4 built them from their entire collection. When choosing which playlist to choose from and what song to start listening to, 3 participants said that their choice was affected by a recency bias. They felt that they "had a good chance of forgetting songs that were added to the collection earlier on".

Passive listeners also had issues with the shuffle feature of randomly creating an order. 4 participants felt that they had to fix or guide the queue manually due to the shuffled order creating a listening experience that did not match their expected mental queue.

2 participants mentioned that they would keep skipping songs if it did not match their expectations until they reached one that was "close enough".

2 participants did state that even if the shuffled queue was giving a song that wasn't what they wanted, they would not change it partially due to "not being bothered" and if the playing song was "close enough" to what they were subconsciously expecting.

2 participants also mentioned that whilst they used the shuffle feature frequently, they felt/knew that there were songs that hadn't been played for quite some time as the shuffle simply wasn't playing them. This meant that they weren't fully experiencing their collection.

5.2.3.1 New Song Buffer

3 participants mentioned that when they listen to songs passively and are "less aware of what I'm listening to" that they can forget to add these new songs to their collection. 1 of these participants also mentioned that after adding a new song to their collection they would sometimes remove it on subsequent listens due to realising they didn't actually like it. As such 2 participants mentioned that they would like a user-facing buffer region feature where newly listened to songs could be permanently added/removed after a few listens.

5.2.3.2 Listening as a Trajectory

4 participants mentioned wording that indicated they understood their listening journey to have a direction which was sometimes reflected in the queue. 2 participants mentioned that a factor deciding how they actively mentally change listening direction is when they

are on a boundary region of their collection. "When I listen to this song, it has a sad part that'll make me want to listen to more sad songs instead of the direction the queue is currently heading in".

5.3 Partial Feedback on Unfinished Features

During the interviews participants were also shown the planned design for the dynamic graphs and listening journey features.

All 6 participants noted that they would've liked to see their collection using the dynamic graph feature.

All 4 of the participants who organised their collection using playlists also expressed interest in seeing how the potential clusters formed in the dynamic graph would map to their created playlists.

2 participants also noted that when they're listening they would like to be able to select songs as points to listen around (for queue generation and modification).

Chapter 6

Evaluation

This project aimed to ask two research questions:

- *how can the organisation of digital music collections be improved to better reflect people's mental models of their collections, without increasing the mental load required?*
- *How can we improve the process of creating and controlling*

People have a expected mental audio journey. Passive listeners usually have much larger audio journeys that they'd be happy with

6.1 Active vs Passive Listening Spectrum

When someone is listening to a sequence of songs they exist somewhere on the passive-active spectrum. This spectrum is concerned with the number of possible song sequences that the person is satisfied listening to.

At the extreme end, a fully passive listener is satisfied with any sequence of any songs, no matter how similar or dissimilar the songs in the listening journey are.

At the other extreme end is a fully active listener, someone who has an exact set of songs that they want to listen to, in an exact order. This order may be for many reasons.

All the participants were somewhere on this spectrum when they were listening to their collection, some would stay in one place or would move about on the spectrum. [-] Casper usually passive, but then switches to active to make sure that they listen to album in the correct original order [-] Vedarth passive when doing more mindless activities, but when more focused, preferred to actively create the queue.

The interactions made by users (regarding song queues) can be mapped to relative positions on this spectrum: [-] 100% Active -¿ Manually find song then click add to queue [-] -¿ Shuffle a small playlist / Song radio [-] -¿ Shuffle a large playlist

Keeping track of the next 10 upcoming songs -¿ slightly active Reorder queue -¿ active Switch to new playlists -¿ temporary active then back to passive

6.1.1 What is a Listening Journey?

A listening journey is simply a sequence of songs and as with any sequence, it has the following properties:

- Initial Item (First Song)
- Previous Item (Previously Played Song)
- Current Item (Currently Playing Song)
- Subsequent item (Next Song)
- End Point (Last Song)
- Length (Number of Songs Listened to)

However this sequence is not simply a sequence of scalar items, but should be thought of as a sequence of vectors, items with direction. As such the overall song queue can be thought of being a listening journey, with an overall direction (and a direction from song to song).

This direction is a vector in an n-dimensional space, where n is the total attributes and metadata for a song. This project only looks at a subset of these attributes and metadata -¿ the Echo Nest attributes and the Spotify metadata.

What we propose is that the full listening history of a user is comprised of these listening journeys. However, due to the continuous nature of listening to music in the digital era, further research will have to be done to see if people still have 'end points' or final songs as this was not investigated during the participant study.

A listening journey has 3 parts: its history, its current position, its future:

- **History:** a list of songs ordered by when they were listened to (with how much of each song was listened to)
- **Current Position:** the current position in the currently playing track

- **Future:** the trajectory that the listener will follow, comprised of segments that either have a fixed or unfixed/random order:
  - **Fixed** similar to the history this is a collection of songs which are played through in order
  - **Unfixed** the next song to be played is randomly chosen from a collection of 1 or more songs
  - **Trajectory** this is the n-dimensional vector between the next song and the current song

Both viewing history and the currently playing track are easily viewable and interactable in Spotify (and other streaming services). Spotify does allow for creating and interacting with the future aspect of queues, both fixed and unfixed, however the process to do so still has some friction and can only be done over the entire queue:

- Fixed Order Queue → the user must manually locate and add songs one by one to create a fixed order queue
- Fixed Order Segment → a user can reorder songs in the queue to ensure that those songs are played in a fixed order
- Unfixed Order Queue → a user can click shuffle play to create a new queue of all the songs in a playlist, which have been shuffled to be in a random order. Also they can shuffle the queue once it exists to randomise the order of songs within it.
- Unfixed Order Segment → should a user want to listen to 5 songs in a specific order, then listen to 15 different songs in a random order, then another five songs in a fixed order, there is no built-in way to do this.
  - First the user would have to wait until they've listened through the first 5 fixed order songs.
  - Then they would have to remember the 5 fixed-order songs they want to listen to at the end and remove those songs from the queue
  - They could then shuffle the remaining 15 songs to achieve an unfixed/random order.
  - If they are happy with the shuffle then they can re-add the final 5 fixed-order songs.
  - However, if at any point they want to reshuffle the queue (but only the random-order songs) they have to go through the entire process again.

As can be seen above, whilst Spotify does allow for creating both fixed and unfixed queues, they do not have a way of easily creating unfixed sections of the queue (without losing any desired fixed sections). This project aimed to provide a way for accomplishing this by using the graph visualisations as a base.

Unfortunately, as explained in the Project Retrospective chapter, this feature was not implemented and as such could not be fully tested. However, many participants noted that they would like to see this feature added to Spotify so that they could create song queues with both fixed-order and unfixed-order sections in their queue.

6.1.1.1 Trajectory of a Listening Journey

A listening journey can also be thought of in terms of its trajectory, both between 2 songs, and over multiple songs. These trajectories are comprised of n-dimensional vectors, where n is the sum of all attributes and metadata on a song.

This project aimed at visualising these trajectories by rendering the listening history and upcoming queue as a line in the dynamic graph view, thereby representing all the dimensions of the songs.

Unfortunately, this feature was not completed (as explained in the Project Retrospective Chapter) but when asked to the participants, they all expressed interest in seeing their past and future listening rendered as a directional line. As such this is a concept worth researching further into, to determine how useful it can be, as this was not able to be fully tested during this project.

6.1.2 Song Sources for Building Queues

When passively listening, listeners often have very little mental energy they want to allocate to controlling the music they listen to[Shruthi Quote]. The easiest way to do this is to pick a playlist and start listening within it.

As such, playlists are an effective solution at reproducing listening queues. For some, one large playlist is enough (often Spotify's Liked Songs) as the listener is happy to listen to any possible sequence of songs in that large playlist. However, for others, they do not want to listen to all their songs at once[Vedarth: larger playlists lose their shuffleability], so they decompose their collection into multiple distinct playlists.

These playlists contain songs which all have a common aspect, which forms the identity of the playlist. This can be anything from having a specific artist or genre in common, as well as being for a specific activity like a workout.

However, due to the enclosed nature of the playlists it is difficult to know how much they overlap and what the true full collection looks like unless they are combined and rendered as one.

Unfortunately, combining multiple playlists into one was a feature that was not implemented, but is worth investigating further. Many participants noted that they wanted to see how similar their playlists were to each other. Further research will be needed to analyse the behaviour of choosing what songs go into which song queues[Vedarth agreed with this]. This will be detailed further in the Future Work Section.

6.1.2.1 Difficulties in Maintaining Playlists

At first glance, these playlists seem like the perfect solution for easily recreating queues. However, maintaining these playlists can be difficult for some[Riya: playlists started to converge] and applying clean-up is usually avoided [Riya: can't be bothered to go through and remove songs] as listeners just want to listen to their music usually. They are not usually in the mood to perform spring cleaning on their collection.

There is also a perceived heavy mental load associated with the process of creating playlists and adding songs to them. [Vedarth: likes the design and identity of a playlist and pulling different songs together] This puts off user's from creating new playlists with new identities with songs from their collection.[Shruthi+Vedarth would like maybe 2010's playlists but cba to make it]

There is currently too much friction associated with creating playlists, even though there is a desire.

Automatic Generation of Playlists One possible solution that was proposed in this project, but not attempted due to being low priority was song tags. These are words or phrases that can be attached to any song (similar to the genre metadata) and were a core feature of Apple's iTunes. These song tags can then be used to automatically generate playlists for that tag, taking away the tedious labour from a user. This will be explained futher in the Future Work Section.

Buffer Zone Another issue with utilising playlists effectively is adding the right songs when they're found by the user and removing ones that don't belong anymore. Most of the participants didn't remove songs, but that was also because they were more certain that a song belong to the playlist. For the participant who was less certain if a song belonged then they would add it and remove it afterwards if they deemed it necessary. A few participants also noted that they didn't always remember to add songs that they liked listening to, so they would prefer to

6.2 Design Improvements

6.2.1 Visualising using Continuous Attributes in a Collection of Songs

The Echo Nest attributes were useful in helping provide more ways for listeners to understand their music collection. Each combination of attributes and songs has a specific visualisation that was found to be the best by the users.

Note: in the below table, Each Song is referring to how a song compares to the rest of the songs in the collection it belongs to.

Table 6.1: Optimal views for 1 or more songs and 1 or more continuous attributes

<table>
<tr><th></th><th>Singular Song</th><th>Each Song</th><th>Overall Distribution</th><th>Extremities</th></tr>
<tr><td>1 Attribute</td><td rowspan="3">Table</td><td>Table/1D Graph</td><td>Histogram</td><td>Table</td></tr>
<tr><td>2 Attributes</td><td colspan="3">2D Static Graphs</td></tr>
<tr><td>3 Attributes</td><td colspan="3">3D Static Graphs</td></tr>
<tr><td>>3 Attributes</td><td>Polar Chart/Table/
Line over Ridge Plot</td><td>Dynamic Graph</td><td>Ridge Plot</td><td>Table/
Dynamic Graph</td></tr>
</table>

6.2.1.1 Understanding a single song using any and all attributes and metadata

No novel visualisations were required. Of note, some participants[Riya,] liked the polar chart as a way of rendering all the continuous attributes.

6.2.1.2 Understanding multiple songs according to 1 attribute

Conventional methods sufficed: tables, histograms, bar charts, etc.

6.2.1.3 Understanding multiple songs according to 2 continuous attributes

The conventional method of 2-dimensional cartesian graphs were sufficient for the participants. It allowed for viewing the collection as a whole and for also looking at and comparing individual songs. Finding outliers and max and minimum points was a trivial process in this view.

6.2.1.4 Understanding multiple songs according to 3 continuous attributes

The 3-dimensional cartesian graph was useful here but harder to navigate and represent on a 2D desktop screen. This would be easier to interact with using a 3D medium, such as AR or VR.

6.2.1.5 Understanding multiple songs according to more than 3 attributes

The conventional cartesian graphs do not hold up anymore. Although it was not developed in time to be fully evaluated, the design of the dynamic graph was evaluated by the participants. Their feedback indicated that this view is an effective method of better understanding their collection using more than 3 attributes. 2 participants in particular were particularly excited as they felt it better matched their mental model of their music collection and the way they listen.

Whilst the dynamic graph is good at visualising multiple songs using multiple attributes, it collapses the attributes in order to do so. This makes it harder to distinguish how each attribute is affecting the collection. To see how the collection as a whole can be represented by the attributes, the best visualisation is a ridge plot → histograms of each continuous attribute layered over each other. This feature was designed but not implemented, so the design was evaluated, but requires further research.

The participants all found the ridge plot as a significantly useful feature. 1 participant[Casper] noticed that it was very effective as mini-map of the whole collection. When combine with the range sliders for filtering, this feature had more application. 1 participant[Casper] stated that they would prefer to use this filterable ridge plot to filter out the table and graph views over the conventional filtering method.

6.2.2 Attribute Combinations: Overwhelming Choice

All the participants enjoyed seeing their collections according to different combinations of attributes. However, with there being 12 attributes in total that could be applied to the axis there was a significant amount of combinations to go through:

- 66 combinations over 2 axes
- 220 combinations over 3 axes

This is such high number of combinations that most people would not bother trying to go through all of them. The patricipants only went through 3 or 4. To assist with understanding and using the static cartesian graphs, 2 participants noted that they would like suggested combinations to start with. This is detailed more in the Future Work section.

6.2.3 The Echo Nest Attributes

Users enjoyed being able to see their collection defined in terms of the Echo Nest attributes as some data points affirmed pre-existing thoughts about their songs.

It was also an effective way of defining one's music tastes.[who?]

However, not all the attributes were useful to everyone, most notably key, mode and time signature. Further testing on a higher sample size would be required to fully test which attributes are more relevant over others.

The most influential dimension turned out to be time as this was easiest to interpret from the user's side as noted by one participant[Vedarth quote]. [Casper] also noted that a sped up line of their audio history would be cool, also implying that time was a instinctively useful attribute

The attributes also required some explanation as they were not all self-explanatory. As such for these to be introduced in a more user-facing role and to be rolled out to the public, a walkthrough would need to be created, going through each attribute and giving a brief understandable description.

6.2.4 Distinguishing Songs in the Graph Views

The static graph views were highly effective at understanding one's entire music collection and learning the different landscapes of their collection. However, a major trade-off with the graph views (including dynamic) is the loss of individual song identifiability.

To ensure the individual songs were still identifiable in the graph views, the following options were queried with the participants to receive feedback:

- **Title Text:** Most recognisable and interpretable. Usually not enough space to full render however.
- **Album Image Square:** Less recognisable than song title, but still recognisable. Requires much less space to fully render.
- **Coloured Sphere:** Least recognisable but requires the least amount of space to render.

The approach taken in the project was to render the songs as 3D spheres, with planned functionality to colour them based on a discrete attribute or metadata. 5 participants[no Josh] expressed that they would want the colour to be determined by genre and 3 participants[] stated they would want to colour by the mood/vibe of the song.

One participant noted that they would want to colour by artist, however this is likely due to the fact that they build their collection around a specific set of a few artists.

The right method of distinguishing songs can also have a secondary bonus effect, allowing for one to roughly see the distribution of a discrete attribute/metadata on their collection.

5 participants[not Riya] said they would prefer album images over 3D spheres as this method hit the right balance of not taking up too much visual space (resulting in overlap and obscurity) and still alowing for song recognition. However, they all agreed that they did not mind the song spheres as well and would only prefer the album images to render when zoomed in enough to be legibile.

6.2.5 Graph-Based Suggestions

Currently in the Spotify app, suggestions are just limited to a small list of five after the end of a playlist. 3 participants[] noted that they would heavily appreciate seeing suggested songs rendered in their collection, so they can see how it fits into their existing landscape.

6.2.6 Graph Navigation Controls

All the participants felt that the controls for the 2D graph were satisfactory, however some[] noted that they would've also preferred an alternate set of controls for the 3D graphs.

Instead of the implemented controls using the mouse and having the graph as the center of the 3D scene, a more first person videogame-esque control system was suggested. This would involve using the keyboard to move the camera in all directions to better explore the 3D space.

Chapter 7

Project Retrospective

This chapter will go over the significant limitations and an improved ideal project plan.

7.1 Handling Complexity: Miro Mindmap

One of the major elements that significantly helped with planning the project and managing the complexity of the project was building a hierarchical mindmap in Miro (a highly flexible diagramming tool). The reason Miro's mindmap was useful was that it allowed for children nodes to be toggled, meaning that they were visually hidden, but still accessible.

This feature significantly helped in breaking down the mammoth complexity of the entire project. The project was decomposed into the high-level parts, getting more specific as the tree became deeper. These high-level parts can be seen in figure 7.1. Expanded sections of this diagram can be found in appendix .3.

The only issue with this diagram is the point at which it was used. This diagram was only built at the end of January meaning it was not helpful for the research and design stages, where it would have helped significantly.

The only drawback with this method is that whilst being able to hide child nodes is very useful, the count of hidden nodes is not that useful a metric. As such it was possible to hide nodes and forget about the contents which slightly hindered development.

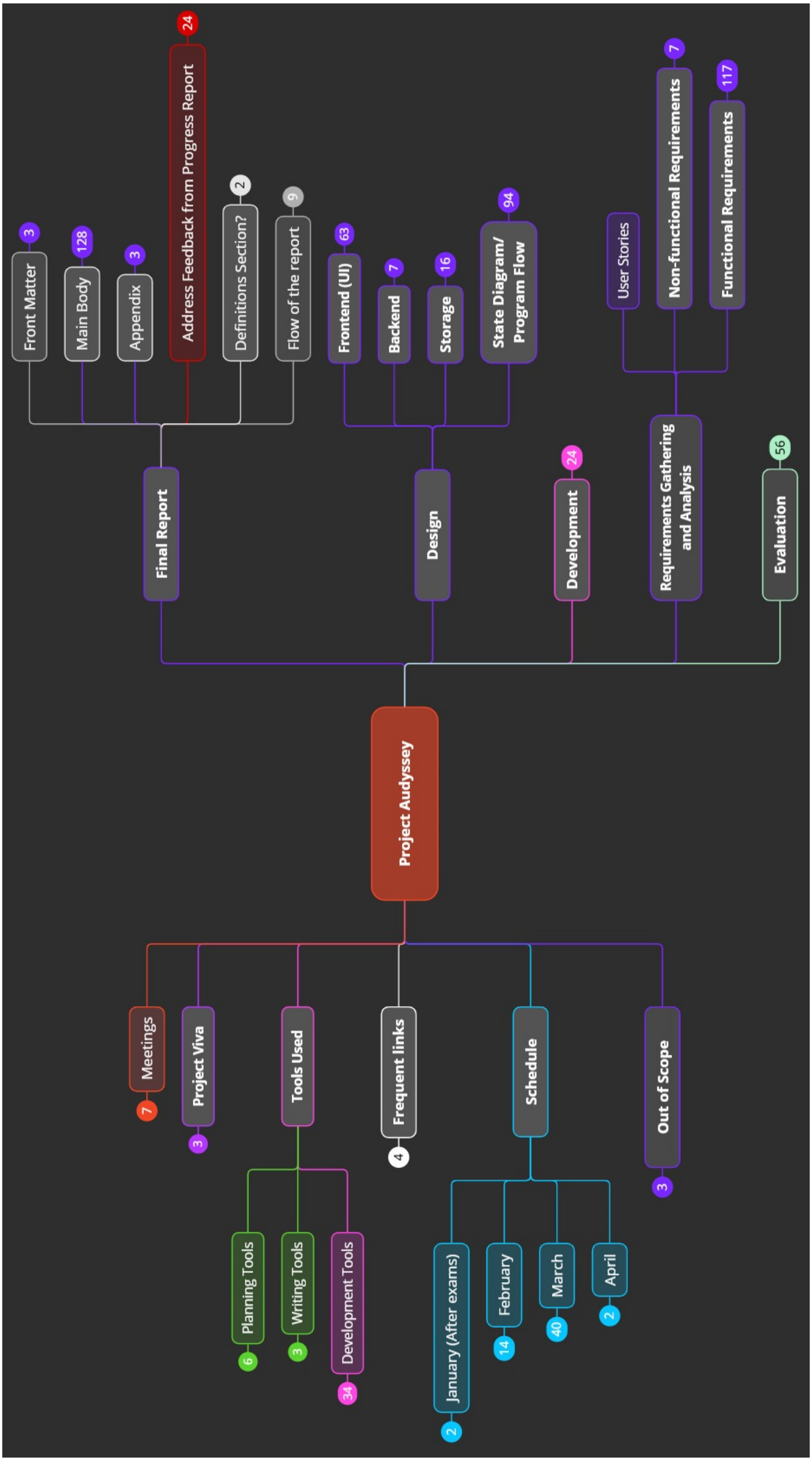

Figure 7.1: Miro Mindmap high level overview

7.2 Limitations

7.2.1 Unfinished Code

One of the key limitations of this project resides in the fact that the dynamic graph and audio journey features were not developed in time. This meant that these features were not able to be fully evaluated and therefore will require further research to truly understand them.

This is partly due to the unnecessary complexity in obtaining the Echo Nest attributes. The initial process to gain these attributes was through these endpoints via Spotify's API: `Get Track's Audio Features` and `Get Track's Audio Analysis`. However, as stated in this blog post, these endpoints were deprecated.

As an alternative, I switched to using SoundCharts API to access the attributes. However, this API is paywalled and less precise but would still provide sufficient values for the graph views. Another issue with this workflow is that the complexity used up a lot of valuable development time. Had Exportify been found earlier, the project development phase would've been significantly smoother.

7.2.1.1 Missed Opportunity to Reprioritise

Whilst the SoundCharts workflow was unnecessarily complex, the other big issue is that the requirements should've been reprioritised. Whilst the static graphs provided a lot of insights as detailed in the evaluation, they were not the core focus of this project. Due to its novel nature, the dynamic graphs were more important and should've been re-prioritised once I realised that I would not have enough time to complete both graph views to a satisfactory standard.

The true/original niche/gap in the research was investigating better ways to create listening journeys, mainly using the graph visualisations as a foundation for controlling and viewing them. This is because, with the advent of the digital streaming era, there has not been any research for creating better tools for users. As discussed in the Evaluation, there is an expressed desire for these tools.

However, during development, after the static graph views were created, I realised that these were only really specifically useful in a data analysis perspective (something that has already been researched extensively (as mentioned in the Background chapter) and is not the focus of this project). The evaluation also confirmed this, as the participants felt that the static/cartesian graphs were only useful as a one-off and not as a basis for reflecting their mental model of their music collection.

In hindsight, developing the staic graph views should've been allocated to be done after the dynamic views so that the listening journeys could be accomplished as fast as possible. Static cartesian graphs were intially planned first as I did not realise at the time that they were significantly less useful for interacting with listening journeys than the dynamic view.

One participant in the evaluatory study also mentioned that they liked there being an absolute order to their songs/collection that they could return to. This absolute order synergises better with the dynamic graph, as the static graph produced different distributions for each combination of attributes on the axes.

7.2.1.2 Prototype Review Meeting Not Held

A side-effect of the unfinished codebase is that there was not enough time to review the application before the evaluation. Whilst unfortunate this did not hold the project back too much as the software-related feedback was still obtained during the participant study (such as how to make the songs in the graphs easier to identify). However, more critical feedback would be ascertainable had the review meeting occurred.

7.2.2 Evaluation: Small Participant Count

Only 6 out of 10 participants were able to be evaluated in the participant study. Whilst these participants were varied in their listening and organisational styles and were sufficient for the evaluation, more would've been better to ensure the thematic analysis performed was as extensive as possible.

7.2.3 Evaluation: Lengthy Interviews

Another drawback of the evaluation interviews is that they were very lengthy, taking 1hr on average. Whilst this in of itself is not an issue, considering two of the big features were not fully developed, the intended interviews would be even longer. As such, the interview should've been broken down into two parts:

- Understanding Participants' Music Organisation and Listening Style/Behaviours
- How the Audyssey affected their Music Organisation and Listening Behaviours

This first stage should've been done either before or after the design stage as it would also help inform the functional requirements for the system.

7.2.4 Difficulties in Sourcing the Echo Nest Attributes

As explained in the Background Chapter, the Echo Nest is a compnay that created a database of rich attribute data gained from high-quality analysis of song audio files. However, gaining access to these attributes has become increasingly difficult over the years:

7.2.4.1 2014: Spotify buys EchoNest

EchoNest's extensive and free API is now locked behind a Spotify Premium Account. Whilst not a good event for the widespread community, this was not an issue for the project as the Spotify API was already a planned part of the project.

7.2.4.2 Spotify Deprecates Key Endpoints

27 Nov 2024 Spotify announces they are deprecating several endpoints for applications made after the 27th November - these endpoints included both `Get Track's Audio Features` and `Get Track's Audio Analysis` which were key for the project.

These audio features (or attributes) were critical for the project as they were the basis for graphing the songs and helping control the audio journeys.

To ensure that this didn't derail the project, an alternative was quickly found: the SoundCharts API. This API had an endpoint, that given a track's Spotify ID, would provide the attribute data that Spotify had deprecated access to.

However, this data was less accurate (only to 2 decimal places) and was also behind a paywall. For one month, the cost of access for 500,000 API calls at a 30% academic discount amounted to 125 USD. This was within the budget of the project. Initially, the plan was to purchase one month once development had finished. As such the API access would be used only when it was fully needed.

Late March 2025 Due to significant issues with purchasing the API access using the University's system, another alternative method to gaining the attributes had to be found. This solution was found in Exportify, a web tool built by Pavel Komarov. This tool accesses the Spotify API, including the newly deprecated endpoints, to allow for exporting a spotify user's library to a `.csv` file.

This tool was made before the deprecation announcement and is free to use, making it a very suitable replacement. Futhermore, the attribute values are to the original precision as provided by Spotify. The tool also allowed for exporting of individual playlists, making it easier for my software application to know how one's full collection was composed by the playlists and the catch-all liked songs.

7.2.4.3 Accuracy

Due to not having direct access to the full audio analysis of a user's songs, I was not able to use the confidence intervals that exist for each attribute. 2 participants mentioned that they disagreed with some of the attribute values for certain songs. Without access to the confidence interbals, there was no way to know if this was true. I effectively had to assume that all the attribute values were fully accurate.

7.2.5 Scope Creep: Implementing Better Listening

The goals for this project can be separated into two unique (but heavily linked) parts: improving large-scale music organisation and improving ways to control listening to music libraries.

As can be seen in the project brief the better listening was the initial focus, but during the research stage this pivoted to improving the organisation of large music libraries. When this pivot was made, improving better listening should've been re-classified as out of scope to ensure the project expectations remained manageable within the allotted timeframe.

However, improving better listening is a core aspect that affects how people organise their music libraries as found in the evaluation.

7.3 Initial vs Final vs Ideal Project Plan

Below are three project plans:

- **Initial Plan** → the planned progress of the project (created before the development stage)
- **Actual Progress**
- **Ideal Plan** → upon retrospection, this is how the project should be approached if to be done again

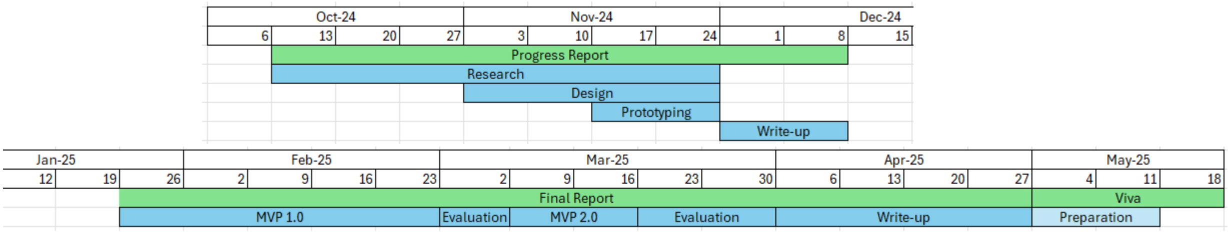

Figure 7.2: Gantt Charts of Initial Plan

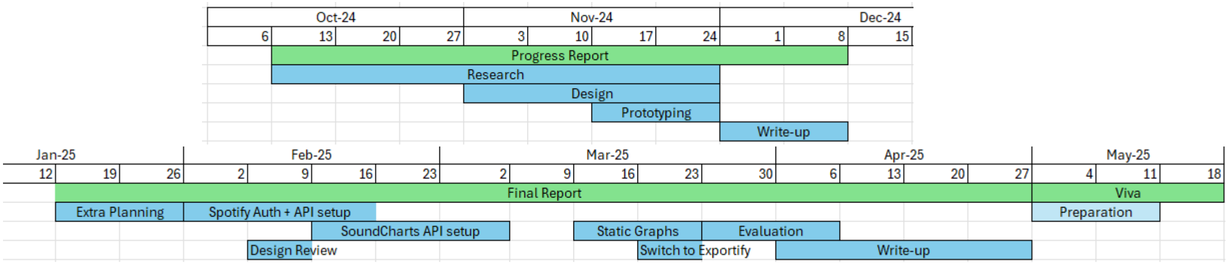

Figure 7.3: Gantt Charts of Actual Progress

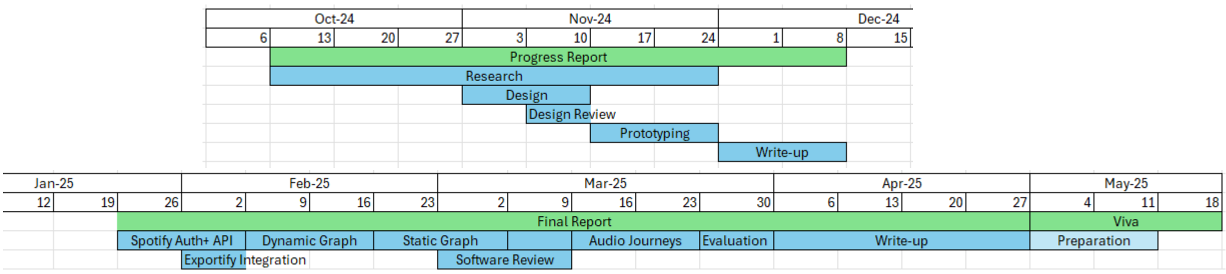

Figure 7.4: Gantt Charts of Ideal Progress

Chapter 8

Conclusion

To help manage the increasingly large scale of digital song libraries this project investigated both static cartesian graphs and a dynamic similarity graph as potential alleviators. Whilst the static graphs were useful in providing insights about one's collection, the dynamic graphs proved to have more potential as a method of organising a music collection. Whilst unimplemented, the dynamic similarity graph did recieve heavy interest and even had heavy overlap in a participant's pre-existing mental model of their collection.

The less rigid nature of dynamic graph adds to this as users felt mentally taxed by having to explore and understand the different possible static graphs. The dynamic graph also served as a better foundation for visualising and controlling listening queues, something that requires further research due to time constraints within the project.

There is a spectrum of control that exists when people listen to their collection. This ranges from passive (hands-off uncontrolled) to active (manually choosing the next song). Listeners can move around on this spectrum depending on how Spotify's method of building a listening queue matches the mental expectation. When there is a misalignment and more control is required this can be mentally taxing on non-active listeners, who don't want to allocate mental resources to control their listening.

There is also a spectrum of that can be found in the ordering of the song queues, from fully fixed (known absolute order of songs) to unfixed (unknown order of songs). Spotify (and other streaming services) achieve the unfixed order by utilising the random shuffle feature on a listener's queue. However, there are cases when a listener wants to only listen randomly to a section of their queue and then switch to a more fixed section. There is no software solution to this that has been implemented or researched, but there is heavy interest from listeners for this feature.

8.1 Future Work

8.1.1 Buffer Zone for New and Suggested Songs

When a song that isn't already in a user's library is listened to, this song will be added to the collection, but only partially. These songs should have a distinguishing visual identifier that shows that they aren't fully part of the collection yet.

To fully add these songs, they can either be automatically added after a certain number of listens within a timeframe (and likewise possibly removed if they don't reach this number) or also manually added/disliked by the user.

This buffer zone could also act as a way to introduce song suggestions from Spotify or also friends of the user.

This feature would also require research into ensuring that the user does not get overwhelmed by a large number of songs in this buffer zone.

8.1.2 Customisable Presets: Attribute Combinations

This feature would help to handle the significantly large number ofthe possible combinations of attributes for the static cartesian graphs. Users would be presented with suggested attribute combinations to get started exploring. They could then configure these presets to tailor the static graphs that are most meaningful to them.

Further research would be required to figure out the best default combinations to show as this would require ranking all possible combinations using a very large and varied data set. These may also vary depending on the distributions of a user's collection of songs they are viewing.

8.1.3 Storing and Viewing Listening Histories

Although listening histories were initially scoped into the project, there is a lot of hidden complexity in storing sections of a user's full listening history. Then being able to create a mechanism to easily re-listen to these sections of the full listening history (but in a different order) is another complex challenge that requires further research and investigation.

The main complexity resides in the datatypes used to efficiently store sections of the full listening history: whether to use an array, a linked list, a sub-graph or a more complex datatype.

8.1.4 Automatic Creation and Management of Playlists

This feature would allow for playlists to be automatically created based on certain ranges of attributes. This would effectively allow for overlapping subsets of the dynamic graphs, so users can easily look at specific parts of their collection without having to go through the mentally taxing process of creating a playlist themself.

These defined regions could also be created using custom user tags, explained below.

8.1.5 User-added Song Tags

These are strings that can be added to songs, similar to how Explicit functions in Spotify. These tags are usually implemented with a flat nature, that is they are all distinct from each other.

This implementation of song tags is a very popular feature of iTunes. However, there isn't research on using these custom tags as a method of helping to manipulte the structure of the dynamic graph. Users can add tags to account for missing information or ways to introduce similarity that is personal to them.

One issue with adding song tags is that this can be quite a mentally taxing activity. As such a potential avenue for research would be to adapt the Apple Photos approach where photos are automatically classified using machine learning. Users would then be able to modify and configure these tags, meaning most of the grunt work is done for them.

These tags would also require investigation into the benefits and tradeoffs of hierarchical tags (similar to Spotify's genres) where each tag can have a set of sub-tags.

8.1.6 Other Mediums

The Audyssey was only designed and developed for desktop, however most people use Spotify on their phones [**?**]. As such more research is required to figure out how to best translate the static and dynamic graph views to be accessible on mobile.

Further research also needs to be done in seeing how well the graph views perform in mixed reality, where the 3D graphs are easier to navigate and explore.

Bibliography

[1] E. Topics, “Music streaming stats,” 2025. [Online]. Available: https://explodingtopics.com/blog/music-streaming-stats

[2] A. N. Hagen, “The playlist experience: Personal playlists in music streaming services,” *Popular Music and Society*, vol. 38, pp. 625 – 645, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:193242204

[3] W. T. Glaser, T. B. Westergren, J. P. Stearns, and J. M. Kraft, “Consumer item matching method and system,” patentus 7 003 515B1. [Online]. Available: https://patents.google.com/patent/US7003515

[4] A. Saravanou, F. Tomasi, R. Mehrotra, and M. Lalmas, “Multi-task learning of graph-based inductive representations of music content,” in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*. ISMIR, Nov. 2021, pp. 602–609. [Online]. Available: https://doi.org/10.5281/zenodo.5624379

[5] J. Constine, “Inside the spotify - echo nest skunkworks,” 2014. [Online]. Available: https://techcrunch.com/2014/10/19/the-sonic-mad-scientists/

[6] E. Pampalk and M. Goto, “Musicsun: A new approach to artist recommendation.” 01 2007, pp. 101–104.

[7] S. Cherubin, “A novel music browsing framework: The ambif project,” Ph.D. dissertation, Scuola di Ingegneria Industriale e dell’Informazione, 2013/2014. [Online]. Available: https://hdl.handle.net/10589/106630

[8] E. Joy, V. Braun, and V. Clarke, “Doing reflexive thematic analysis: A reflexive account,” 2023, wrote to publisher re AAM Tue 07/02/2023 12:12 Publisher granted permission on 08/02/2023. [Online]. Available: https://uwe-repository.worktribe.com/output/10434562

Project Brief

.1 Problem

Digital music streaming services (Spotify, Apple Music, Tidal, etc.) provide an easy way to listen to songs and playlists, from users and artists alike.

These services allow users to listen to multiple songs by traversing through a list/queue. This list can either be manually ordered (playlists), semi-ordered (through the radio feature) or fully disordered (random shuffling). The latter two methods exist to ensure that the listening experience doesn't become stale, i.e. the user doesn't have to listen to the same songs in the exact same sequence.

These three methods all have drawbacks however. Playlists, while effective and fun to make, can be a lot of overhead for the desired effect. Playlists are quite static and don't easily evolve with the user. Random shuffling is very effective at keeping the listening experience fresh and new. However there is no way to control this random jumping which can lead to both smooth and jarring sequences of songs. The radio feature, which allows for listening to songs which are deemed similar to a specific song is more effective than the prior methods, however it only allows for song sequences based on a specific song and not based on other musical dimensions.

These services also don't provide a method to transition smoothly between songs, as currently, the end of a song leads immediately into the next. To create better transitions, specific DJ software/experience is required which can be a lot of overhead for a simple effect.

.2 Goal

This project will consist of a web-based application allowing for advanced visualisations of a user's song library. The predominant view will be a 3D graph view where songs are positioned based on their dimensions, such as genre, instrument, artist. To avoid clutter, the graph will be able to hide the links belonging to their respective dimensions.

To listen to sequences of songs, users will mainly use coordinates in the multidimensional space. These coordinates can refer to one, many or all dimensions (all being a song). These coordinates can then be used as milestones, heuristics which decide the next song to listen to. Eventually, after a milestone is reached, if no new milestone is set, then the next similar song will be played and so on.

Traversing between milestones can be done smoothly or roughly, depending on tweakable parameters. Milestones can also be used an attractor, for the next song to be chosen from within a certain radius of the milestone.

This project will focus on enriching each song's metadata with its values in their respective dimensions. As such movement between songs has the ability to be much smoother. If the gap between the end of the current song and the start of the next song, a low-code/no-code (similar to Scratch, node editors, etc.) system can be leveraged to bridge the gap. This system would allow for simple transitions to be chained.

Enriching of song's metadata can be done automatically (potentially via the Spotify API) or manually, by the users/artists. User's can also create additional tags to enforce simple rules on the traversal system.

.3 Scope

This project will only use songs from a user's Spotify library (via their API) as this will be sufficient to test the new multidimensional navigation system.

This project could also let the user upload audio clips that aren't necessarily songs, but this is currently out of scope and will only be attempted if time allows for it.

This project will not involve creating a stem separation algorithm for further decomposing songs but a pre-existing system (such as Gaudio Studio) will be considered if they are worth the time (and possible financial) investment.

To make bridging the gap between songs even smoother, sections of other songs can be used, however this isn't one of the main priorities but will be considered.

Miro Diagrams Expanded

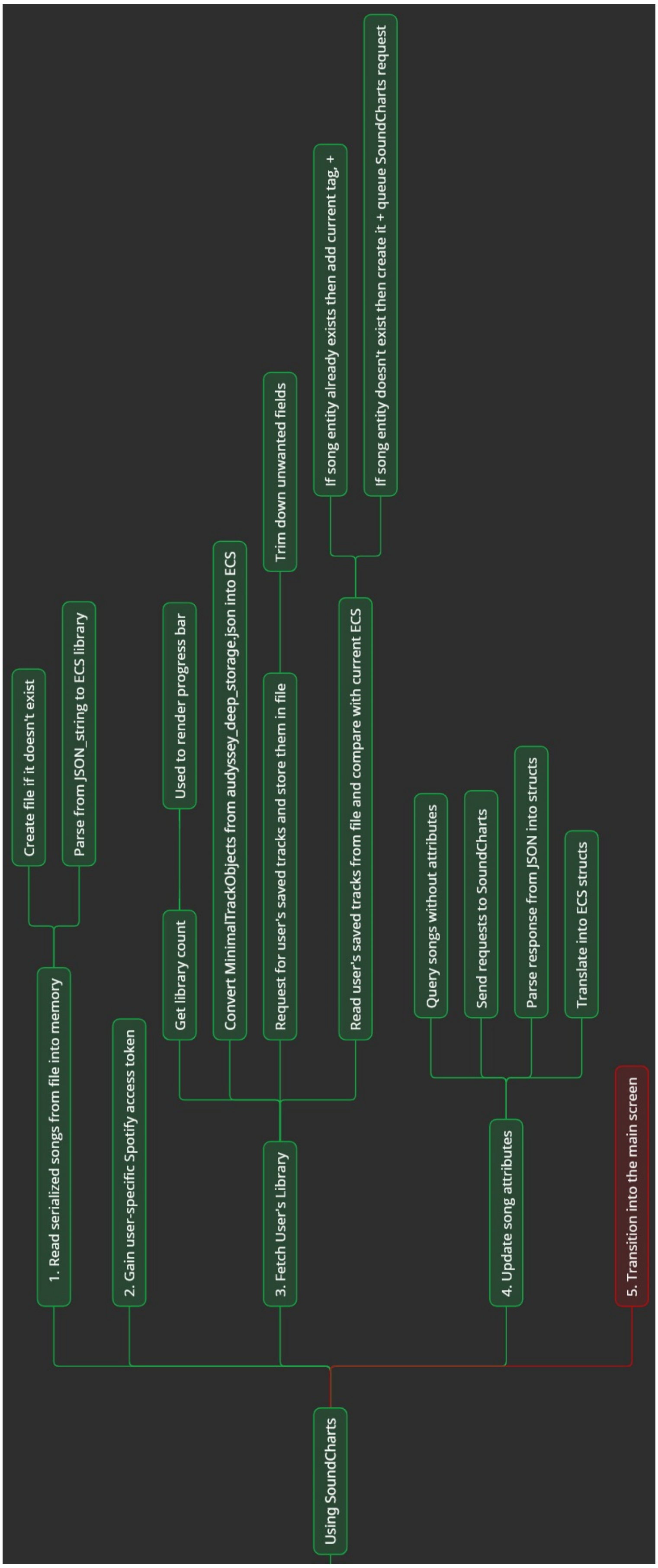

Figure 1: SoundCharts Expanded State Flow for filling user's songs with attributes

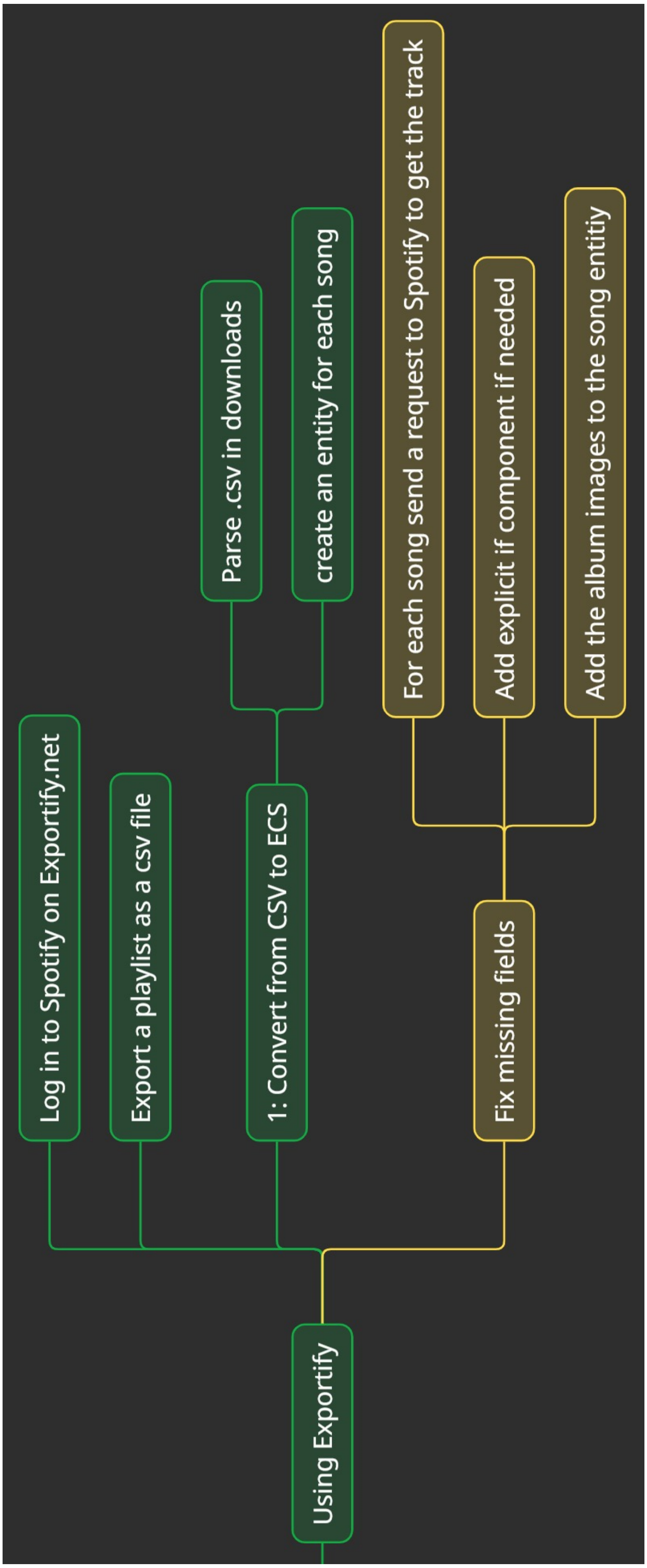

Figure 2: Exportify Expanded State Flow for getting all songs (with attributes) in a specific playlist

Risk Analysis

Risk	Severity	Likelihood	Risk Exposure	Mitigation Strategy
Spotify API deprecates Web Playback SDK	3	5	15	Switch to playing songs locally, accounting for surge in time required
Spotify API deprecates core API features being used	3	5	15	Switch to an alternative API, such as Soundcharts to get the same data.
Soundcharts API free limit of 500 calls is exceeded	4	2	6	A one month subscription for 500k calls is 140 USD (after a 30% academic discount) the majority of which is payable under the comp3200 budget
Soundcharts deprecates core API features being used	1	5	5	Switch to an alternative dataset/API for the required data. Should none exist or are suitable enough, then implement open-source algorithms to gain the data myself.
Falling ill, preventing progress	2	2	4	Make sure to stay physically healthy and always have medicine to hand for a speedy recovery
Failing to meet self-made deadlines	2	3	6	Set realistic, achievable deadlines. If needed, reorder work accordingly such that the quality of work is not compromised
Loss of communication with supervisor	1	3	3	Plan meetings in advance, make sure to have frequent updates with supervisor so that we are both aware of the current situation
Third-party tools (Notion, Overleaf, VScode) going down temporarily	2	4	12	Local backups will be made periodically to ensure progress isn't fully lost.
Third-party tools become unusable for extended period of time	1	5	5	Use alternative tools, respectively: Obsidian/Notepad, Notepad/VScode, Notepad. Notepad is chosen as the final fall-back as it is the most reliable.
Code being lost/-corrupted	3	5	15	Make frequent backups, also using code versioning to allow for safe rollbacks.
Loss of (progress) report	2	5	10	As above, make frequent backups.
Loss of key API credentials	1	4	4	Store API credential info in multiple separate places, and if needed request new keys.

Table 1: Risk Assessment Table