COS10009

Introduction To Programming

# Design Report

## Custom Program

Origami Musica – Advanced Music Player

Name: Viet Hoang Pham

Student ID: 104506968

Tutor: Xinyi Cai

Date: Semester 2 – 2023

# TABLE OF CONTENTS

# I. INTRODUCTION

***Origami Musica*** is a music player designed for music aficionados seeking an enriched listening experience. With a focus on functionality, ease of use, and sophisticated features, this program redefines conventional music playback.

## 1. Report Objective

This report aims to achieve the following key objectives:

- An overview of ***Origami Musica*** goals.
- An in-depth analysis of ***Origami Musica*** structure, followed by a structure chart.
- A data dictionary that explains the structure and format of the records in the ***Origami Musica***.
- A comprehensive report of innovative functionalities of the ***Origami Musica***.

## 2. Report Outline

The rest of the report is organized as follows:

- ***Origami Musica Goals:*** A summary of the custom program goals, and functionalities.
- ***Origami Musica Overview:*** An overview of the custom program, including the program presentation and used Ruby libraries.
- ***Origami Musica Structure:*** An outline of the custom program structure, including the program structure chart and Gosu structure.
- ***Origami Musica Data Dictionary:*** A presentation of records and enumerations format and how it is connected.
- ***Origami Musica Functionalities:*** An in-depth description of the core functionalities of the custom program and how it works.

# II. ORIGAMI MUSICA

## 1. Origami Musica Goals

***Origami Musica*** sets itself apart by implementing cutting-edge features with an intuitive UI. Its primary goals are enhancing user interaction, and offering advanced functionalities that go beyond basic music playback. It provides track searching with advanced search algorithms like ***Danmereu-Levenshtein Edit Distance***, an Approximately String Matching algorithm, as well as easy navigation across album libraries and the ***personalized*** generation of ***smooth-transition*** playlists utilizing ***Repeated Nearest Neighbor (RNN)*** *and* ***Genetic Algorithms (GA).***

## 2. Origami Musica Overview

### *A. Program Appearance*

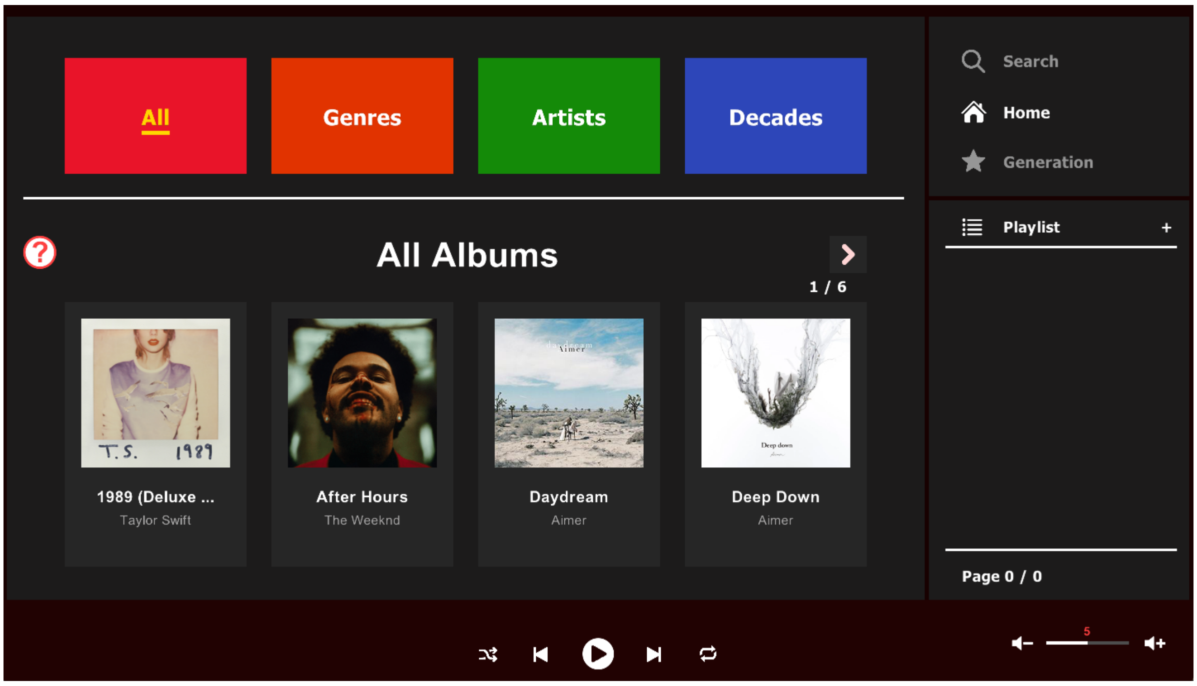


*Figure 1: Home Page*

- ***Categories Bar:*** Users can interact with the categories bar to filter albums based on their selected genres/artists/decades.
- ***Albums Display:*** The Home Page will display available albums, and the ability to scroll through all albums by clicking on the arrow at the top right. There are a maximum of 4 albums on each page.

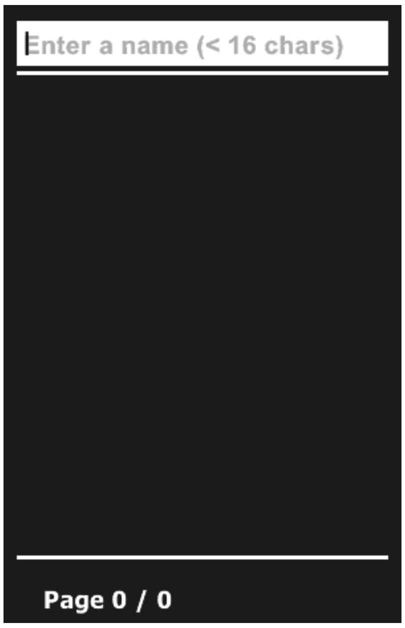


*Figure 2: Playlist Bar*

- ***Create Playlist:*** The user can type a name when they want to create a new playlist.

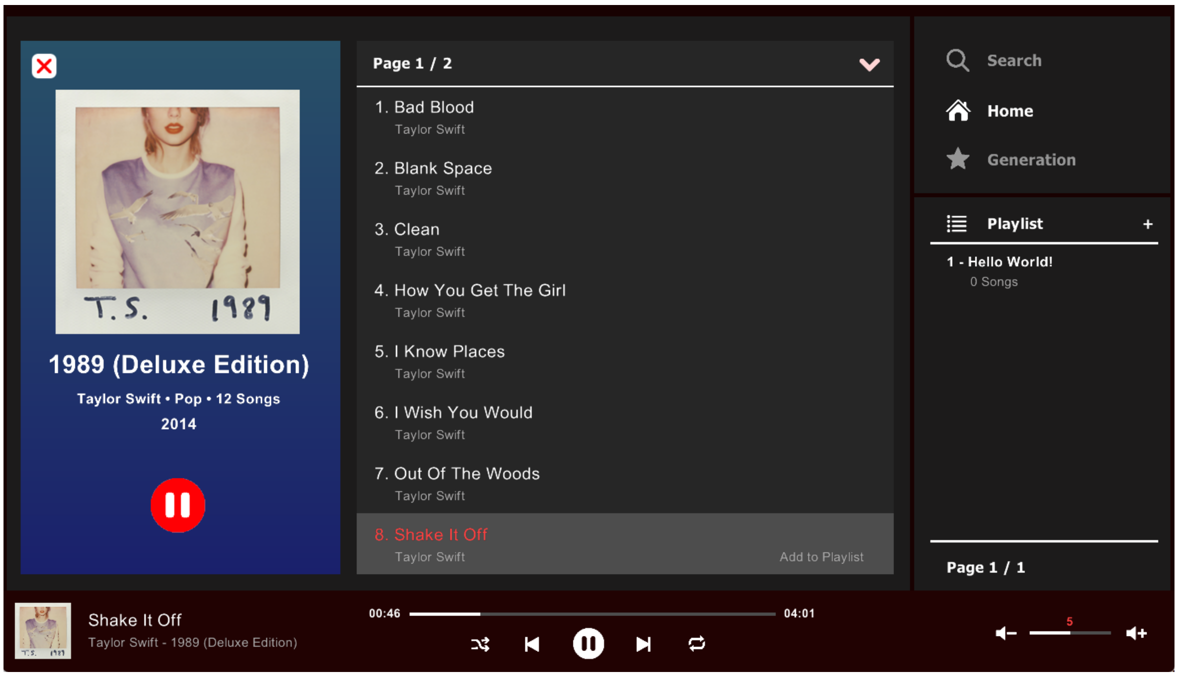


*Figure 3: Selected Albums Page*

- ***Tracks Display:*** This page will display all available tracks in the album, and the ability to scroll through all tracks by clicking on the arrows at the top right. There are a maximum of 8 tracks on each page.
- ***Add to Playlist:*** When the user hovers over a track, they will see *"Add to Playlist"*, by clicking on that, they will see ***this Page***:

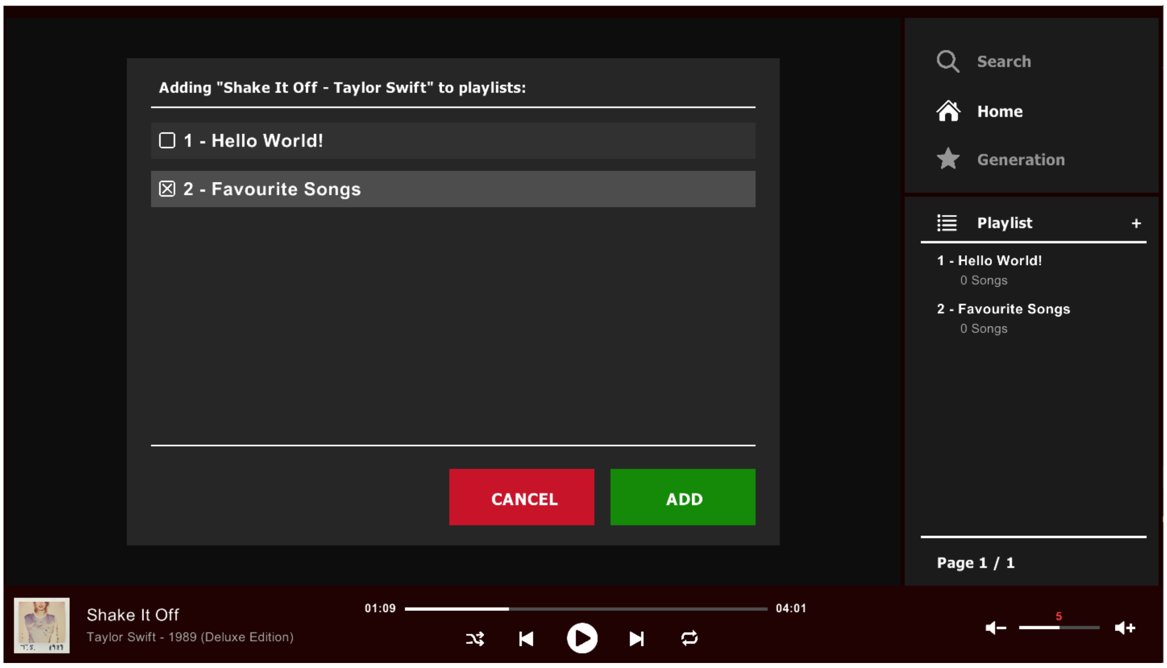


*Figure 4: Add Tracks To Playlist Page*

- ***Add Tracks To Playlist:*** The user can either cancel or add track to the playlist. After adding a track to the playlist, there will be a notification box appear:

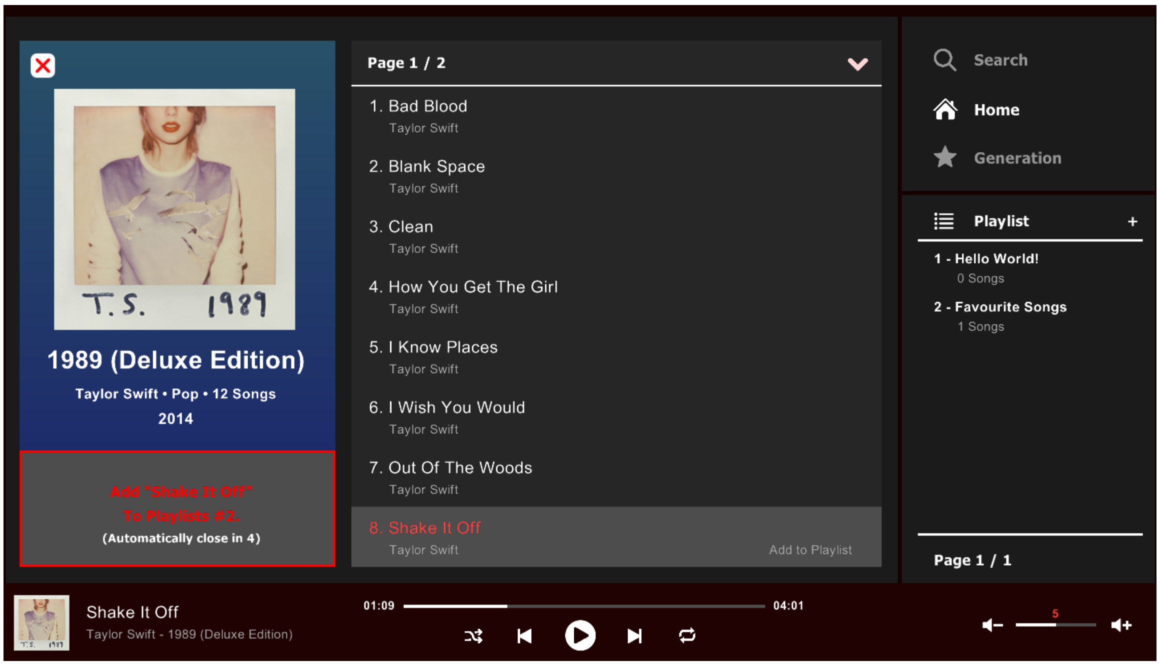


*Figure 5: Notification Box Appear*

- ***Notification Box:*** When a user interaction needs to be notified, the notification box will appear and automatically close in 5 seconds.

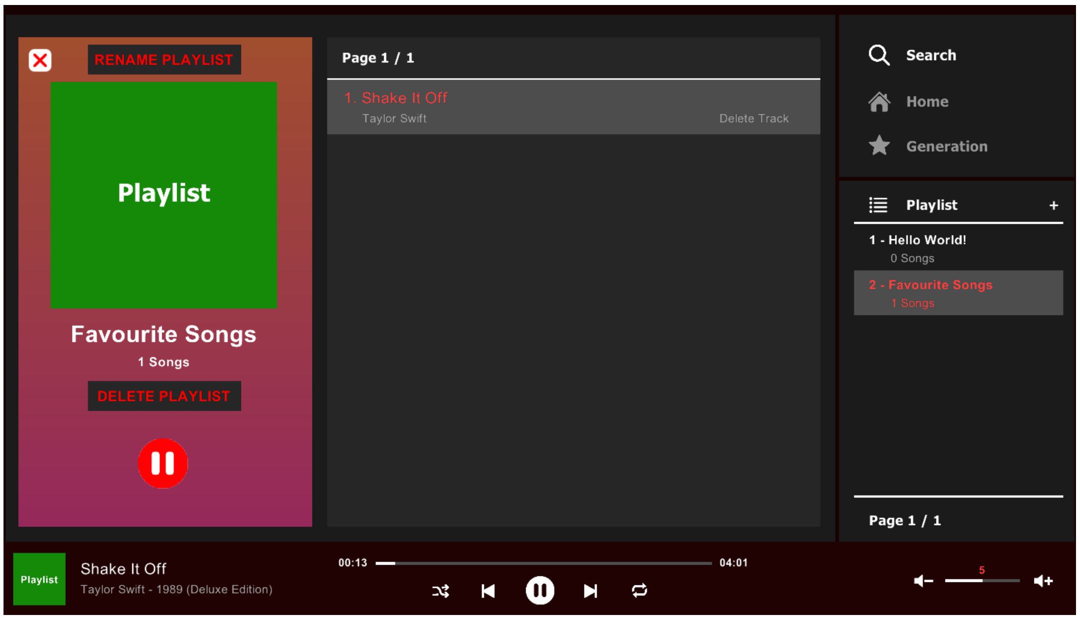
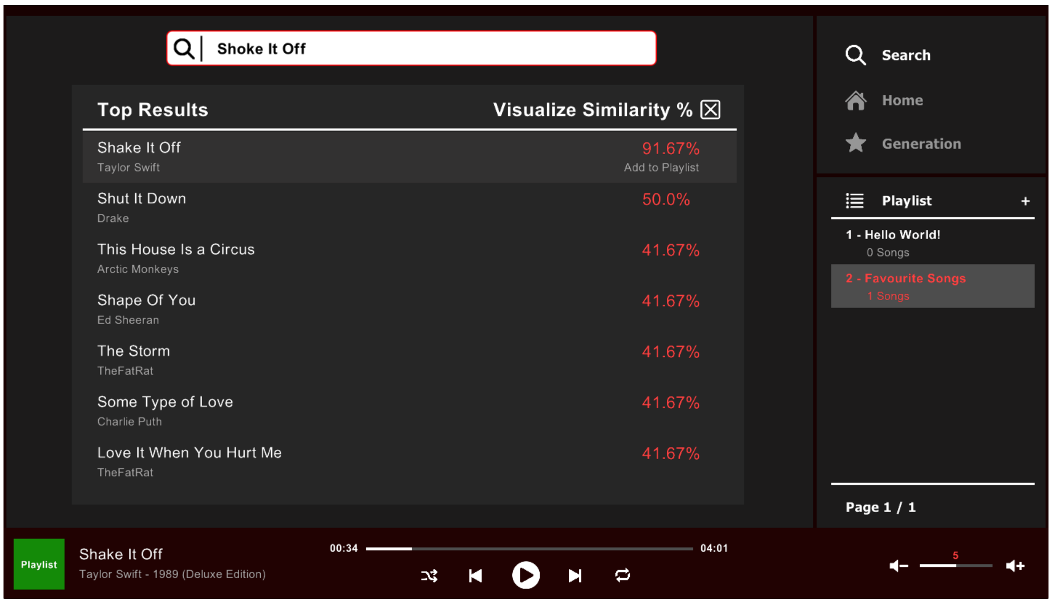


Figure 6: Playlist Page

- ***Tracks Display:*** This page will display all tracks in the selected playlist, and the ability to scroll through all tracks by clicking on the arrows at the top right. There are a maximum of 8 tracks on each page.
- ***Some Playlist Operations (Delete Playlist, Delete Track From Playlist, Rename Playlist, …)*** will be documented in the **Playlist Operations** in **Origami Musica Functionalities**.



| Top Results | Visualize Similarity % |
|---|---|
| Shake It Off<br>Taylor Swift | 91.67%<br>Add to Playlist |
| Shut It Down<br>Drake | 50.0% |
| This House Is a Circus<br>Arctic Monkeys | 41.67% |
| Shape Of You<br>Ed Sheeran | 41.67% |
| The Storm<br>TheFatRat | 41.67% |
| Some Type of Love<br>Charlie Puth | 41.67% |
| Love It When You Hurt Me<br>TheFatRat | 41.67% |

Figure 7: Search Page

- ***Approximately Track Searching:*** By using the ***Restricted Danmereu-Levenshtein Algorithm,*** the user can search for the track they want in the music player, even if they make two or three typos in the track name (**maximum typos is 7)**. The user can also search for the keyword in a track with a long name. See **Track Searching**.

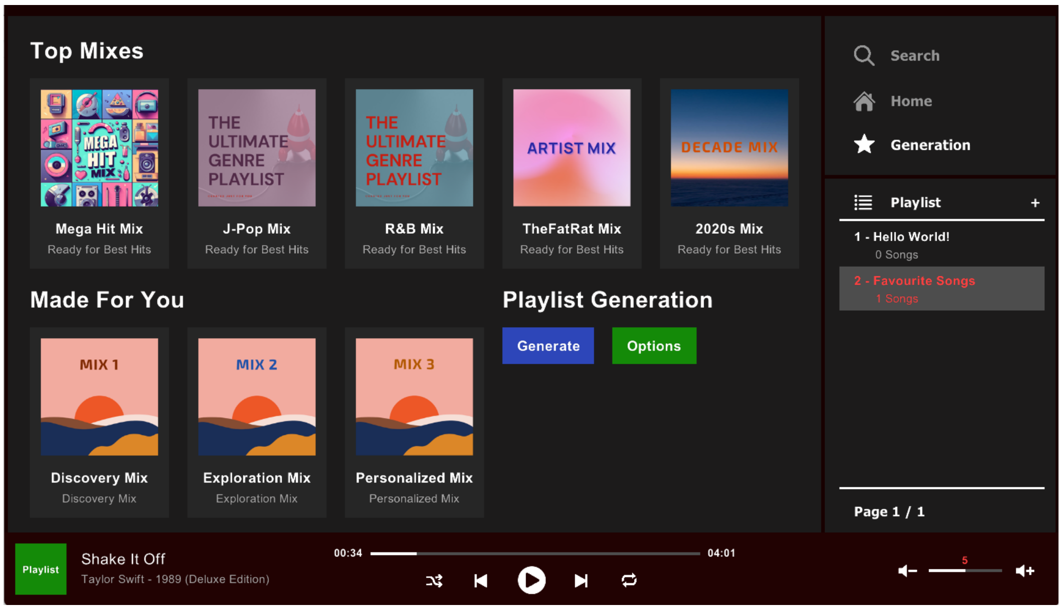


Figure 8: Generation Page

- ***Initial Generation Playlist:*** The initial generation playlist will be generated when the user opens the music player **(the music player must contain tracks to show this page)**. These playlists are not personalized because the program will choose random genres/artists/decades and the playlists will contain popular tracks starting from the most popular.
- ***Personalized Generation Playlists:*** By tracking user interactions history and storing it in ***historyinteractions.txt,*** the playlists will be generated based on the most interactions genres, artists, decades, and audio features, …. However, the user can also choose their preferences for the playlist:

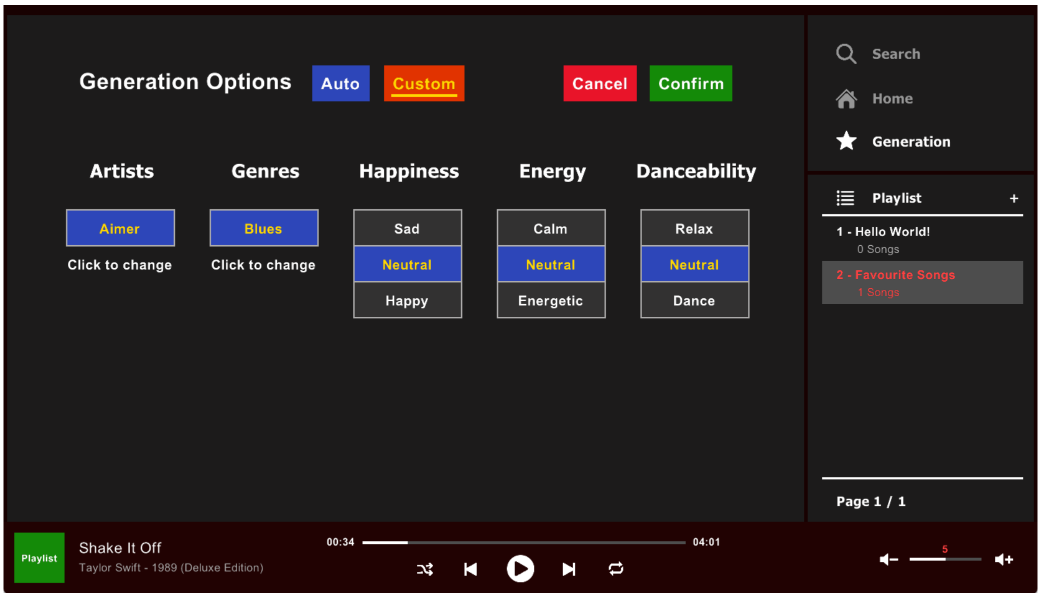
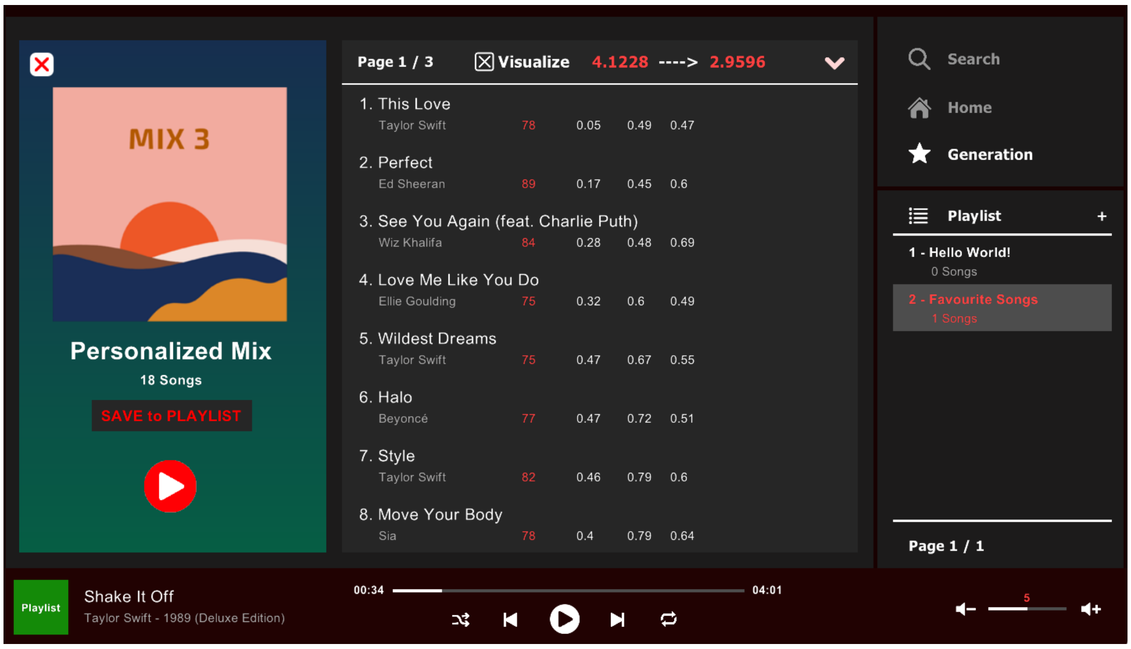


Figure 9: User Preferences Control Panel

- ***Smooth-Transition Playlist Generation:*** After the initial playlists or personalized playlists have been created, the program performs ***the last step*** that reorders the track's position in the playlist making the tracks transition in the playlist as smooth as possible. These steps will be documented in the **Personalized Playlist Generation System** in **Origami Musica Functionalities.**



| | | | | |
|---|---|---|---|---|
| 1. This Love<br>Taylor Swift | 78 | 0.05 | 0.49 | 0.47 |
| 2. Perfect<br>Ed Sheeran | 89 | 0.17 | 0.45 | 0.6 |
| 3. See You Again (feat. Charlie Puth)<br>Wiz Khalifa | 84 | 0.28 | 0.48 | 0.69 |
| 4. Love Me Like You Do<br>Ellie Goulding | 75 | 0.32 | 0.6 | 0.49 |
| 5. Wildest Dreams<br>Taylor Swift | 75 | 0.47 | 0.67 | 0.55 |
| 6. Halo<br>Beyoncé | 77 | 0.47 | 0.72 | 0.51 |
| 7. Style<br>Taylor Swift | 82 | 0.46 | 0.79 | 0.6 |
| 8. Move Your Body<br>Sia | 78 | 0.4 | 0.79 | 0.64 |

*Figure 10: A smooth-transition personalized playlist with Taylor Swift, Pop, and the 2010s as user preferences*

- ***Save the generated playlist to playlist libraries:*** The user can save these generated playlists into their playlist libraries.

## B. Program Ruby Libraries

- ***Gosu*** enables efficient display of graphics, audio elements, and user interactions, ensuring an engaging user experience while navigating through the ***Origami Musica***.
- ***Set*** is used for storing unique music elements like genres, artists, and decades.
- ***Mp3info*** is an essential library to read ID3 metadata in MP3 files, providing ***Origami Musica*** the ability to automatically discover and load albums and track metadata from the program directory.
- ***Rspotify*** is a **Ruby wrapper** for the **Spotify Web API.** The program used ***Rspotify*** to retrieve audio features of tracks like valence, acousticness, energy, danceability, energy, and tempo. The ***personalized playlist generation system*** is built by using these data.
  ➔ However, because the **Spotify Web API** is not fast enough to retrieve data, which deteriorates user experiences, the retrieved audio features of each track are stored locally in ***LocalAPIStorage.txt***. Therefore, the ***Tracks_Audio_features.rb*** file is used for demonstration only as it won't contribute to the main program code.

## 3. Origami Musica Structure

### A. *Structure Chart*

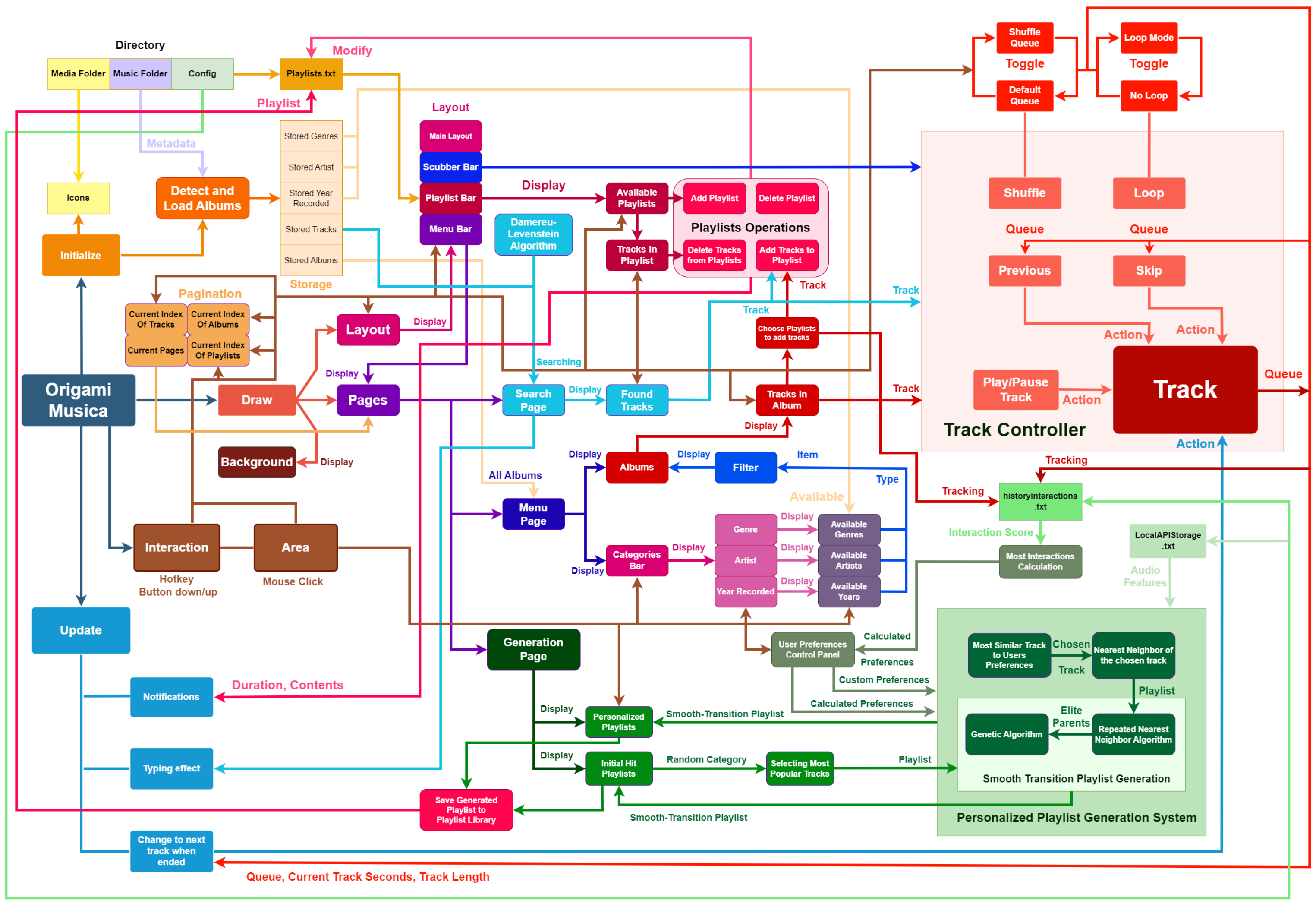


*Figure 11: Origami Musica Structure Chart (Higher Resolution Image is included in the submission)*

### B. *Program Gosu Structure*

#### a. Program Initialization

- I have used many instance variables to keep track of the records and logical process of the program. Some important variables are:

***@albums_storage & @albums***

- Storing available albums in the music player. The ***detect_and_load_albums()*** function will automatically scan through all folders in the ***music*** folder for the ***mp3*** files and album cover image. The data format will be documented in the **Data Dictionary** section.

***@tracks_storage***

- With the ***read_tracks_to_store()*** function, it will work with ***@albums_storage*** to extract tracks from each album for storage. This variable is mainly used for *track searching* and *personalized playlist generation.* The data format will be documented in the **Data Dictionary**.

***@genres_storage, @artists_storage, @decades_storage***

- By retrieving available genres, artists, and decades from ***@albums_storage,*** these variables are used for *filtering albums by categories* and *personalized playlist generation.*

***@playlists_storage & @playlists***

- If the ***playlists.txt*** file in the ***config*** folder existed, it means that the user has created some playlists when they are using the music player. ***@playlists*** variable is used to store available playlists that the user has created.

***Layout Variables (@margin, @playback_height, @right_bar_x, …)***

- These variables are used for maintaining the consistency of the custom program appearance.

***Pagination Variables (@first_current_page_album_index, @first_current_page_tracks_index, …)***

- These variables are used for **Pagination** documented in the **Origami Musica Functionalities.**

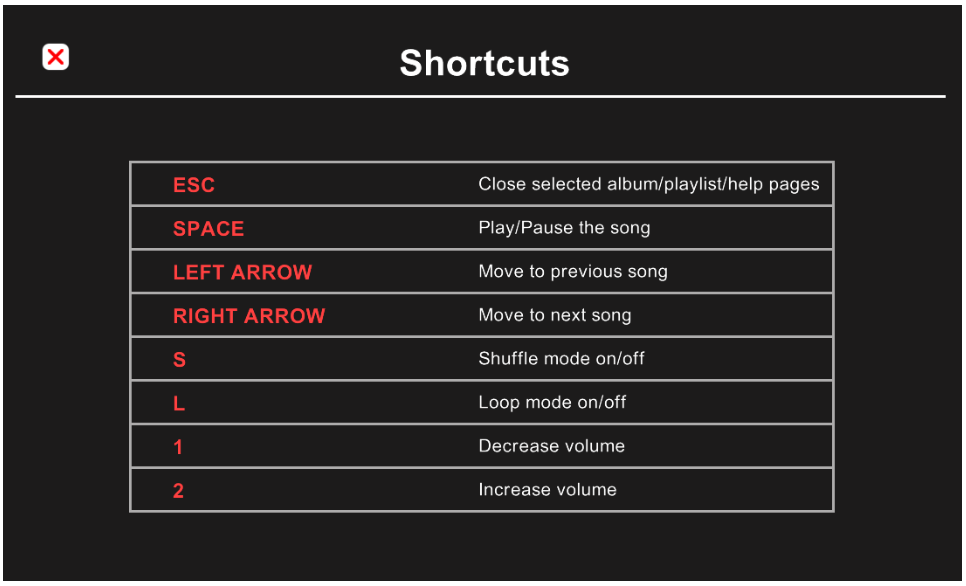
***@selected_track_index_array***

- This variable is an array representing the current queue of the music player.

**b. Draw()**

- In ***Origami Musica*** program, ***draw()*** is closely linked to ***button_down()*** in **Interactions.** If ***button_down()*** is used for manipulating and controlling the logic flow of the program, ***draw()*** will visualize pages and contents based on the user interaction in ***button_down()***.

## c. Interactions (Shortcuts, Area Interaction)



Shortcuts

| | |
|---|---|
| ESC | Close selected album/playlist/help pages |
| SPACE | Play/Pause the song |
| LEFT ARROW | Move to previous song |
| RIGHT ARROW | Move to next song |
| S | Shuffle mode on/off |
| L | Loop mode on/off |
| 1 | Decrease volume |
| 2 | Increase volume |

*Figure 12: Help Page of Origami Musica displaying all shortcuts that the user can use.*

- By utilizing ***button_down()*** and ***Gosu::MsLeft,*** many areas in the music player have been created for different interactions in the ***Origami Musica.*** The interaction of each area is also based on the logic flow of the program.

## d. Update

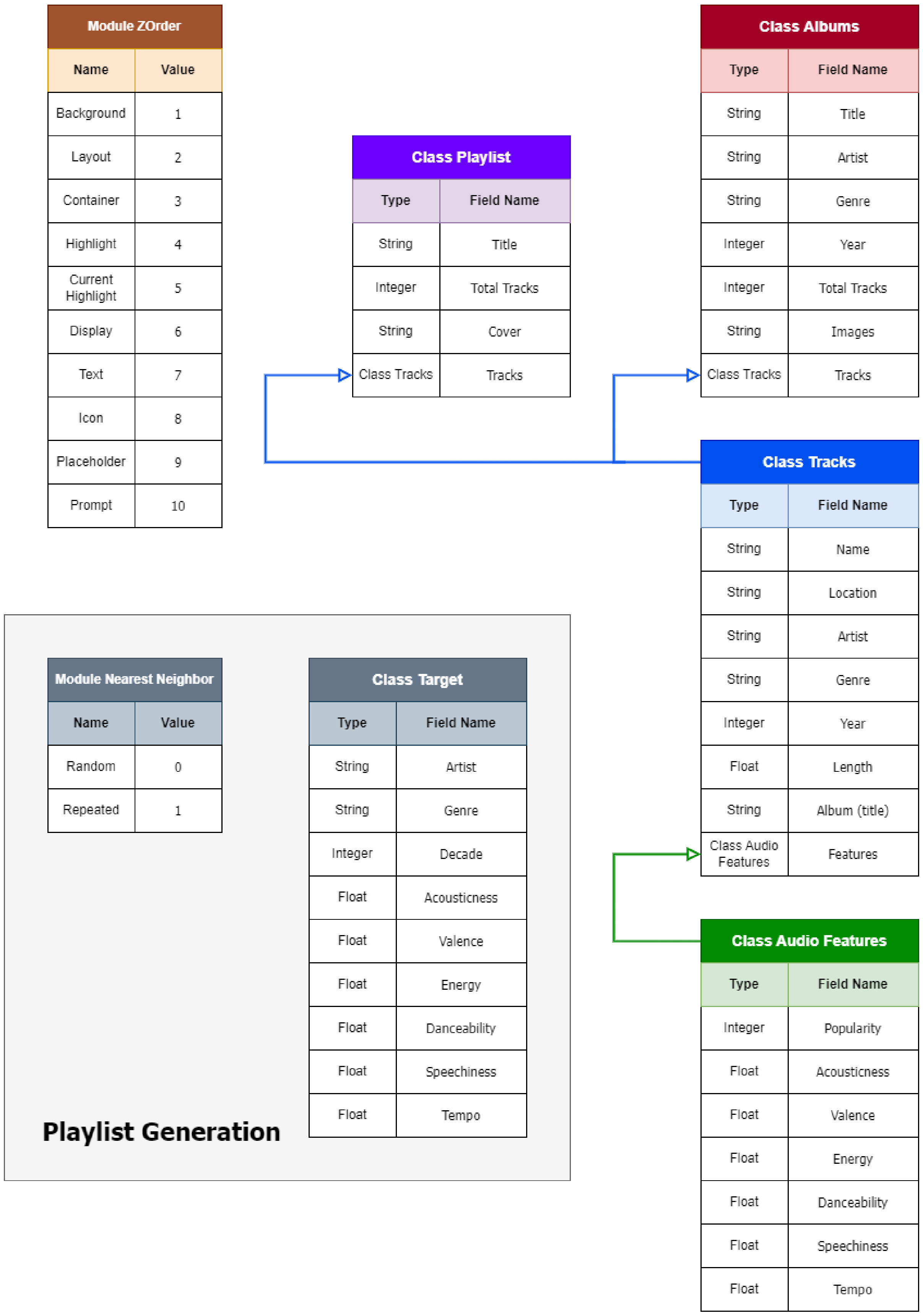
***Automatically play the next song when the current song ends***

- By using ***Gosu.milliseconds*** and because the ***update()*** is being called 60 times per second, whenever a song is being played, the music player will store the last update ***Gosu.milliseconds*** of the Gosu Program, then at the next call of the ***update()*** function, it will subtract the current ***Gosu.milliseconds*** with the last update ***Gosu.milliseconds*** and add the result to the current song seconds. After the current song's seconds exceed the track length, it will automatically play the next song.

***Notification Duration***

- Whenever an interaction needs to be notified to the user. The notification durations will be counted in the ***update()*** function so that the notification box will automatically close in 5 seconds.

## 4. Origami Musica Data Dictionary



**Module ZOrder**

| Name | Value |
|---|---|
| Background | 1 |
| Layout | 2 |
| Container | 3 |
| Highlight | 4 |
| Current Highlight | 5 |
| Display | 6 |
| Text | 7 |
| Icon | 8 |
| Placeholder | 9 |
| Prompt | 10 |

**Class Playlist**

| Type | Field Name |
|---|---|
| String | Title |
| Integer | Total Tracks |
| String | Cover |
| Class Tracks | Tracks |

**Class Albums**

| Type | Field Name |
|---|---|
| String | Title |
| String | Artist |
| String | Genre |
| Integer | Year |
| Integer | Total Tracks |
| String | Images |
| Class Tracks | Tracks |

**Class Tracks**

| Type | Field Name |
|---|---|
| String | Name |
| String | Location |
| String | Artist |
| String | Genre |
| Integer | Year |
| Float | Length |
| String | Album (title) |
| Class Audio Features | Features |

**Module Nearest Neighbor**

| Name | Value |
|---|---|
| Random | 0 |
| Repeated | 1 |

**Class Target**

| Type | Field Name |
|---|---|
| String | Artist |
| String | Genre |
| Integer | Decade |
| Float | Acousticness |
| Float | Valence |
| Float | Energy |
| Float | Danceability |
| Float | Speechiness |
| Float | Tempo |

**Playlist Generation**

**Class Audio Features**

| Type | Field Name |
|---|---|
| Integer | Popularity |
| Float | Acousticness |
| Float | Valence |
| Float | Energy |
| Float | Danceability |
| Float | Speechiness |
| Float | Tempo |

*Figure 13: Records and Enumeration*

## 5. Origami Musica Functionalities

### A. Audio Playback

- ***Origami Musica*** provides standard audio playback functionality, allowing users to play, pause, skip, and control the volume of music tracks.

    - ***Play/Pause/Volume Control:*** By using Gosu built-in functions, the user can play, pause, and change the volume of the track.
    - ***Skip to the next/Back to the previous track:*** With ***@selected_track_index_array (documented in <u>Program Initialization</u>)*** being used as the array to store the current queue of the music player, when the user skips to the next track or back to the previous track, the current index in the queue will be incremented or decremented accordingly.

### B. Audio Playback Control Features

- ***Origami Musica*** allows users to turn on/off shuffle and loop modes.

    - ***Shuffle Mode:*** When the *Shuffle Mode* is turned on, the ***@selected_track_index_array*** queue will be shuffled in a random order. If the *Shuffle Mode* is turned off, the queue will restore to the default queue.
    - ***Loop Mode:*** When the *Loop Mode* is turned on and the current index is at the end of the ***@selected_track_index_array*** queue, the index will be assigned the first index of the queue when the next track plays.

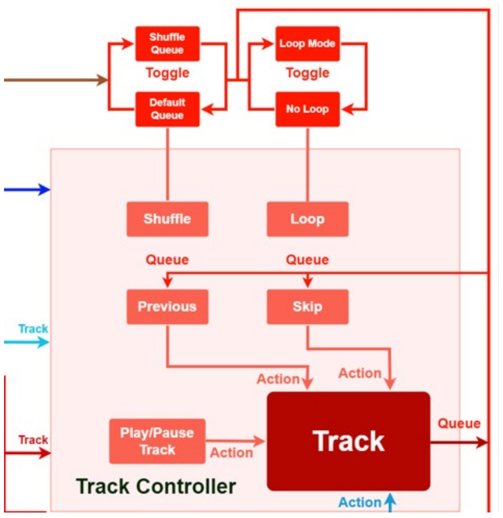


*Figure 14: Audio Playback Control Features in the Structure Chart*

### C. Pagination

- ***Origami Musica*** offers the user a way to scroll through their albums or playlist libraries. By tracking ***the index of the first item of each page,*** the program will display (**4 for albums and categories, 6 for playlists, 8 for tracks)** items until the user reaches the last page.

## *D. Automatically Discover and Load Albums*

- See **Program Initialization** for details.

## *E. Filter album libraries with a genre/an artist/decade.*

- See **Program Appearance** for details.

## *F. Playlist Operations*

- The ***playlists.txt*** will be used to store the playlist libraries. Whenever the user makes the following actions to the playlist libraries, the file will be modified accordingly.

**a. Create a playlist**

- The user can create a playlist with a name of less than 16 characters. See **Program Appearance** for visualization.

**b. Add a track to playlists**

- The user can add a track to multiple playlists. See **Program Appearance** for visualization.

**c. Delete a track from a playlist**

- The user can delete a track from a playlist. See **Program Appearance** for visualization.

**d. Delete playlist**

- The user can delete a playlist from playlist libraries. See **Program Appearance** for visualization.

**e. Rename playlist**

- The user can rename a playlist whatever they want (within 16 characters). See **Program Appearance** for visualization.

**f. Save generated playlist to playlist libraries**

- After the personalized playlists are generated from the **Personalized Playlist Generation System** or initial playlists are generated from **Smooth-Transition Playlist Generation**, the user can save them to the playlist libraries. See **Program Appearance** for visualization.

## *G. Track Searching*

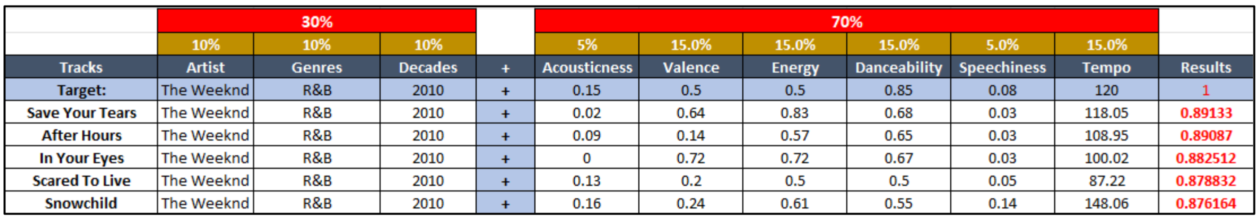
- ***Origami Musica*** supports the user in finding a track, even if the user makes a typo in the query by using the ***Danmereu-Levenshtein algorithm. Danmereu-Levenshtein distance*** is an approximate string-matching algorithm that counts the number of insertions, deletions, replacements, and transpositions required to transform the query into the target string (here is a track name). **The fewer operations required to transform to the target string, the more similar the two strings are.**

- The program also checks for the substring of a long track name, which means that it uses the above algorithm to check the two strings **with the same amount of words.** If the track name is too long, the string will be separated into **substrings** that have **the same amount of words** as the query string. This feature helps the user to find **the keyword of the track** without having to write the full name of the track to find it.

## *H. Personalized Playlist Generation System*

### *Step 1: Choose the most similar song to user interaction history*

- Calculate a target value based on user interaction history. Then, select the most similar track based on the target preferences. There are 9 categories to compare a track to the target preferences.



| | 30% | | | | 70% | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10% | 10% | 10% | | 5% | 15.0% | 15.0% | 15.0% | 5.0% | 15.0% | |
| Tracks | Artist | Genres | Decades | + | Acousticness | Valence | Energy | Danceability | Speechiness | Tempo | Results |
| Target: | The Weeknd | R&B | 2010 | + | 0.15 | 0.5 | 0.5 | 0.85 | 0.08 | 120 | 1 |
| Save Your Tears | The Weeknd | R&B | 2010 | + | 0.02 | 0.64 | 0.83 | 0.68 | 0.03 | 118.05 | 0.89133 |
| After Hours | The Weeknd | R&B | 2010 | + | 0.09 | 0.14 | 0.57 | 0.65 | 0.03 | 108.95 | 0.89087 |
| In Your Eyes | The Weeknd | R&B | 2010 | + | 0 | 0.72 | 0.72 | 0.67 | 0.03 | 100.02 | 0.882512 |
| Scared To Live | The Weeknd | R&B | 2010 | + | 0.13 | 0.2 | 0.5 | 0.5 | 0.05 | 87.22 | 0.878832 |
| Snowchild | The Weeknd | R&B | 2010 | + | 0.16 | 0.24 | 0.61 | 0.55 | 0.14 | 148.06 | 0.876164 |

*Figure 15: The most similar tracks based on target preferences.*

**Similarity method**: **Manhattan Distance**.

### *Step 2: Find the nearest neighbor of the chosen track to create a playlist*

- The difference between each type of personalized playlist is the level of exploration. The more exploration the playlist is, the more likely a new track with a different genre but the same audio features appear.

**Type 1: Discovery Mix (High Exploration)**

- Compare **valence, energy, danceability**

**Type 2: Exploration Mix (Medium Exploration)**

- Compare **valence, energy, danceability**. Only selected tracks with **70 or higher popularity** and slowly decremented only if there are not enough tracks to create a playlist.

**Type 3: Personalized Hit Mixes (No Exploration)**

- Compare **valence, energy, danceability**. Only selected tracks with **70 or higher popularity <u>in the same genre</u>** and slowly decremented only if there are not enough tracks to create a playlist.

**Similarity method**: **Euclidean Distance**.

| Song | Valence | Energy | Danceability | Euclidean Distance to the chosen track |
|---|---|---|---|---|
| Save Your Tears (Chosen) | 0.64 | 0.83 | 0.68 | 0 |
| Rap God | 0.63 | 0.84 | 0.71 | 0.0331 |
| Just A Little Step | 0.67 | 0.85 | 0.69 | 0.0374 |
| So Much Better | 0.62 | 0.86 | 0.72 | 0.0538 |
| Love Letter | 0.69 | 0.86 | 0.66 | 0.0616 |
| Nothing But Trouble | 0.7 | 0.81 | 0.68 | 0.0632 |
| Don’t Tread On Me | 0.68 | 0.88 | 0.66 | 0.067 |
| Killing Boys | 0.68 | 0.9 | 0.69 | 0.0812 |
| Berzerk | 0.68 | 0.87 | 0.74 | 0.0824 |
| Unity | 0.57 | 0.88 | 0.71 | 0.0911 |

*Table 1: Nearest Neighbor Tracks with “Save You Tears” as the chosen track*

***Step 3: Smooth-transition Playlist Generation***

- After the playlist has been created in **Step 2**, smooth-transition playlist generation is performed. Making a smooth-transition playlist is curating a playlist that transitions smoothly from one song to the next based on the similarity between consecutive songs. The smoothest transition playlist will be the playlist with the lowest sum of all distances between two consecutive songs in the playlist. Therefore, ***Repeated Nearest Neighbor (RNN)*** and ***Genetic Algorithm (GA)*** have been used to solve this problem. More technical details will be explained in my ***Custom Project Report.***

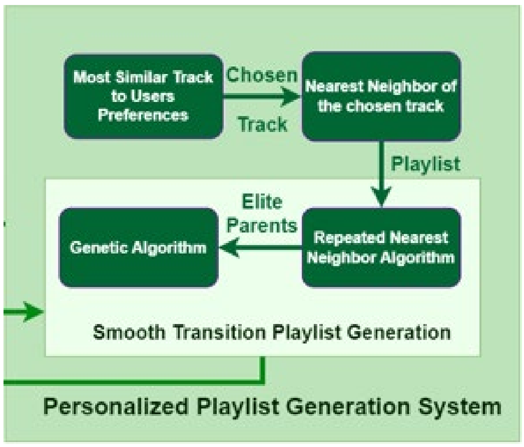

Most Similar Track to Users Preferences
Chosen Track
Nearest Neighbor of the chosen track
Playlist
Elite Parents
Genetic Algorithm
Repeated Nearest Neighbor Algorithm
Smooth Transition Playlist Generation
Personalized Playlist Generation System

*Figure 16: Personalized Playlist Generation System Structure Chart.*

## III. CONCLUSION

In conclusion, the development and analysis of the music player represent a cohesive integration of advanced algorithms and user-centric design. By incorporating basic playback functionalities, playlist creation and modifications, and advanced playlist generation using sophisticated algorithms, the music player offers a pleasing user listening experience. Overall, this report outlines the successful implementations of cutting-edge technology and intuitive design, serving as a testament to my efforts in creating a **High Distinction** custom program.