A Python
Music Player

MELODIA

Grade 11 Computer Science
Project file

Prepared By :
Adeeb S Rahman
Nazia Niha Kaesh
Subham Sharma

Github Repository

Table of Contents

Github Repository

Acknowledgements

The successful completion of this project by our team has been made possible through the invaluable support extended by various individuals.

Firstly, we express our sincere gratitude to our esteemed teacher, Mrs. Charu Negi, whose guidance and profound insights played an instrumental role in shaping the development of Melodia. Her constructive feedback and thoughtful suggestions have significantly contributed to refining and enhancing the project to its current polished state.

Additionally, we extend our appreciation to our classmates, whose unwavering encouragement, commendations, and constructive critiques propelled us forward throughout the project's evolution. Their active involvement in testing and diligent assistance in identifying and resolving bugs at various stages have been indispensable to the project's success.

We would like to express our sincere appreciation to John Elder from codemy.com for his outstanding tutorials, which played a vital role in the development of our project. His instructional content provided invaluable insights, guidance, and a solid foundation that greatly contributed to our understanding and successful implementation. We are grateful for the wealth of knowledge he shared, enriching our project and enhancing our skills in the process.

Lastly, we extend our heartfelt thanks to the reader for demonstrating a keen interest in our project. Your attention is a testament to the significance of our work, and we are truly honored by your engagement.

Melodia

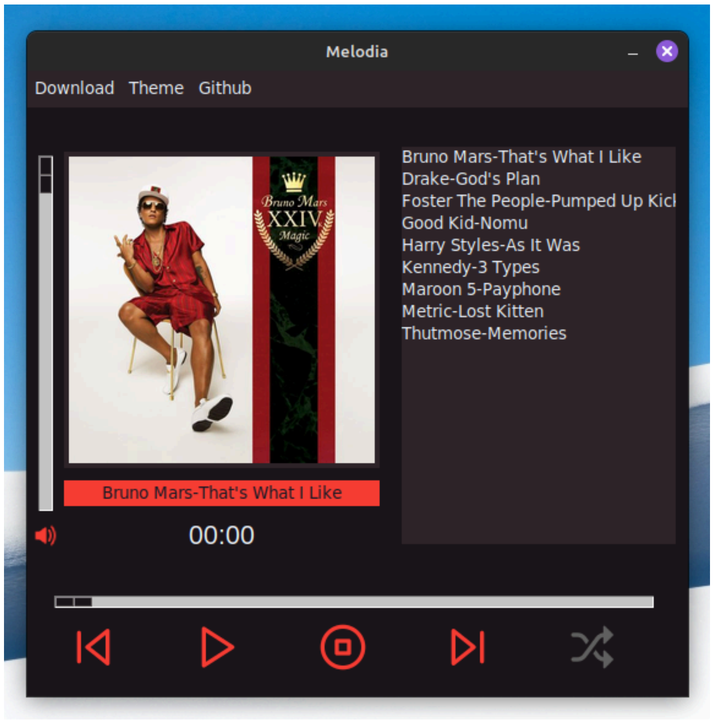
1: Introduction

1.1 : Abstract

Melodia is a python based, feature-rich music playing application that effectively serves as an offline mp3 player, providing users with a wide range of functionalities. It uses the Deezer API and youtube to provide users a way to listen to their favorite songs offline, by downloading the song on the device.It has a easy-to-use, modern UI.

The main features of the application include::

- Provide all standard issue features of a general MP3 player (pause, play, skip, autoplay, etc.)
- Ability to search and save music locally.
- Play, Create and Edit Playlists
- Interact with the app through a modern
- Various themes for user customization



Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories

Caption : Preview of Main Screen with "Magma" theme

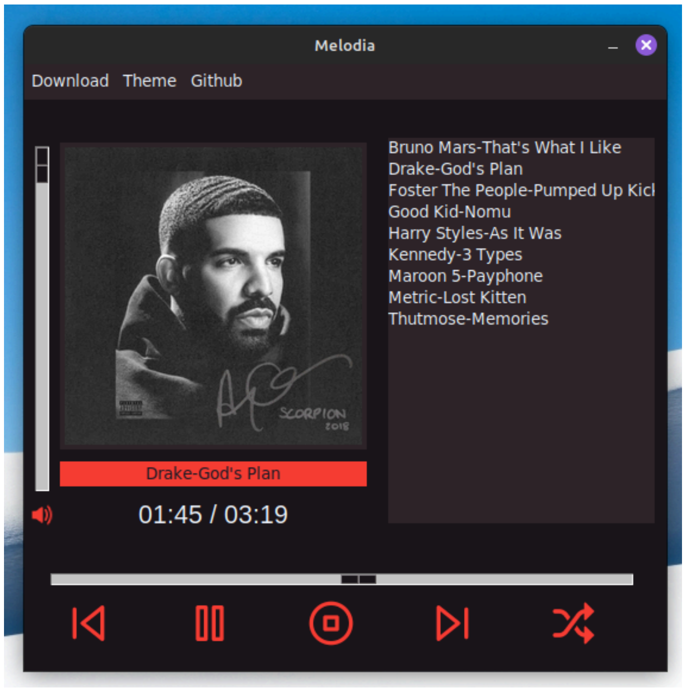
2 : Front End

The program is fully based on python's built-in Tkinter module. This was chosen due to ease of development, light-weight functionality, and user friendliness.

2.1 : Screens

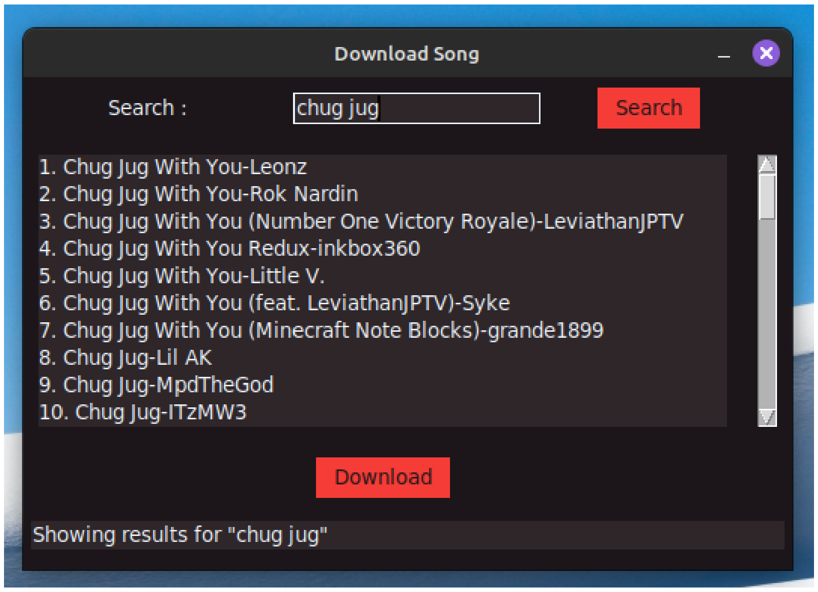
The program consists of two screens:

1) Main Screen : Where users can see their downloaded songs and play them on their device,



Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories

2) Download Screen : Where users can search for more songs to download to their device.



1. Chug Jug With You-Leonz
2. Chug Jug With You-Rok Nardin
3. Chug Jug With You (Number One Victory Royale)-LeviathanJPTV
4. Chug Jug With You Redux-inkbox360
5. Chug Jug With You-Little V.
6. Chug Jug With You (feat. LeviathanJPTV)-Syke
7. Chug Jug With You (Minecraft Note Blocks)-grande1899
8. Chug Jug-Lil AK
9. Chug Jug-MpdTheGod
10. Chug Jug-ITzMW3

Showing results for "chug jug"

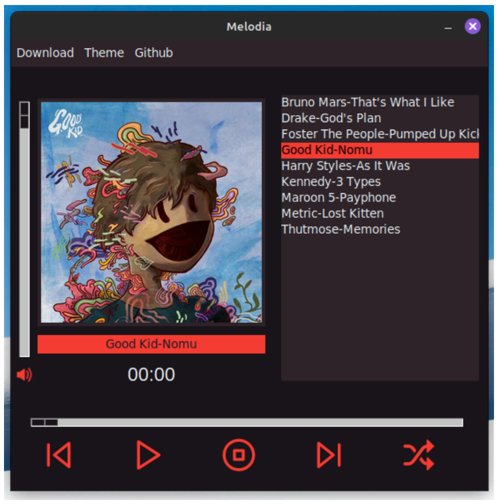
2.2 : Themes

The application has various themes which add a sense of customization and personalization to the user experience. They can be changed dynamically. The themes include:

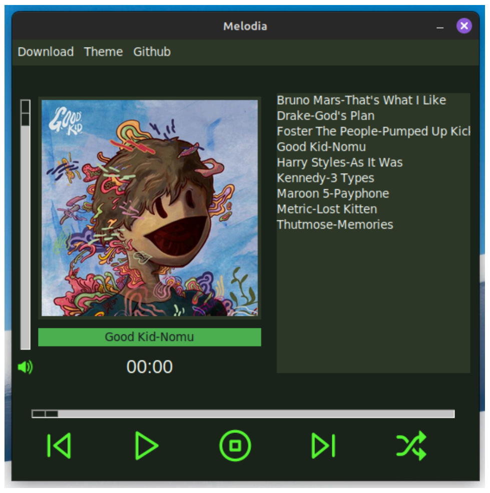
● Magma	: Red coloured theme (Top-Left)
● Lush	: Green coloured theme (Top-Right)
● Moonlit	: Blue coloured theme (Bottom-Left)
● Nebula	: Purple coloured theme (Bottom-Right)

(These themes also carry over to the download page)

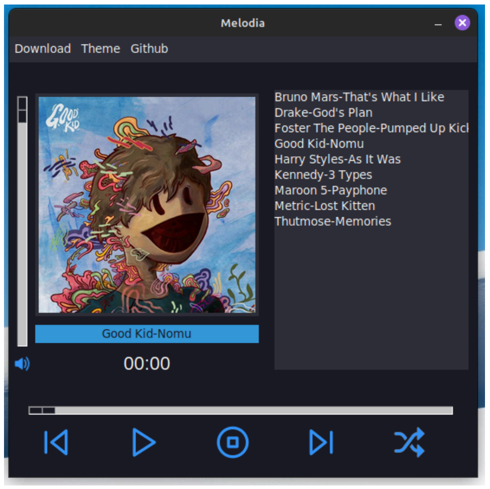
The themes are shown below:



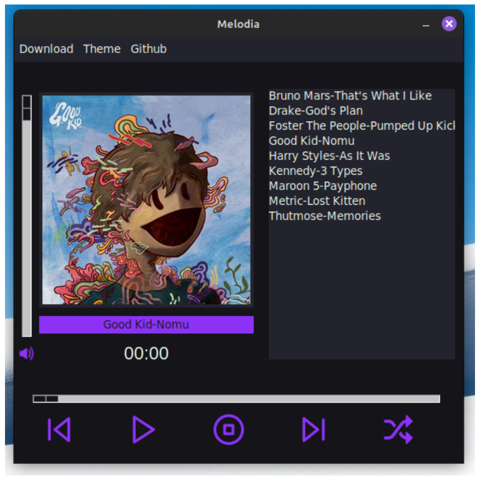
Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories



Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories



Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories



Bruno Mars-That's What I Like
Drake-God's Plan
Foster The People-Pumped Up Kicl
Good Kid-Nomu
Harry Styles-As It Was
Kennedy-3 Types
Maroon 5-Payphone
Metric-Lost Kitten
Thutmose-Memories

3 : Back End

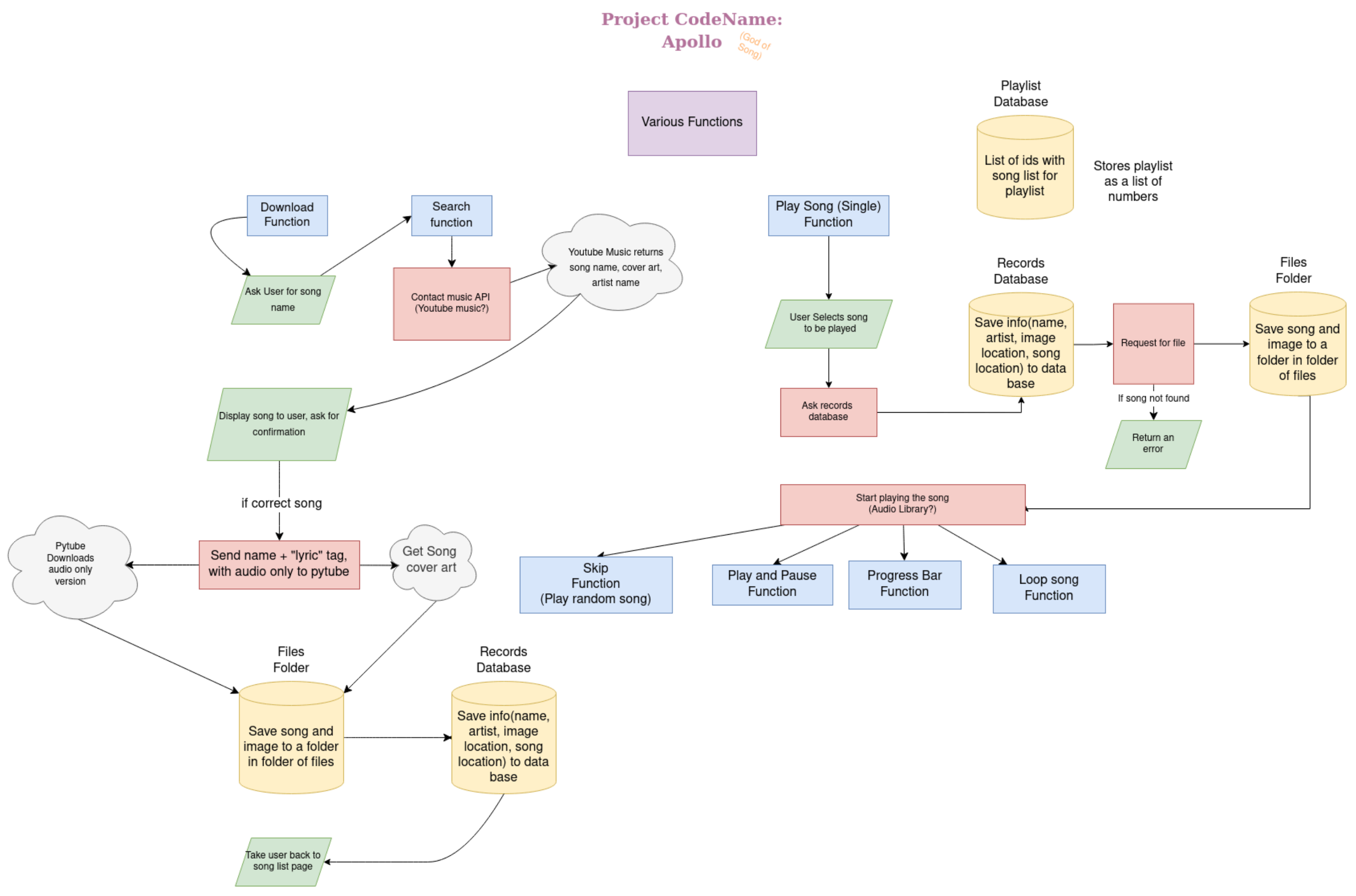
3.1 : Flow of Application

The following is a breakdown of how our back end works :

- **Overview of Scripts:**
  - The project consists of two Python scripts: `main.py` and `download.py`.
  - `main.py` is responsible for the graphical user interface (GUI) using Tkinter.
  - `download.py` manages the download and conversion of music tracks.
- **API Integration:**
  - The Deezer API is used to fetch information about artists and tracks.
- **User Interaction:**
  - Users interact with a Tkinter GUI that prompts them to search for an artist.
- **Download Process:**
  - `download.py` takes care of downloading the chosen track from YouTube.
  - It then converts the downloaded MP4 file to MP3 using the `moviepy` library.
- **File Organization:**
  - The resulting MP3 file is stored in the "./music/" directory.
  - The script also downloads the associated album cover, placing it in "./music/albumCover/".
- **User Instructions:**
  - To use the application, users run `main.py` and provide their Deezer API credentials.
  - The GUI allows users to search for artists, select tracks, and initiate the download process.

3.2 : Graphical Flowchart

The following is a flowchart to explain the functionalities of the app.


Project CodeName: Apollo
(God of Song)
Various Functions
Download Function
Search function
Ask User for song name
Contact music API (Youtube music?)
Youtube Music returns song name, cover art, artist name
Display song to user, ask for confirmation
if correct song
Send name + "lyric" tag, with audio only to pytube
Pytube Downloads audio only version
Get Song cover art
Files Folder
Save song and image to a folder in folder of files
Records Database
Save info(name, artist, image location, song location) to data base
Take user back to song list page
Play Song (Single) Function
User Selects song to be played
Ask records database
Playlist Database
List of ids with song list for playlist
Stores playlist as a list of numbers
Records Database
Save info(name, artist, image location, song location) to data base
Request for file
If song not found
Return an error
Files Folder
Save song and image to a folder in folder of files
Start playing the song (Audio Library?)
Skip Function (Play random song)
Play and Pause Function
Progress Bar Function
Loop song Function

3.3 : Libraries Used

Modules from the following libraries are used. Information is provided with each to explain its function.

tkinter:	This module is used for creating graphical user interfaces (GUIs) of the main window, buttons, labels, and other GUI elements.
pygame:	This module is used for handling audio playback.
PIL(Python Imaging Library, now known as Pillow):	This module is used for handling images. In this code, it's used for resizing and displaying album cover images.
os:	This module provides a way to interact with the operating system. It's used for file and directory operations, such as checking if a file or directory exists, opening a folder, deleting files, etc.
time:	This module provides various time-related functions. In this code, it's used for formatting time durations.
tkinter.ttk:	This submodule of tkinter provides a themed widget set used for styling the ttk.Scale widget (slider).
mutagen.mp3:	This module is used for reading metadata from MP3 files to get information about the length of a song.
tkinter.messagebox:	This submodule of tkinter is used for displaying message boxes.
download:	It contains functions related to downloading songs.
webbrowser:	This module is used to open a GitHub page.
sys:	This module is used to determine the platform (OS) the program is running on.
random:	This module is used for selecting a random song when in shuffle mode.
deezer:	This module is used to access information about artists, albums, tracks, and playlists on Deezer using deezer API.
json:	This module is used to store song lists and encode JSON format (serialization) and decoding JSON data into Python objects (deserialization).
requests:	This module is used for accessing deezer API and fetching

	URLs.
pytube and pytube.search:	This module is used to find song on youtube to be downloaded and downloads it
moviepy.editor	This module is part of the MoviePy library, which is used for converting the downloaded mp4 file into a mp3 file only
shutil:	This module is used to make folders for songs, etc.
urllib:	This module is used to get images from url.
dotenv:	It is used to store API information

Note : Deezer API credentials are securely stored in a `.env` file. To use the application, users run `main.py` and provide their Deezer API credentials.

3.4 : File Structure

The various files and directories are explained below :

./Config	color.txt is used for COLOR of slider in GUI
./Music	Contains the mp3 files of the songs downloaded by the user
./Music/albumcover	It is a subfolder of music containing all the album cover images of downloaded songs.
./Sources	Contains images of the app logo, button icons, placeholder images, etc.
./Sources/ctrlbtn	Contains image of all the buttons used in the GUI
./Archive	Contains “Legacy Versions” of code for retrospective
./Reference	This folder contains various files which are used as reference when coding, such as flow chart, font list, etc
.env_sample	A sample file. This directs the user on how the .env file should look like.

Requirements.txt	All necessary libraries to run the application. This has been put here for ease of installation.

3.5 : Main Functions

Following is a table explaining the functions used in the code.

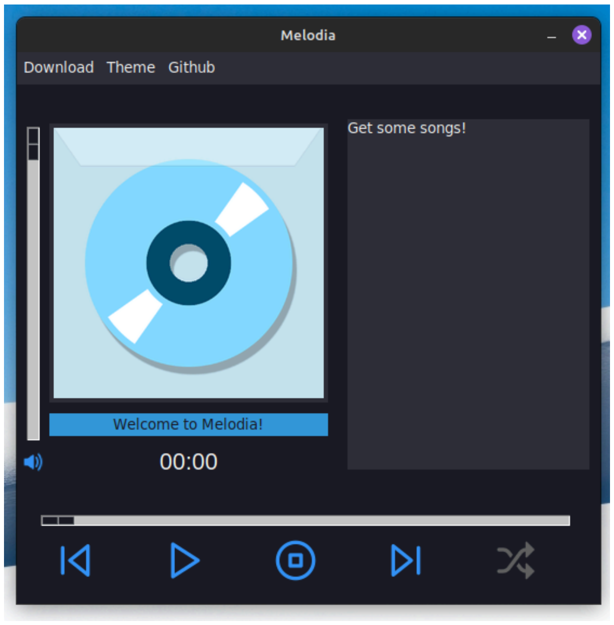
master():	updates the song duration on the GUI, handles the play state, controls shuffle and autoplay.
songLengthGrabber():	Retrieves the length of the currently playing song using the Mutagen library.
changeName():	Updates the displayed name of the currently playing track in the GUI.
stop():	Stops the currently playing song, cancels scheduled updates, and resets the play state.
shuffleBtnFunc():	Toggles the shuffle state and updates the shuffle button's image accordingly.
randSelect():	Selects a random track from the listbox, excluding the currently playing track, the one before, and the one after.
mainBtnFunc(mainQuery):	Controls the play/pause functionality of the main button.
nextTrack(move):	Plays the next or previous track based on the movement direction.(-1 or 1 respectively)

changeCover(trackNum):	Changes the album cover image based on the selected track.
delSong():	Stops the currently playing song, deletes the selected song and its cover, updates the GUI, and moves to the next track.
getSongName(path):	Converts a file path to a readable song name.
getSongPath(name):	Converts a song name to a file path.
getSongCov(name):	Converts a song name to an album cover image file path.
slide(pos):	Controls the slider for song duration and seeks the song to the specified position.
openFolder():	Opens the music folder.
changeColor(scheme):	Changes the color scheme of the GUI based on the selected theme.
reloadTracks():	: Reloads the list of tracks in the GUI.
volSliderFunc(x):	Controls the volume slider functionality and sets the volume for the Pygame mixer.
openGithub():	Opens the GitHub page of the project in the user's default browser.

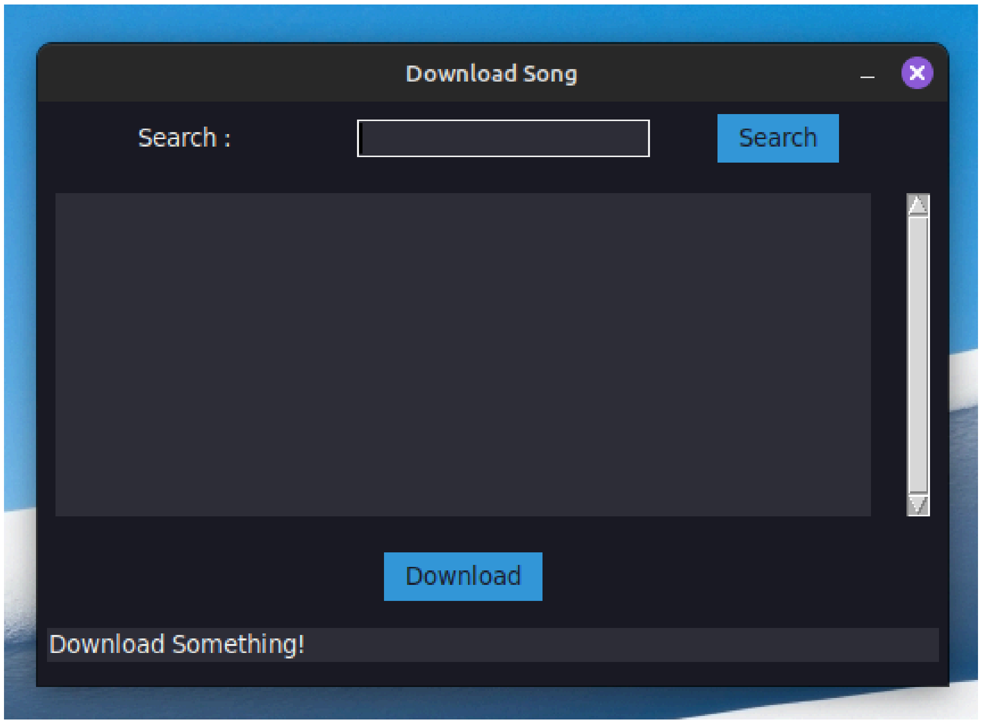
4 : Screenshots

4.1 : Flow of User Experience

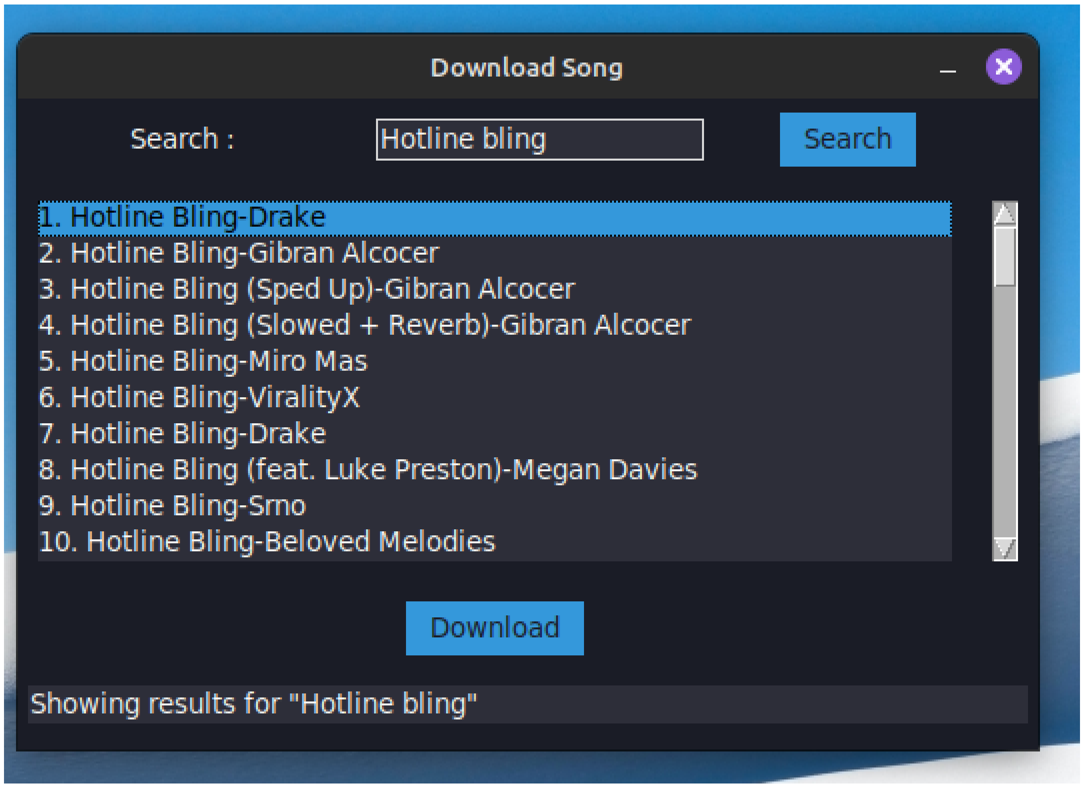
1. The main screen upon installation :



2. Let's download a song!
   Download > Get Song



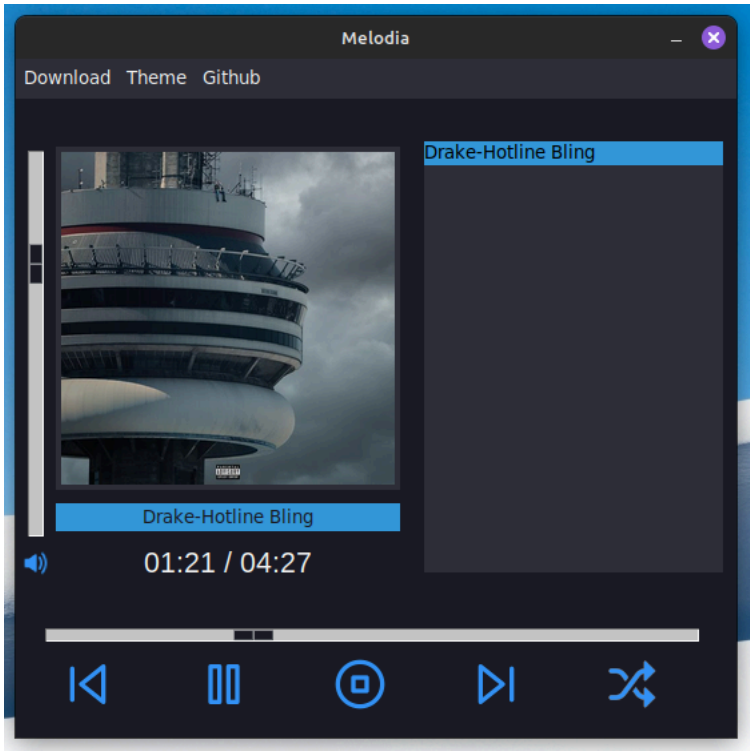
3. Searching and Downloading



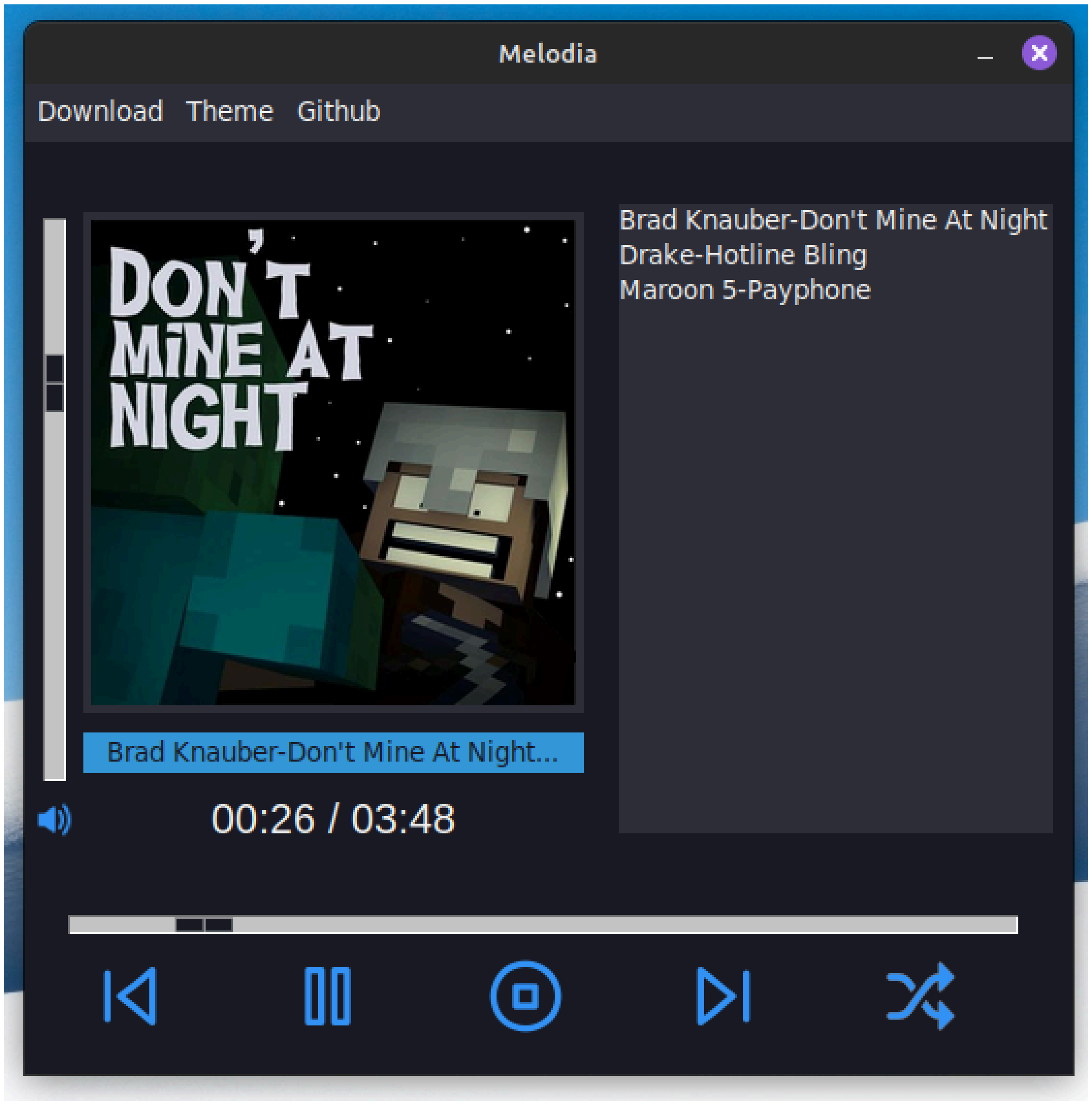
1. Hotline Bling-Drake
2. Hotline Bling-Gibran Alcocer
3. Hotline Bling (Sped Up)-Gibran Alcocer
4. Hotline Bling (Slowed + Reverb)-Gibran Alcocer
5. Hotline Bling-Miro Mas
6. Hotline Bling-ViralityX
7. Hotline Bling-Drake
8. Hotline Bling (feat. Luke Preston)-Megan Davies
9. Hotline Bling-Smo
10. Hotline Bling-Beloved Melodies

Showing results for "Hotline bling"

4. Now we can play it!



5. We can take advantage of the shuffle play/ Autoplay if we have many songs



Brad Knauber-Don't Mine At Night
Drake-Hotline Bling
Maroon 5-Payphone

Brad Knauber-Don't Mine At Night...

00:26 / 03:48

5 : Code

5.2 : Melodia.py

```
# Import necessary libraries
from tkinter import *
import pygame
from PIL import ImageTk, Image
import os
import time
from mutagen.mp3 import MP3
import tkinter.ttk as ttk
from tkinter import messagebox
import download
import webbrowser
from sys import platform
import random

# Define color constants
bgMain = "#1A1C26"
bgSec = "#2D2E39"
fgMain = "#F4F4F2"
accent = "#3498DB"

global USER_OS
OS_LINUX=1
OS_MAC=2
OS_WIN=3
#Identify operating system
if platform == "linux":
      USER_OS = OS_LINUX
      #Optimal Trackbox width is different
      # for different operating systems
      TBWidth= 28
elif platform == "darwin":
      USER_OS = OS_MAC
elif platform == "win32":
      USER_OS = OS_WIN
      TBWidth= 35

#Create the main tkinter screen
screen = Tk()
```

```
screen.title("Melodia")
screen.configure(bg=bgMain)

#Modify the width of ttk slider (its too fat)
s = ttk.Style()
s.configure("Horizontal.TScale",sliderthickness=8)
s.configure("Vertical.TScale",background=bgMain)
s.configure("Horizontal.TScale",background=bgMain)

# Import and display program icon
img = PhotoImage(file="./sources/icon.gif")
screen.tk.call("wm", "iconphoto", screen._w, img)
screen.iconphoto(True, img)

# Set window properties
screen.resizable(0, 0)
# Default 330x550
screen.geometry("540x480")

# Load control button images
global playBtnImg, pauseBtnImg, stopBtnImg, frontBtnImg, backBtnImg,
shuffleBtnImg, volIcon
playBtnImg = PhotoImage(file="./sources/ctrlbtn/playBtnImgBlue.png")
pauseBtnImg = PhotoImage(file="./sources/ctrlbtn/pauseBtnImgBlue.png")
stopBtnImg = PhotoImage(file="./sources/ctrlbtn/stopBtnImgBlue.png")
frontBtnImg = PhotoImage(file="./sources/ctrlbtn/frontBtnImgBlue.png")
backBtnImg = PhotoImage(file="./sources/ctrlbtn/backBtnImgBlue.png")
shuffleBtnImg = PhotoImage(file="./sources/ctrlbtn/shuffleBtnImgBlue.png")
volIcon = PhotoImage(file="./sources/ctrlbtn/volContBlue.png")

#Shuffle button greyed out
global shuffleBtnImgGray
shuffleBtnImgGray =
PhotoImage(file="./sources/ctrlbtn/shuffleBtnImgGray.png")


# Initialize global variable for play state, as well as what play state
means
global playState
SONG_NOT_PLAYING = 0
SONG_IS_PLAYING = 1
SONG_IS_PAUSED = 2
```

```
# Initially, we dont want teh song to be playing
playState = SONG_NOT_PLAYING

#Global variable for shuffle
global shuffleState
shuffleState = False

# Initialize the Pygame mixer
pygame.mixer.init()

# Various functions used in the program
# Function to change song duration (and auto play)
def master():
	global tracks
	global playState

	# If the song is stopped, dont do all this
	if playState == SONG_NOT_PLAYING:
	return

	# Grab current time, edit the duration text (as integer)
	currentDur = pygame.mixer.music.get_pos() / 1000

	# Sometimes there is a error when song is played, so fix it
	if currentDur < 0:
	currentDur = 0

	# Convert it into proper format
	convCurrentDur = time.strftime("%M:%S", time.gmtime(currentDur))

	# Getting the total time
	songLen = songLengthGrabber()

	# Convert again
	convTotLen = time.strftime("%M:%S", time.gmtime(songLen))

	# Small bug where current dur is not responsive of the first second
	currentDur += 1

	# Change the position and size of slider
	slider.config(to_=songLen, value=int(slider.get()))

	# If the song has finished playing
```

```
        if int(slider.get()) == int(songLen):
        # Show that the song is complete
        durLabel.config(text=f"{convTotLen} / {convTotLen}")

        #if shuffle mode is on, go to a random song
        if shuffleState:
              randSelect()
        else:
        #Otherwise, move on to the next song
              nextTrack(1)

        # Else, if its paused, we dont want to do anything
        elif playState == SONG_IS_PAUSED:
        pass

        # If the slider hasnt moved
        elif int(slider.get()) == int(currentDur):
        # Change text
        durLabel.config(text=f"{convCurrentDur} / {convTotLen}")
        # Change the position of the slider to the current position
        slider.config(to_=songLengthGrabber(), value=int(currentDur))

        else:
        # We need the position of the slider in time format
        sliderConv = time.strftime("%M:%S", time.gmtime(int(slider.get())))

        durLabel.config(text=f"{sliderConv} / {convTotLen}")

        # Manually move the slider
        slider.config(value=(slider.get() + 1))

        # Run this again and again after 1 second
        global secLoop
        secLoop = screen.after(1000, master)


# Get length of song length
def songLengthGrabber():
        # Get file path of currently playing song
        track = trackBox.get(ACTIVE)
        track = getSongPath(track)

        # Read song length with mutagen
```

```
      songMutagen = MP3(track)
      songLength = songMutagen.info.length
      # Convert to proper format
      return songLength


# Function to change name of track
def changeName():
      currentPlaying = trackBox.get(ACTIVE)
      name = getSongName(currentPlaying)
      # If song name is longer than 32 chars, shorten it
      if len(name) > 32:
      name = name[:32] + "..."

      curTitle.configure(text=name)


# Function to stop the music
def stop():
      global playState

      #If the song is already stopped, we error handle
      if playState == SONG_NOT_PLAYING:
      return
      # Check to see if folder is empty
      if emptyFolder:
      messagebox.showerror(
            "Error: No Songs!",
            "Looks like you don't have any songs downloaded! Please
download some",
      )
      return

      global secLoop

      # Stop the song in mixer
      pygame.mixer.music.stop()

      # Cancel the currently playing job
      screen.after_cancel(secLoop)

      # Change playState
      playState = SONG_NOT_PLAYING
```

```
	# Change all GUI elements
	mainBtn.configure(image=playBtnImg)
	mainBtn.photo = playBtnImg
	slider.config(value=0)
	durLabel.config(text=f"00:00")

#Functionality for shuffle button
def shuffleBtnFunc():
	global shuffleState

	#If shuffle True, set shuffle false
	if shuffleState:
	shuffleState = False
	shuffleBtn.config(image=shuffleBtnImgGray)
	#print("Shuffle Off")

	#If shuffle False, set shuffle ture
	else:
	shuffleState = True
	shuffleBtn.config(image=shuffleBtnImg)
	#print("Shuffle On")

#Random list box selector
def randSelect() :
	global playState
	#Get a random index value for the listbox
	#Make this not the same as current selection, the song before, or
song after (doesnt FEEL random)
	rand = random.randint(0,trackBox.size()-1)
	while rand == trackBox.curselection()[0] or rand ==
trackBox.curselection()[0]-1 or rand == trackBox.curselection()[0]+1:
	rand = random.randint(0,trackBox.size()-1)
	#Clear current selection
	trackBox.selection_clear(0, END)
	#Set selection to our random one
	trackBox.selection_set(rand) # Set the index
	#Activate the random one
	trackBox.activate(rand)

	# Get the name of the next track, modify for file path, and update
listbox selection
	playTrack = trackBox.get(rand)
	playTrack = getSongPath(playTrack)

	# Change the album cover based on the selected track
	changeCover(rand)

	# Change the current title name, current play time
	changeName()
	durLabel.config(text=f"00:00")

	#Set slider back to zero
	slider.config(value=0)

	# Cancel the currently playing job
	screen.after_cancel(secLoop)

	# Reset play state and simulate a button click to start playing the
new track
	playState = 0
	mainBtnFunc(0)


# A function that controls teh working of the main play button
def mainBtnFunc(mainQuery):
	# Check to see if folder is empty
	if emptyFolder:
	messagebox.showerror(
		"Error: No Songs!",
		"Looks like you don't have any songs downloaded! Please
download some",
	)
	return

	global playState, tracks

	# Copy the current play state to the local variable mainQuery
	mainQuery = playState

	# If the play state is 0 (initial or stopped)
	if playState == SONG_NOT_PLAYING:
	# Pause the music (if playing), #print a message, and get the
selected track
	pygame.mixer.music.pause()
	# print("Play pressed first time")
```

```
        track = trackBox.get(ACTIVE)

        # Modify track name for file path and find its index in the tracks
list
        track = getSongPath(track)
        trackIndex = tracks.index(track.replace("./music/", ""))

        #Send slider to 0
        slider.config(value=0)

        # Load and play the selected track, update play state, and change
album cover
        pygame.mixer.music.load(track)
        pygame.mixer.music.play(loops=0)
        playState = SONG_IS_PLAYING
        changeCover(trackIndex)

        mainBtn.configure(image=pauseBtnImg)
        mainBtn.photo = pauseBtnImg

        # Change the current title name
        changeName()

        # Run the song duration fucntion when first played
        master()

        # If the play state is 1 (playing)
        elif playState == SONG_IS_PLAYING:
        # Pause the music,
        # print a message, update play state, and change button image
        pygame.mixer.music.pause()
        playState = SONG_IS_PAUSED
        mainBtn.configure(image=playBtnImg)
        mainBtn.photo = playBtnImg

        # If the play state is 2 (paused)
        elif playState == SONG_IS_PAUSED:
        # Unpause the music, #print a message, update play state, and change
button image
        pygame.mixer.music.unpause()
        playState = SONG_IS_PLAYING
        mainBtn.configure(image=pauseBtnImg)
        mainBtn.photo = pauseBtnImg
```

```
# Function to play the next or previous track
def nextTrack(move):
      # Check to see if folder is empty
      if emptyFolder:
      messagebox.showerror(
            "Error: No Songs!",
            "Looks like you don't have any songs downloaded! Please
download some",
      )
      return
      try:
      # Get the index of the currently selected track in the listbox
      curTrack = trackBox.curselection()[0]
      except IndexError:
      messagebox.showerror(
            "Error: Select a song!", "You have to click on the track you
want to play"
      )
      return

      # Stop the currently playing song
      stop()

      # If shuffle is on, just move to a shuffled song
      if shuffleState:
      randSelect()
      return

      global playState

      # Calculate the index of the next track based on the movement
direction
      if curTrack == 0 and move == -1:
      nextTrack = trackBox.size() - 1
      elif curTrack == (trackBox.size() - 1) and move == 1:
      nextTrack = 0
      else:
      nextTrack = trackBox.curselection()[0] + move

      # Get the name of the next track, modify for file path, and update
listbox selection
```

```
	playTrack = trackBox.get(nextTrack)
	playTrack = getSongPath(playTrack)

	trackBox.selection_clear(0, END)
	trackBox.activate(nextTrack)
	trackBox.selection_set(nextTrack, last=None)

	# Change the album cover based on the selected track
	changeCover(nextTrack)

	# Reset play state and simulate a button click to start playing the
new track
	playState = 0
	mainBtnFunc(0)

	# Change the current title name
	changeName()


# Function to change the album cover image based on the selected track
def changeCover(trackNum):
	global curCover
	albumCover = tracks[trackNum].replace(".mp3", "")
	albumCover = f"./music/albumCover/{albumCover}-cover.jpg"
	curCover = Image.open(albumCover)
	curCover = curCover.resize((250, 250), Image.LANCZOS)
	curCover = ImageTk.PhotoImage(curCover)
	curCoverLabel.configure(image=curCover)

#Function to delete a song
def delSong():
	#Stop currently playing song
	stop()

	#Get path for song and cover
	curSelect = trackBox.curselection()
	if not curSelect:
	messagebox.showerror(
		"Select Song!",
		"You must select the song to delete it"
		)
	return
```

```
      selected = trackBox.get(curSelect[0])
      selectedPath = getSongPath(selected)

      selectedCoverPath = selectedPath.replace("./music/" ,
"").replace(".mp3", "")
      selectedCoverPath =
f"./music/albumCover/{selectedCoverPath}-cover.jpg"

      #Delete cover, then image
      os.remove(selectedCoverPath)
      os.remove(selectedPath)

      #Change album cover
      curCover = Image.open("./sources/template.png")
      curCover = curCover.resize((250, 250), Image.LANCZOS)
      curCover = ImageTk.PhotoImage(curCover)
      curCoverLabel.configure(image=curCover)

      #Garbage collection
      curCoverLabel.photo=curCover

      #Reload track list
      reloadTracks()

      # #Move to next track
      # nextTrack(1)

# Function to get song name
def getSongName(path):
      name = path.replace(".mp3", "")
      name = name.replace("_", " ")

      return name


# Function to get song path
def getSongPath(name):
      name = name.replace(" ", "_")
      path = f"./music/{name}.mp3"
      return path


# Function to get song album path
```

```
def getSongCov(name):
      path = name.replace(".mp3", "")
      path = path.replace(" ", "_")
      path = f"./music/albumCover/{path}-cover.jpg"
      return path


# slider function
def slide(pos):

      if playState != SONG_IS_PLAYING:
      pass
      else:
      track = trackBox.get(ACTIVE)
      track = getSongPath(track)
      # # Load and play the selected track, update play state, and change
album cover
      pygame.mixer.music.load(track)

      curPos = slider.get()
      pygame.mixer.music.play(loops=0, start=int(curPos))
      slider.config(value=curPos)

#Open folder
def openFolder():
      print(USER_OS)
      if USER_OS==OS_LINUX:
      os.system('xdg-open "%s"' % "./music/")
      elif USER_OS==OS_WIN:
      os.startfile('.\\music')
      elif USER_OS==OS_MAC:
      pass

# Function to change color scheme
def changeColor(scheme):
      global playState
      # Change the color pallette
      # Col is for changing file path to button icons
      if scheme == "BLUE":
      bgMain = "#1A1C26"
      bgSec = "#2D2E39"
      fgMain = "#F4F4F2"
      accent = "#3498DB"
```

```
col = "Blue"

elif scheme == "GREEN":
bgMain = "#19231A"
bgSec = "#2E392A"
fgMain = "#F2F8F2"
accent = "#4CAF50"
col = "Green"

elif scheme == "RED":
bgMain = "#1C161A"
bgSec = "#2E2629"
fgMain = "#EDF3FA"
accent = "#F53C36"
col = "Red"

elif scheme == "PURPLE":
bgMain = "#16181C"
bgSec = "#27262E"
fgMain = "#EEFAED"
accent = "#8F36F5"
col = "Purple"

# Change elements that use the background color
for i in bgMainBgList:
i.configure(bg=bgMain)

# Change elements that use the foreground color
for i in fgMainFgList:
i.configure(fg=fgMain)

# Change elements that use the secondary background color
for i in bgSecBgList:
i.configure(bg=bgSec)

# Change elements that use the accent color
# (not in a loop because we have to change different things)
trackBox.configure(selectbackground=accent)
curTitle.configure(bg=accent)

#Change cur cover border color
curCoverLabel.config(highlightbackground=bgSec)
```

```
      # Change the file we're using for the images (it will now default to
these)
      global playBtnImg, pauseBtnImg, stopBtnImg, frontBtnImg, backBtnImg,
shuffleBtnImg,volIcon

      playBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/playBtnImg{col}.png")
      pauseBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/pauseBtnImg{col}.png")
      stopBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/stopBtnImg{col}.png")
      frontBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/frontBtnImg{col}.png")
      backBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/backBtnImg{col}.png")
      shuffleBtnImg =
PhotoImage(file=f"./sources/ctrlbtn/shuffleBtnImg{col}.png")
      volIcon= PhotoImage(file=f"./sources/ctrlbtn/volCont{col}.png")

      # Change the button images

      #If song is playing, pause button image should be used
      if playState == SONG_IS_PLAYING:
      mainBtn.configure(image=pauseBtnImg)
      #Stop Garbage collection
      mainBtn.photo = pauseBtnImg
      else:
      #Otherwise use play button
      mainBtn.configure(image=playBtnImg)
      #Stop Garbage collection
      mainBtn.photo = playBtnImg

      #if shuffle is on, change the image
      if shuffleState:
      shuffleBtn.configure(image=shuffleBtnImg)
      #Stop Garbage collection
      shuffleBtn.photo = shuffleBtnImg

      stopBtn.configure(image=stopBtnImg)
      frontBtn.configure(image=frontBtnImg)
      backBtn.configure(image=backBtnImg)
      volSliderLabel.configure(image=volIcon)
```

```
    # Stop garbage collection (if this code is not here,
    # the loaded images are deleted before they are used)
    mainBtn.photo = playBtnImg
    stopBtn.photo = stopBtnImg
    frontBtn.photo = frontBtnImg
    backBtn.photo = backBtnImg
    volSliderLabel.photo = volIcon

    #Change the background color of the slider
    s.configure("Vertical.TScale",background=bgMain)
    s.configure("Horizontal.TScale",background=bgMain)

    # Change this to new default
    defaultColor = open("./config/COLOR.txt", "r+")
    # Erase the file
    defaultColor.truncate(0)
    # Write new default
    defaultColor.write(scheme)
    defaultColor.close()


# Function to reload the track box
def reloadTracks():
    # Redefine tracks list
    global tracks
    tracks = []
    for name in os.listdir("./music"):
    if name in [".gitignore", "albumCover"]:
        continue
    tracks.append(name)
    tracks = sorted(tracks)

    global emptyFolder
    if len(tracks) == 0:
    tracks = ["Get some songs!"]
    # This variable tracks if the directory is empty
    emptyFolder = True
    else:
    emptyFolder = False

    # Clear current trackbox
    trackBox.delete(0, "end")
```

```
      # Add everything back
      for name in tracks:
      name = name.replace(".mp3", "")
      name = name.replace("_", " ")
      trackBox.insert("end", name)

#Volume slider funtion
def volSliderFunc(x):
      vol= volSlider.get()
      pygame.mixer.music.set_volume(volSlider.get())

#volSliderLabel.config(text=f"{int(pygame.mixer.music.get_volume()*100)}%")
      # printpygame.mixer.music.get_volume()
      # pass

#Link to github page
def openGithub():
      webbrowser.open('https://github.com/Arctican4Real/Melodia')

#Auto play capability

### UI CODE ###

# Code for main menu bar
settings = Menu(screen, bg=bgSec, fg=fgMain, bd=0)
screen.config(menu=settings)

# Code for download button on menu bar
downloadMenu = Menu(settings, bg=bgMain, fg=fgMain, bd=0,tearoff="off")
settings.add_cascade(label="Download", menu=downloadMenu)

# Button to download songs
downloadMenu.add_command(label="Get Song",
command=lambda:download.downloadSong(screen))
# Button to reload the tracks
downloadMenu.add_command(label="Reload Tracks", command=reloadTracks)
# Add folder button
downloadMenu.add_command(label="Open Folder", command=openFolder)
# Delete song
downloadMenu.add_command(label="Delete Song", command=delSong)


# Code for themes button on menu bar
```

```
themeMenu = Menu(settings, bg=bgMain, fg=fgMain, bd=0, tearoff="off")
settings.add_cascade(label="Theme", menu=themeMenu)
themeMenu.add_command(label="Magma", command=lambda: changeColor("RED"))
themeMenu.add_command(label="Lush", command=lambda: changeColor("GREEN"))
themeMenu.add_command(label="Moonlit", command=lambda: changeColor("BLUE"))
themeMenu.add_command(label="Nebula", command=lambda:
changeColor("PURPLE"))

#Github page link
settings.add_command(label="Github", command=openGithub)

# Frames
left_frame = Frame(screen, bg=bgMain)
left_frame.grid(row=0, column=1, padx=(0,5), pady=(0,0), sticky="ew")

right_frame = Frame(screen, bg=bgMain)
right_frame.grid(row=0, column=2, padx=(0,10), pady=(0,0), sticky="ew")

down_frame = Frame(screen, bg=bgMain)
down_frame.grid(row=1, column=0, padx=22, pady=0,sticky="ew",columnspan=4)

btnDiv = Frame(screen, bg=bgMain)
btnDiv.grid(row=2,column=0,padx=22,pady=10,sticky="ew",columnspan=4)

# Add weight to the rows and columns for right_frame
screen.grid_rowconfigure(0, weight=1)
screen.grid_columnconfigure(1, weight=1)
right_frame.grid_rowconfigure(0, weight=1)
right_frame.grid_columnconfigure(1, weight=1)

# Create a listbox to display tracks
trackBox = Listbox(
      right_frame,
      bg=bgSec,
      fg=fgMain,
      borderwidth=0,
      highlightthickness=0,
      selectbackground=accent,
      selectborderwidth=0,
      width=TBWidth,
      height=18,
      activestyle="none",
      selectmode=SINGLE
)
trackBox.grid(row=0, column=0, sticky="ew", padx=0,pady=(21,30))

# Defualt to the first track in the listbox
trackBox.activate(0)
trackBox.selection_set(0)

# Initially load the tracks
reloadTracks()

# Load the first cover from the folder
if not emptyFolder:
      albumCover = getSongCov(trackBox.get(0))
# If the folder is empty, get default cover
else:
      albumCover = "./sources/template.png"

volSliderFrame=Frame(screen, bg=bgMain)
volSliderFrame.grid(row=0,column=0,padx=5)

volSliderLabel = Label(
      volSliderFrame,
      borderwidth=0,
      highlightthickness=0,
      bd=0,
      bg=bgMain,
      fg=fgMain,
      image=volIcon
      )
volSliderLabel.grid(column=0, row=2)

# Volume control Slider
volSlider = ttk.Scale(
      volSliderFrame,
      from_=1,
      to=0,
      orient=VERTICAL,
      value=100,
      length=290,
      command=volSliderFunc
      )
s=ttk.Style()
s.configure("Vertical.TScale",sliderthickness=10)
```

```python
volSlider.grid(column=0,row=0,pady=(0,10),sticky="ns")

#Cover art
global curCover
curCover = Image.open(albumCover)
curCover = curCover.resize((250, 250), Image.LANCZOS)
curCover = ImageTk.PhotoImage(curCover)

curCoverLabel = Label(left_frame,image=curCover, borderwidth=0,
highlightthickness=4, highlightbackground=bgSec, bg=bgMain)
curCoverLabel.grid(pady=10,column=1,row=0)

# Display the song duration
durLabel = Label(
      left_frame,
      text="00:00",
      borderwidth=0,
      highlightthickness=0,
      bd=0,
      bg=bgMain,
      fg=fgMain,
      width=20,
      height=1,
      font=("Arial", 16),
)

durLabel.grid(row=2,column=1, columnspan=1,pady=10)

# Slider for song duration
slider = ttk.Scale(
      down_frame,
      from_=0,
      to=100,
      orient=HORIZONTAL,
      value=0,
      length=470,
      #command=slide,
)
slider.grid(column=3,pady=0,ipadx=10,columnspan=4)

#Only trigger the slide event when slider stops moving
slider.bind("<ButtonRelease-1>", slide)
```

```
# Text box for song length
if not emptyFolder:
      firstTrack = getSongName(trackBox.get(0))
# Default variable if no songs
else:
      firstTrack = "Welcome to Melodia!"

# Display the current song name
curTitle = Label(left_frame, text=firstTrack, bd=1, bg=accent, fg=bgMain)
curTitle.grid(row=1,column=1, ipady=0, pady=0,sticky="ew")

#Buttons

# Create control buttons
mainBtn = Button(
      btnDiv,
      image=playBtnImg,
      borderwidth=0,
      command=lambda: mainBtnFunc(playState),
      bg=bgMain,
      highlightthickness=0,
      bd=0,
      relief=SUNKEN,
      activebackground=bgSec
)
stopBtn = Button(
      btnDiv,
      image=stopBtnImg,
      borderwidth=0,
      command=stop,
      bg=bgMain,
      highlightthickness=0,
      bd=0,
      relief=SUNKEN,
      activebackground=bgSec
)
backBtn = Button(
      btnDiv,
      image=backBtnImg,
      borderwidth=0,
      command=lambda: nextTrack(-1),
      bg=bgMain,
      highlightthickness=0,
      bd=0,
      relief=SUNKEN,
      activebackground=bgSec
)
frontBtn = Button(
      btnDiv,
      image=frontBtnImg,
      borderwidth=0,
      command=lambda: nextTrack(1),
      bg=bgMain,
      highlightthickness=0,
      bd=0,
      relief=SUNKEN,
      activebackground=bgSec
)
shuffleBtn = Button(
      btnDiv,
      image=shuffleBtnImgGray,
      borderwidth=0,
      command=shuffleBtnFunc,
      bg=bgMain,
      highlightthickness=0,
      bd=0,
      relief=SUNKEN,
      activebackground=bgSec
)

# Grid layout for control buttons
backBtn.grid(row=0, column=0, padx=(10,60), pady=(0,10))
mainBtn.grid(row=0, column=1, padx=(0,60), pady=(0,10))
stopBtn.grid(row=0, column=2, padx=(0,60), pady=(0,10))
frontBtn.grid(row=0, column=3, padx=(0,60), pady=(0,10))
shuffleBtn.grid(row=0, column=4, padx=(0,10), pady=(0,10))

# A list containing elements which use saved colors
bgMainBgList = [
      screen,
      downloadMenu,
      themeMenu,
      durLabel,
      curTitle,
      btnDiv,
      mainBtn,
      stopBtn,
      backBtn,
      frontBtn,
      shuffleBtn,
      settings,
      left_frame,
      right_frame,
      down_frame,
      btnDiv,
      volSliderLabel,
      volSliderFrame
]
fgMainFgList = [trackBox, settings, downloadMenu, themeMenu, durLabel]
bgSecBgList = [settings,trackBox]

# Get the default color
defaultColor = open("./config/COLOR.txt", "rt")
changeColor(defaultColor.read())
defaultColor.close()

#initiate
screen.mainloop()
```

5.3 : download.py

```python
# Song search API wrapper
import deezer
# Access to Web browser
import webbrowser
# To conver JSON files
import json
# To open URLs
import requests
# To download form youtube
from pytube import YouTube
# To search youtube
from pytube import Search
# To convert MP4 to MP3
from moviepy.editor import *
# To make folders
import shutil
# To download images
import urllib
# Accessing API password
import os
from dotenv import load_dotenv

# Tkinter
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter.ttk import *

# Library for adding cover art to mp3
import eyed3
from eyed3.id3.frames import ImageFrame


# Get the App id and app secret
try:
      load_dotenv()
      app_id = os.environ["APP_ID"]
      app_secret = os.environ["APP_SECRET"]
except KeyError:
      print("FATAL ERROR :")
      print("Uh Oh! You don't have an API key, so I can't access Deezer!")
      print("Did you get a env file with API keys")
      raise SystemExit(0)

# Log into deezer with our app id (username) and password(app secret)
client = deezer.Client(
      app_id=app_id,
      app_secret=app_secret,
      # This is so results are in English only
      headers={"Accept-Language": "en"},
)
# This is a function to turn MP4 files into just audio MP3 files
def mp4_to_mp3(mp4, mp3):
      mp4_without_frames = AudioFileClip(mp4)
      mp4_without_frames.write_audiofile(mp3)
      mp4_without_frames.close()


# This function modifies the mp3 to add album cover art
def addAlbumCover(audio, image, title, albumName, artist):
      audiofile = eyed3.load(audio)
      if audiofile.tag == None:
      audiofile.initTag()
      audiofile.tag.images.set(3, open(image, "rb").read(), "image/jpeg")
      audiofile.tag.save()
      audiofile.tag.title = title
      audiofile.tag.album = albumName
      audiofile.tag.album_artist = artist
      audiofile.tag.save(version=eyed3.id3.ID3_V2_3)
def download_button_clicked():
      status_bar.config(text=f"Finding Track...")
      ws.update_idletasks()
      global deezerTrack
      # Get currently selected song
      chosenIndex = results_listbox.curselection()[0]
      chosenTrack = deezerTrack[chosenIndex]
      # This is the song name and artist name, replacing spaces with
underscores, and replace the /
      songName = chosenTrack.title_short.replace(" ", "_").replace("/",
"-")
      artistName = chosenTrack.artist.name.replace(" ", "_").replace("/",
"-")
      # This is the song name and artist name, replacing spaces and slashes
with underscores
```

```
      songName = chosenTrack.title_short.replace(" ", "_").replace("/",
"_")
      artistName = chosenTrack.artist.name.replace(" ", "_").replace("/",
"_")

      # Search for the Artist + Song Name + "Audio" on youtube with pytube
      # Example - "Avicii Waiting for love audio" is searched
      searchList = Search(songName + artistName + " audio")
      # Get the first video from search result
      firstResult = str(searchList.results[0])
      status_bar.config(text=f"Getting Song...")
      ws.update_idletasks()
      # These two lines get the video id, and uses that to make the link
      # Basically, get the link
      vidID = firstResult.split("=")[1].replace(">", "")
      finalLink = "https://www.youtube.com/watch?v=" + vidID

      # This is a path to the folder where our song will be saved
      # It names the folder the title of the song, replacing spaces with
underscores
      # Example - "Waiting_for_love"
      path = "./music/"  # + str(chosenTrack["title"]).replace(" ", "_")

      # Make the folder
      # os.mkdir(path)
      status_bar.config(text=f"Downloading song...")
      ws.update_idletasks()

      # Use the link we found before to download the video at lowest
resolution (we only need audio) to the folder
      target = YouTube(finalLink)

      # This downloads the MP4 file inside the folder we made
      target.streams.filter(file_extension="mp4").first().download(
      path, filename=songName + ".mp4"
      )

      status_bar.config(text=f"Changing files to MP3...")
      ws.update_idletasks()
      # This will convert our MP4 to MP3 using that function
      mp4_to_mp3(f"{path}/{songName}.mp4",
f"{path}/{artistName}-{songName}.mp3")
```

```
      # Delete the original file to save on space
      os.remove(f"{path}/{songName}.mp4")

      # These two lines get the cover art of the album from the API, and
download it to our folder
      coverImg = chosenTrack.album.cover_big

      status_bar.config(text=f"Getting cover image...")
      ws.update_idletasks()
      # This will save the cover image as
"./Downloads/ARTISTNAME-SONGNAME-COVER.jpg"
      # Example - ./Downloads/Avicii-Waiting_for_love-cover.jpg
      urllib.request.urlretrieve(
      coverImg, f"{path}/albumCover/{artistName}-{songName}-cover.jpg"
      )

      # Change the cover art of the mp3
      audio = f"{path}/{artistName}-{songName}.mp3"
      image = f"{path}/albumCover/{artistName}-{songName}-cover.jpg"
      addAlbumCover(
      audio,
      image,
      chosenTrack.title_short,
      chosenTrack.album.title,
      chosenTrack.artist.name,
      )

      status_bar.config(text=f'Downloading Complete! Click "Reload Tracks"
on main menu')


def searchButton():
      status_bar.config(text=f"Searching...")
      ws.update_idletasks()
      # Ask for artist
      query = modify.get()

      # Error handling for empty
      if len(query) == 0:
      return

      # Use deezer to search for this artist
      global deezerTrack
      deezerTrack = client.search(query)

      # Clear listbox
      results_listbox.delete(0, "end")

      cunt = 0
      for result in deezerTrack:
      # Limiting searches to top 50 results
      if cunt >= 50:
            break
      results_listbox.insert(
            "end", str(cunt + 1) + ". " + result.title_short + "-" +
result.artist.name
      )
      cunt += 1

      status_bar.config(text=f'Showing results for "{query}"')
      ws.update_idletasks()


# This is the code for our new window
def downloadSong(main):
      # Generate scheme
      defaultColor = open("./config/COLOR.txt", "rt")
      scheme = defaultColor.read()
      defaultColor.close()

      if scheme == "BLUE":
      bgMain = "#1A1C26"
      bgSec = "#2D2E39"
      fgMain = "#F4F4F2"
      accent = "#3498DB"
      col = "Blue"

      elif scheme == "GREEN":
      bgMain = "#19231A"
      bgSec = "#2E392A"
      fgMain = "#F2F8F2"
      accent = "#4CAF50"
      col = "Green"

      elif scheme == "RED":
      bgMain = "#1C161A"
```

```
       bgSec = "#2E2629"
       fgMain = "#EDF3FA"
       accent = "#F53C36"
       col = "Red"

       elif scheme == "PURPLE":
       bgMain = "#16181C"
       bgSec = "#27262E"
       fgMain = "#EEFAED"
       accent = "#8F36F5"
       col = "Purple"

       # Create main window
       global ws
       ws = tk.Toplevel(main)
       ws.title("Download Song")
       # Set the geometry of the main window (width x height)
       ws.geometry("510x325")
       ws.resizable(0, 0)
       ws.configure(bg=bgMain)

       # Create a StringVar to hold the text input value
       text = tk.StringVar()

       # Create a frame to hold other widgets
       Frm = tk.Frame(ws)

       # Configure the column weights of the main window to manage space
distribution
       ws.columnconfigure(0, weight=1)
       ws.columnconfigure(1, weight=1)
       ws.columnconfigure(2, weight=1)
       # Create an entry widget for text input, linked to the text StringVar
       global modify
       modify = tk.Entry(
       ws,
       textvariable=text,
       bg=bgSec,
       fg=fgMain,
       bd=0,
       highlightcolor=fgMain,
       highlightthickness=1,
       )
```

```
# Text entry box and Submit button in the same row
modify_label = tk.Label(
ws,
text="Search : ",
borderwidth=0,
bg=bgMain,
fg=fgMain,
highlightthickness=0,
bd=0,
)
modify_label.grid(row=0, column=0, pady=10, padx=10, sticky=tk.E)
modify.grid(row=0, column=1, pady=10, padx=10, sticky=tk.E)
modify.focus()

# Create a button that will call the find function when pressed
buttn = tk.Button(
ws,
text="Search",
borderwidth=0,
bg=accent,
fg=bgMain,
highlightthickness=0,
bd=0,
command=searchButton,
)

# Place the button in the grid layout
buttn.grid(column=2, row=0, padx=5, pady=5)

global results_listbox
results_listbox = tk.Listbox(
ws,
selectmode=tk.SINGLE,
bg=bgSec,
fg=fgMain,
borderwidth=0,
highlightthickness=0,
selectbackground=accent,
selectborderwidth=0,
)

results_listbox.grid(
row=1, column=0, columnspan=3, pady=10, padx=10, sticky=tk.W + tk.E
```

```
      )

      scrollbar = tk.Scrollbar(ws, orient=tk.VERTICAL,
command=results_listbox.yview)
      scrollbar.grid(row=1, column=4, pady=10, padx=10, sticky=tk.W + tk.N
+ tk.S)
      results_listbox.config(yscrollcommand=scrollbar.set)
      contacts = []

      # Download button
      download_button = tk.Button(
      ws,
      text="Download",
      command=download_button_clicked,
      borderwidth=0,
      bg=accent,
      fg=bgMain,
      highlightthickness=0,
      bd=0,
      )
      download_button.grid(row=2, pady=10, columnspan=4)

      # Status bar at the bottom
      global status_bar
      status_bar = tk.Label(
      ws,
      text="Download Something!",
      bd=1,
      relief=tk.SUNKEN,
      anchor=tk.W,
      borderwidth=0,
      fg=fgMain,
      bg=bgSec,
      )
      status_bar.grid(row=3, columnspan=5, sticky="ew", padx=5, pady=5)

      ws.mainloop()
```

6 : Conclusion

6.1 : Achievements

Through this project we learned how to use a tkinter based GUI with a comprehensive backend suite. We were able to learn about various python libraries to manage our backend, such as pygame Mixer, Mutagen, Shutil, movie py etc. We also learned how to contact an API (Deezer) which enabled us to get the data.

6.2 : Improvements

Various features can be added to enhance the user experience. These include:

- Change to a more reliable music search API key
- Create database and store more information about the Tracks
- Implement Playlists
- Add option to search both tracks or album
- Connect music GUI to Download feature
- Add secret features
- Autoplay, Themes, Loops

We plan to add these in the upcoming release.

6.3 : Contributors

- Adeeb S Rahman
- Nazia Niha Kaesh
- Subham Sharma

7 : References

https://www.youtube.com/watch?v=djDcVWbEYoE&list=PLXLYwvNGGPoUzjKiGvuXm8qeiOYMnFria
-Used for GUI of the project and as a reference for how to play music using pygame.

https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html
- used for General tutorials on tkinter and resolving many queries in the project.

https://developers.deezer.com/
- API used in a project.

https://deezer-python.readthedocs.io/en/stable/
-Deezer API Wrapper

https://coolors.co/
Color palette

https://www.flaticon.com/free-icons/mp3-player
Used for icons in GUI

https://icons8.com/
Also used for icons in GUI.

The full Github repository can be found in the following link:

https://github.com/Arctican4Real/Melodia

Github Repository