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

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.

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 (980 words)

1.1 Problem

As technology has advanced over the years, people's personal music collections have become more and more digital [**?**].

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"[1].

This hierarchical system allows for the better handling of large scale collections that grow at the pace most collections do [**?**] but if these playlists grow too large, then they require reorganising which can become a chore.

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 Graph** → 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 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

TODO: Background (360 words)

Here we will have a systematic literature review of the prior approaches to this problem. [-] explain the difference between the two types of graphs I will use [-] Data visualisation Cartesian graphs with axis [-] Network Graph Theory [-] Analysing songs to get attributes that correspond to music Theory [-] Pandora's Music Genome Project [-] The Echo Nest Attributes [-] Data analysis using this [-] Previous attempts at Visualising Music Resources Spatially

2.1 Spatial Visualisation of Music Collections

2.2 Song Similarity through Music Information Retrieval

There has been lots of research into analysing song similarity by performing analysis on the audio itself. This leads to extractable features that can then be used to compare songs to find their **sonic distance**. Analysis over the whole song leads to high-level features, including: key, chords, tempo, rhythm, genre, lyrics, etc.

Music Information Retrieval (MIR) is the field of study concerned with extracting musically semantic information from audio files, primarily through the use of machine learning algorithms. One of the biggest successors at this was the Echo Nest, a company who analysed millions of songs to produce low-, mid- and high-level features, made public through their API.

In 2014, the Echo Nest was bought over by Spotify, meaning that their API was now integrated into Spotify's developer API.

There is lots of research and projects done using these Echo Nest attributes to perform data analysis on Spoitfy users' music collections, as explored below.

However, there has been very little research into using these high-level features (attributes) as the foundation for different ways of representing personal digital music collections, aimed at helping manage the growing complexity.

The attributes have been used as the basis for providing song suggestions to users, based on how they listen.

There has been explorations into different ways of visually representing users music collections: [-] **ambif** - used their own feature analysis, closest thing to the Audyssey [-] built in a game engine [-] used 3D static cartesian graphs [-] **Organise Your Music** - uses Echo Nest attributes to create 2D graphs, mainly for easy playlist generation by drawing a line [-] 2D static cartesian graphs [-]

2.2.1 TODO: Pandora's Music Genome Project

2.2.2 The Echo Nest Attributes

These attributes are high-level features analysed from the audio files of song tracks by the Echo Nest group. These attributes and their confidence intervals (how accurate the values for the attributes were)

2.2.2.1 Applications

TODO Exportify's Python Notebooks

TODO Chosic

Chapter 3

Design/Method (850 words)

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

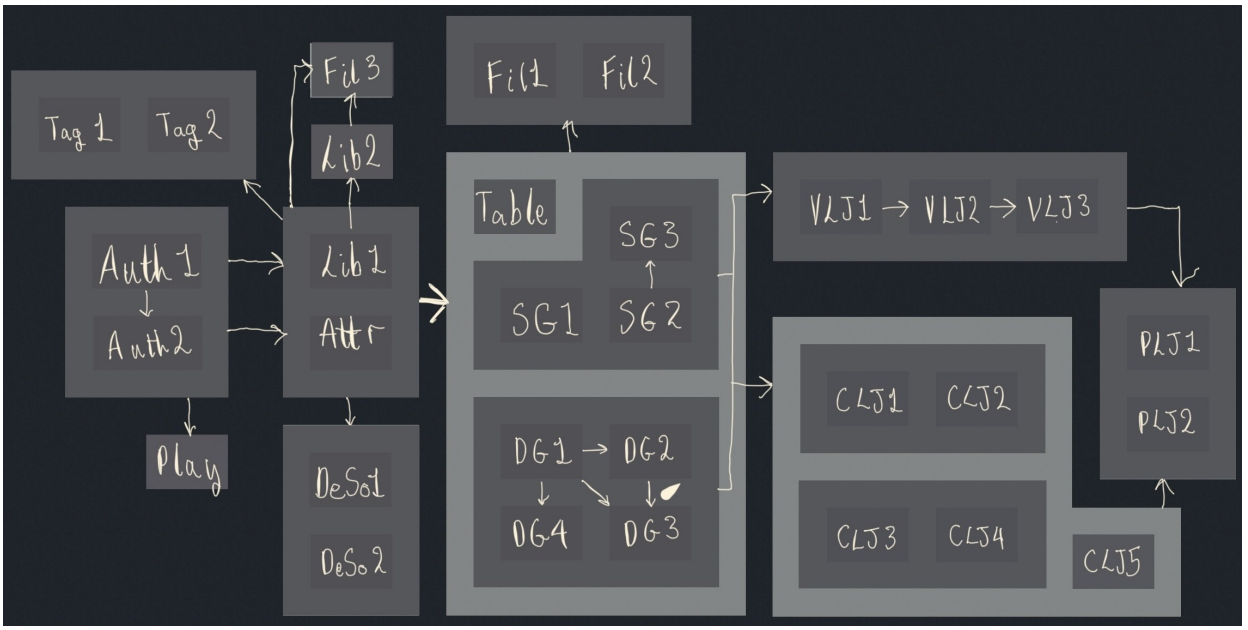

Figure 3.1: Activity Network Diagram of Functional Dependencies

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 TODO: Static Cartesian Graphs

[-] combine the different continuous attributes

3.3.1 TODO: UI mockups

3.4 TODO: Dynamic Similarity Graph

[-] Each song is a node in the graph [-] Each song has multiple bidirectional weighted edges to each other song [-] Discrete attributes/metadata: edge is overlap of discrete metric between song A and song B [-] Continuous attributes/metadata: weight of edge is the 1-x, where x is a the difference between the values of Song A's acousticness and Song B's acousticness (for example) [-] When rendering the graph, songs are pulled closer together depending on strength of weight: [-] continuous weight of 100% - songs pulled as close together as possible [?] maybe have a toggle to turn off collisions? that way clusters that form might be more interesting or the graph never settles on a shape

3.4.1 TODO: UI mockup

3.5 TODO: Audio Journeys

[-] Storing in Data: [-] Specific relationship for full audio history: 'History -listened-to-time-¿ Song' [-] can also add how much of the song was listened to but this is out of scope, but easily addable to the relationship [-] viewing Audio Journeys [-] Table/List view: sort by time, (what else?) [-] Graph: [-] new range slider that is the listened to time (not sure if this replaces the added at time slider) [-] just show the fully history as a static line with arrows to indicate direction [-] further research to find out when it stops being readable/legible (how many songs/ time period)

3.5.1 TODO: UI mockups

3.6 TODO: Storyboards

Chapter 4

Development (1850 words)

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 TODO: Architecture

Diagram of the way I used the tauri API to effectively just access the flecs backend. Although the way I sent data and received data was a bit clunky. Should I have made a trait that allowed things to work together well?

[-] someone interacts with the react frontend, ts function calls `invoke("rust_function")` by using tauri API [-] tauri finds the unique function with name `rust_function` and runs that function [-] if that function needs to access the songs/collection a flecs query is created and run. [-] data is then sent back to the frontend via listening for events or as the callback to the frontend function that did the invoking

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

4.4.1.1 TODO EntRel Diagram

Flecs allows for adding `Tag` components to entities, that is a component where there is no data associated with it. These tags were used to define the different archetypes that exist in the application.

Note that the `MissingAttributes` component is deprecated as it was only required to the SoundCharts workflow.

The only entity where there is one of is the listening history.

4.4.2 (TODO add attribute definitions) Echo Nest Attributes & Spotify Metadata

Table 4.1: The Echo Nest Attributes

Attribute	Definition	Datatype	Possible Values	Continuous /Discrete
Acousticness		`float`	`0-1`	Continuous
Danceability		`float`	`0-1`	
Energy		`float`	`0-1`	
Instrumentalness		`float`	`0-1`	
Liveness		`float`	`0-1`	
Loudness		`float`	`-60-0`	
Speechiness		`float`	`0-1`	
Valence		`float`	`0-1`	
Tempo		`float`	$\geq$`0`	
Key		`integer`	`None/C/C#/D/D#/E/F/F#/G/G#/A/A#/B`	Discrete
Mode		`boolean`	`true/false`	
Time Signature		`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 TODO: Application State Flow (+ Diagram)

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 1, 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 1, 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 TODO: Main application yippee

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

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.

4.5.3.2 TODO: Static Graphs

TODO: three.js Instanced Points

TODO: Draggable Range Sliders

TODO: Detailed Song Component

4.5.3.3 TODO: Dynamic Graph

To create the dynamic graph, a special set of components are used. - Each discrete metric is queried, then all songs found in this query have a `same_metric` relationship added. - Each continuous metric is queried then each song loops through the other songs - calculate difference between metric values - divide this difference by the maximum possible difference to get a percentage similarity - add Component: —SimAcousticness(50%)–¿ (for example)

- There was not enough time to fully pseudocode how to actually render the graph and still being readable.

4.6 TODO: 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 TODO: Reducing Scope

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 (1800 Words)

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 open approach was required.

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 aformentioned `.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 [**?**]

5.2 (TODO Add quotes + who said what + fix longtable)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
		Creating Desired Playlists	2

Friction/High Mental Load

Theme	Sub-Theme	Code	Count
		Switching between Playlists	2
		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 [Riya], 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[Shruthi], a time period[Shruthi, Vedarth], a genre[Josh, Vedarth], a mood or vibe[Riya, Josh, Vedarth] or an activity like the gym[Vedarth, Casper wanted one].

Many of the participants however noted common cases where they felt friction in interacting with their playlists. 2 participants[Shruthi, Vedarth] 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 mentall felt that they needed to create the queue.

3 of the participants also mentioned that they make playlists very infrequently, with 2 of them only making them for each new time period. This was due to ""[insert quote about how much effort it was]

Something that didn't cause friction was the absolute or fixed ordering of the contents of their songs. 3 participants ordered by time[Vedarth,] or alphabetical order[Shruthi], all remarking that this consistnent order provided "muscle memory" and aligns+reinforces with their mental model 3 of the participants also ordered songs in the collection to elicit a specific listening experience. 1 participant[Casper] only listened to albums in their canonical 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[Casper, Roberto]. 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[Josh, Riya] 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"[Casper?] that they can forget to add these new songs to their collection. 1 participant[Riya] 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[Riya,

Josh] 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 TODO: Application Feedback

Before, the interviews were aimed at undrestanding how the participants organised their digital music collection and how they build and listen to queues using this collection. Whilst using the applicatio, the interview changed to being more about gaining feedback on the usability of the system and each individual feature implemented. Any features that the participants felt would be useful (after interacting with the application) were explored in detail.

As the participants interacted with the application they were asked for their feedback on the implemented features and the attributes:

5.3.1 Feedback: Attributes/Metadata

5.3.1.1 Value Accuracy

2 participants disagreed with some of the attribute values for songs in their collection, noting that it did not align with what they expected.

5.3.1.2 Attribute Opinions

Key, Mode, Time Signature Some cared, some didn't care.

Overall these are more useful in the calculating similarity and also being able to be toggled off (although the toggle off is probably unnecessary as no-one said they actively didn't want it there)

Attribute Rankings: Time above all else [Vedarth] noted that the time axis was easier to understand and more interested in as time/history is more familiar to them. [Casper] also noted that a sped up line of their audio history would be cool, also implying that time was a instinctively useful attribute

Attribute Combinations: Overwhelmed with Choice Many participants[] noted that though specific combinations were interesting they were overwhelemed with trying to find specific combinations and weren't sure where to start or what to do. X participants agreed that they would like **customisable presets**, where they could be given combinations to look at that provide easily digestible insights. Using these combinations as a base, the users' could then explore to find their own preferred combinations.

Distributions: Liked seeing ways of representing their Music Taste

Desired Feature: Ridge Plot of Histograms for Individual Histograms

Attribute: Extreme Ends Liked seeing the extreme ends for each attribute (i.e. top 5) although this could be due to the fact that the table view made it easy to see this. 1 participant[Casper] noted that these extreme ends would be useful for automatically creating high danceability playlists for example. There is alreay projects that can do this, though it does beg the question, if we start combing attributes, what sort of playlists would be created.

5.3.2 Feedback: Graph Model

TODO NEXT

5.3.2.1 Graph-Based Suggestions

3 participants[] noted that they would appreciate seeing song suggestions that clearly show how they slot into their entire collection. Both the ridge plot and dynamic graph would be good for this as they both show the user's entire collection.

5.3.2.2

Graph Navigation Controls Participants noted that the controls were good for navigating in the 2D graph. However, some particiapants[] felt that they would also like a 1st person style of controls, where they could fly through the space of songs. something akin to Minecraft's Creative Mode Controls

5.3.2.3 Song Identification

Due to time constraints, setting the colour of a song sphere was unfinished. To understand what would be the most preferable dimension to distinguish songs, participants were asked during the study, with the below responses:

- by colour of
  - Artist = 1[]
  - Genre = 4[]
  - Mood/Vibe = 3[]
  - Discrete Metrics = 0, as participants felt that they would need to see it implemented to see if it would be useful
- the image of the album the track belongs to (but only if there was enough visual space to render it)

5.3.2.4 Dynamic Graph

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.

Songs as Listening Focus Points 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 (2700 words)

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 FINISH: 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 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.

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.

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.

lbum 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.

Chapter 7

Project Planning Retrospective (1650 words)

This section will explore in detail the planning and development of the project itself, throughout the stages: research, designing, development, evaluation and the final write-up.

I will go over what were the significant issues with the approach I took. I will then talk about how I would this project were I to start it over again.

Since this project had a very limited timeframe, ensuring that progress remained on schedule was critical.

7.1 What Went Well

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 meant that I could decompose the project into its significant core parts. Each section could then be decomposed into its significant parts. This significantly reduced the mental load and allowed for me to better breakdown each aspect of the project as much as I needed to.

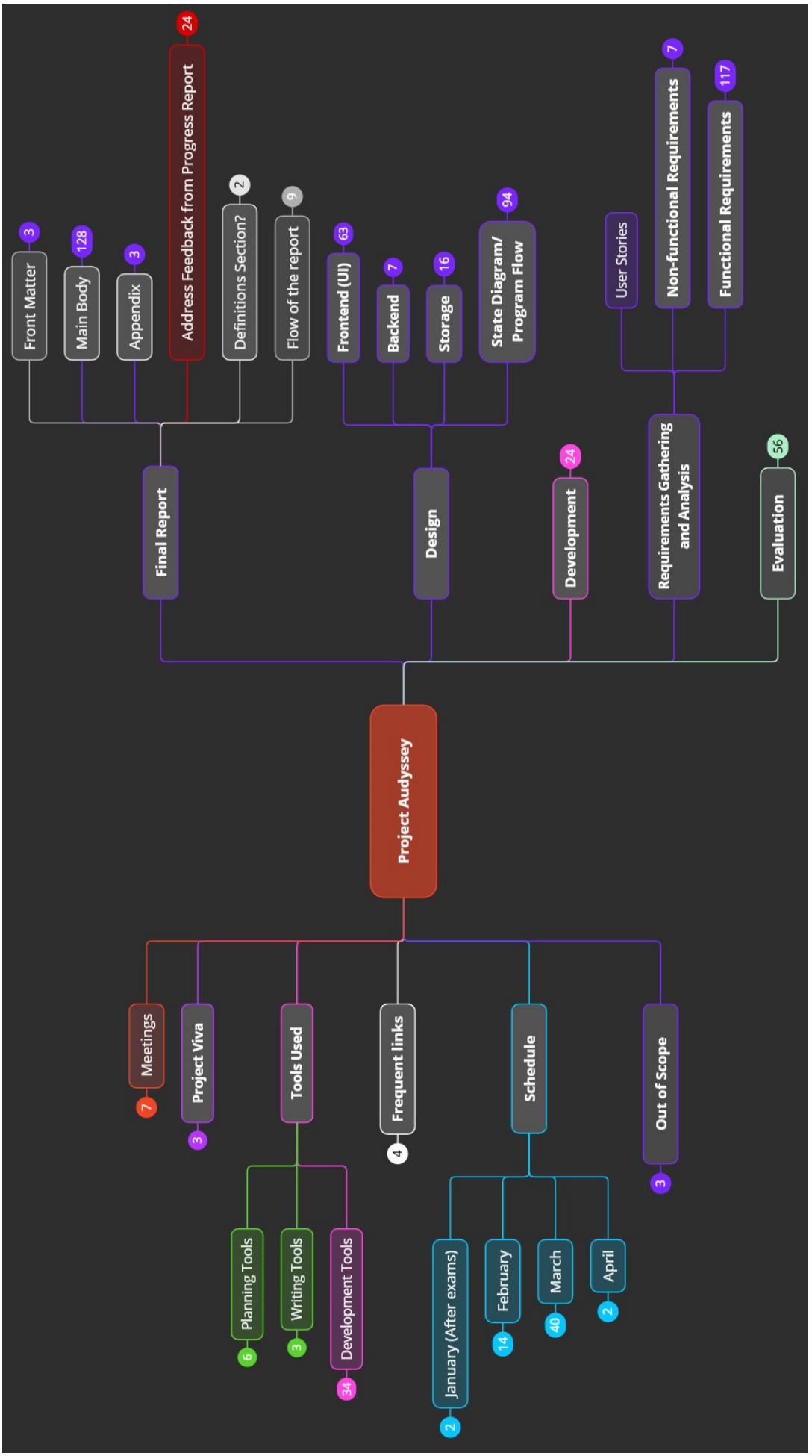

Figure 7.1: Miro Mindmap high level overview

7.2 Limitations

There were some factors that limited this project: [-] Unfinished Code limiting the ability to fully evaluate the concepts [-] spent too long on the Spotify and SoundCharts API stuff [-] lost 1-2 weeks due to LOpSoc and AdvCompArch [-] Only got 6 people evaluated due to time constraints, still was very informative but more would've been better [-] ideally 8-10 participants to really see the patterns [-] final evaluation was quite a lengthy process, should've done a mini user survey before the progress report to both find requirements and get an initial understanding. Then it would've been easier to see how the application and new concepts would've affected the listeners [-] should've done more research into what the attributes that Spotify was offering were earlier in the research stage (not in the development stage) as that would have led to figuring out Exportify as the better route over SoundCharts. - In my defence, Spotify has obscured that the attributes were from the Echo Nest in their developer API. - they also deprecated the API endpoint days before the progress report handin, a significant milestone in the project meaning that as soon as an alternative was found, I had to take what I could get

7.2.1 Spotify Deprecating Access of Echo Nest Attributes

[2014] Spotify buys EchoNest - now EchoNest's API is locked behind a Spotify Premium Account (this wasn't an issue due to already having a premium account, but this is a paywall), however this was also a sort of blessing in disguise as it would've meant that I only need to work with one API.

[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 necessary as they would provide the numerical basis for mapping the songs and helping control the audio journeys.

o ensure that this didn't derail the project, after much searching an alternative was found, namely 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.

Unfortunately, due to the sudden nature of the Spotify deprecation announcement, an alternative had to be chosen very quickly meaning that Exportify wasn't found until very, very late in development. This meant that the API setup portion of the project took longer than it theoretically could've, as the workflow using Exportify's `.csv` files is much simpler (as shown in the figure NUMBER below). The ideal method would've been to continue searching for alternatives after finding SoundCharts allowing for Exportify to be found sooner and be integrated into the application state flow from the start, however due to the time constraint in needing to submit a project brief, this was not done.

SoundCharts - (good but not as detailed info, plus didn't give confidence interval endpoint and also paywalled which was an unreasonable amount of faff) - wasted a lot of time trying to get a workflow that allowed for not having to send repeated requests for the same song

7.2.2 Unfinished Code in the Development Stage

I didn't realise quite how much work it would be to develop the whole project. As such when the project reached a critical decision point in the development I made the wrong decision. Whilst the static graphs were useful for the project as explained in the evaluation section, I should've prioritised the dynamic journeys. This decision point came at a time when only one of the static or dynamic graph features would have time to be done properly.

Another reason is that I let other **subjects and responsibilities slow down progress too much**. (LOpSoc and AdvCompArch I guess??)

Progress was also slowed down by having a more complex application flow due to initially requiring SoundCharts API. Integrating Exportify's `.csv` files meant that the setup state was simpler and make it easier to combine playlists when using the graph views. Unfortunately, due to switching to Exportify too late, there was no time reap these rewards.

7.2.3 Losing sight of the Goal Only learning and understanding the true goal of the project near the end.

This project whilst in a limited timeframe, was still quite long, having transpired over the better part of 7 months. During this process the true goal/research questions were only fully understood quite late into the project sadly.

7.2.3.1 Static Cartesian Graphs: A Red Herring

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 (none that I could find at least) 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 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.

To prevent the static graphs feature be the 'red herring' that they were, in taking away development focus from the actual significant features (the dynamic graph and listening journeys) then this feature should've been omitted from the project during the research stage. For this delay to have not occurred, the ideal scenario would've been to do more research in how the EchoNest attributes have been used much earlier in the project, before the design and development stages. During the research stage, more energy was invested into where to source the song attributes, meaning extensive research into how they had been used was only investigated during the development stage.

Whereas the more detailed in-depth view of the library by looking at the distributions of specific metrics is more useful when the person allocates a specific time to go through it. This is because the view requires more mental energy to understand and explore it, to reach a sufficient mental understanding of it.

7.2.4 Scope Creep: Implementing Better Listening

Initially, as detailed in the project brief, this project aimed to create better listening queues by allowing for combinations of fixed and unfixed queues and greater flexibility in changing the directoin of the song queue.

7.3 TODO: Initial vs Final vs Ideal Project Plan

Below are three project plans:

- **Initial Plan** → this is what was planned initially near the start of the project
- **Actual Progress** → this is the actual progress of the project
- **Ideal Plan** → upon retrospection, this is how the project should be approached if to be done again

7.3.1 Initial Plan

7.3.2 Actual Progress

7.3.3 Ideal Plan

Key Points: - reduce the amount of features - should've only done the dynamic graph as other people have done the static cartesian graphs before - the evaluation itself took ages - either of the below: - do an initial survey of people before design and development -¿ get a better sense of requirements - had a review meeting with the review team on the application - was supposed to do this but hadn't finished enough code in time and they both left the country early - minor thing, but should've asked if the action of skipping was annoying/effort

Chapter 8

TODO: Conclusion (250 words)

8.1 Future Work

8.1.1 User-Requested Features

[-] Buffer zone for songs that have been recently listened to [-] added/viewable in collection but show that they aren't fully part of the collection yet [-] if this grows too much might come across the issue of people not wanting to deal with it. [-] need to be careful to figure out how to make it actually useful and not overwhelming for a user [-] how does this relate to song suggestions? are they one and the same or close enough??

8.1.1.1 Customisable Presets: Attribute Combinatinos

Thiis feature would help to handle the significantly large scale in the possible combinations of attributes for the static cartesian graphs. [-] require a lot of research into the different shapes and distributions that appear for all combinations [-] need a largre varied data set [-] possibly have to find an algorithm or automatable way to see if a distribution might be more useful than another [-] need to build a ranking list effectively of all the combinations

8.1.2 Further Research

[-] Rendering listening history [-] Auto-generating Playlists, possibly using song tags as a base [-] Adding Song tags [-] explicit is an example of a pre-existing tag, however there is a lot of complexity involved with letting users create and add tags. [-] would have to research into how iTunes did their tags and what went well and what didn't (Obsidian and any others as well) [-] automatic tag adding is desired for adding tags to a high volume of songs [-] make them hierarchical like the way genre works? [-] Figuring

out how to make this research more mobile friendly as more people listen using their phones[**?**]

Bibliography

[1] 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

TODO: Project Brief

This is an appendix

Appendix B: Code Listings

This is an appendix

TODO: Miro Diagrams Expanded

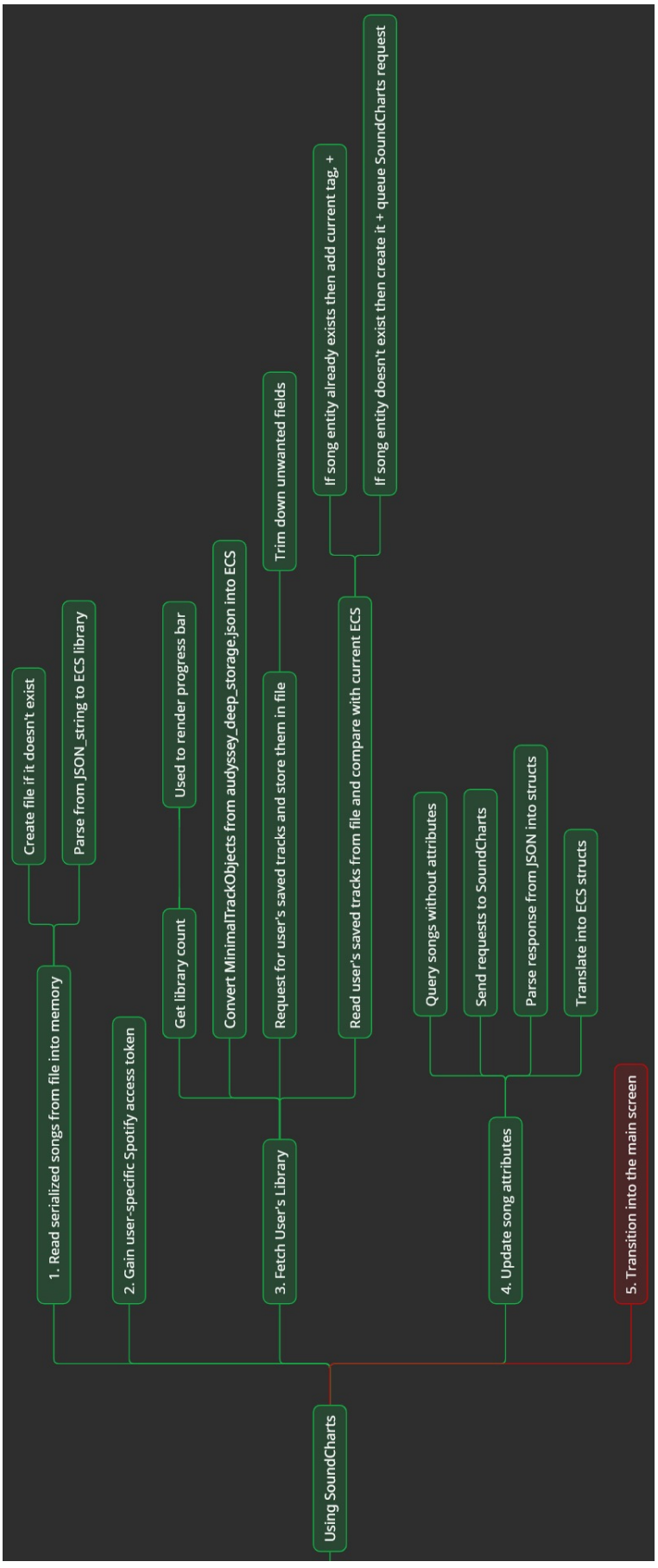

Figure 1: SoundCharts Expanded State Flow for filling user's songs with attributes

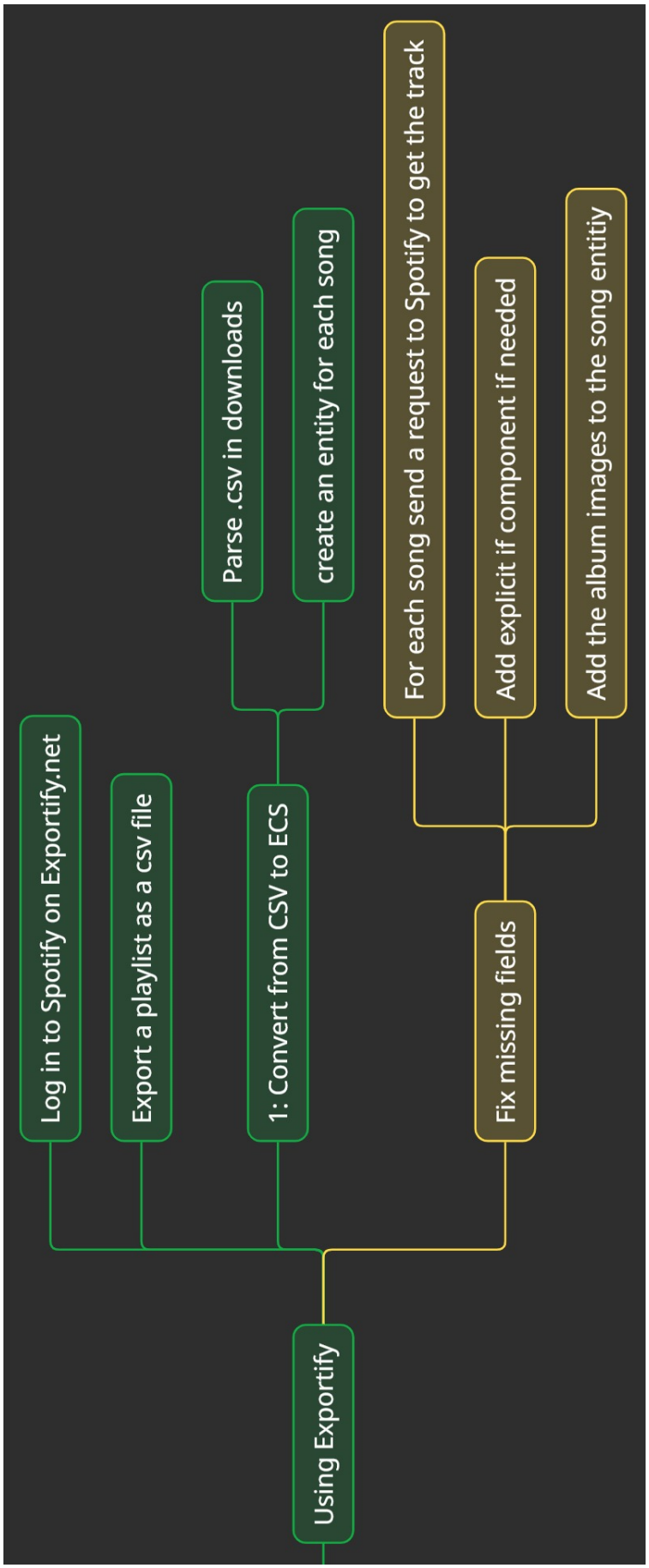

Figure 2: Exportify Expanded State Flow for getting all songs (with attributes) in a specific playlist

TODO: Risk Assessment

TODO: Detailed Component Definitions