MUSICAL TIME MACHINE

A Mini Project Report

submitted by

ROSHNA SHIRIN M(MES24MCA-2046)

to the APJ Abdul Kalam Technological University
in partial fulfilment of the requirements for the award of the Degree

of

Master of Computer Applications

Department of Computer Applications

MES College of Engineering

Kuttippuram, Malappuram – 679582

October, 2025

Declaration

I undersigned hereby declare that the project report MUSICAL TIME MACHINE submitted for partial fulfilment of the requirements for the award of degree of Master of Computer Applications of the APJ Abdul Kalam Technological University, Kerala, is a bonafide work done by me under supervision of Mrs. Reshmi K, Assistant professor, Department of Computer Applications. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

ROSHNA SHIRIN M(MES24MCA-2046)

06-10-25

DEPARTMENT OF COMPUTER APPLICATIONS

MES COLLEGE OF ENGINEERING, KUTTIPPURAM

CERTIFICATE

This is to certify that the report entitled **MUSICAL TIME MACHINE** is a bonafide record of the Mini Project work during the year 2025-26 carried out by **ROSHNA SHIRIN M (MES24MCA-2046)** submitted to the APJ Abdul Kalam Technological University, in partial fulfilment of the requirements for the award of the Master of Computer Applications, under my guidance and supervision. This report in any form has not been submitted to any other University or Institution for any purpose.

Internal Supervisor   Head of The Department

Acknowledgment

I endeavor stands incomplete without dedicating my gratitude to a few people who have contributed towards the successful completion of my project. I pay my gratitude to the Almighty for His invisible help and blessing for the fulfillment of this work. At the outset I express my heart full thanks to my Head of the Department, Prof. Hyderali K for permitting me to do this project. I take this opportunity to express my profound gratitude to Mrs. Reshmi k, my project guide for her valuable support. I am also grateful to all my teaching and non- teaching staff for their encouragement, guidance and wholehearted support. Last but not least, I gratefully indebted to my family and friends, who gave us a precious help in doing my project.

ROSHNA SHIRIN M (MES24MCA-2046)

Abstract

This project, Musical Time Machine, allows users to create personalized Spotify playlists based on any date they choose. By entering a specific date (for example, August 22, 2000), the application fetches the top-charting songs from that period and generates a nostalgic playlist through the Spotify API. This unique experience combines music, memory, and personalization to deepen users' connection to their past. Music has a powerful ability to evoke memories and emotions, yet most streaming platforms lack tools that enable users to revisit specific moments in time through music. Motivated by this gap, this project leverages technical APIs and historical music chart data to help users relive memories by listening to songs that were popular on any chosen date. The core goal is to retrieve the top hits from a given historical date using publicly available music chart information, such as Billboard rankings, and then create a playlist on Spotify using the Spotify Web API. Users authenticate their Spotify account securely via OAuth, allowing the app to automatically save the generated playlist to their library. Users simply input a desired date, and the app collects the top songs from that day, compiles a playlist, and saves it to their Spotify account. The project features a user-friendly interface for easy date entry and quick playback of the resulting playlist. Built primarily with Python, the application supports both Windows and Linux environments. The frontend can be developed with Flask for a web interface. Key libraries include spotipy for Spotify integration, requests for API calls, and datetime for date management.OAuth 2.0 ensures secure user authentication. Development can be done using popular IDEs like Visual Studio Code or PyCharm. Musical Time Machine offers a creative, engaging way for users to reconnect with their past through music. Looking ahead, the project could be expanded with features like genre filtering, mood-based playlist creation, or social sharing options, further enriching the user experience.

Table of Contents

List of Figures .... **viii**

List of Tables .... **ix**

Chapter 1. Introduction **10**

1.1 Motivation 11

1.2 Objectives 11

1.3 Contributions 12

1.4 Report Organization 13

Chapter 2. System Study **14**

2.1 Existing System 14

2.2 Proposed System 14

2.3 Functionalities of Proposed System 15

Chapter 3. Methodology **16**

3.1 Introduction 16

3.2 Software Tools 17

3.2.1 Python 17

3.2.2 Flask 17

3.2.3 Spotify Web API & Billboard Hot 100 18

3.2.4 Chart.js 18

3.2.5 Visual Studio Code 18

3.2.6 GitHub 18

3.2.7 Spotipy 18

3.2.8 Requests 19

3.2.9 Beautifulsoup(bs4) 19

3.3 Module Description 19

3.3.1 Authentication Module 20

3.3.2 Time Travel Module 21
3.3.3 Mood Playlist Module 23
3.3.4 Genre Module 24
3.3.5 Language Module 25
3.3.6 Lyrics Finder Module 25
3.3.7 Search Module 26
3.3.8 Dashboard Module 27
3.4 User Story 29
3.5 Product Backlog 30
3.6 Project Plan 31
3.7 Sprint Backlog 32
Chapter 4. Results and Discussions 35
4.1 Results 35
Chapter 5. Conclusion 40
References 41
Appendix 43
Appendix A Workflow Diagram 43
Appendix B Source code 44
Appendix C Screenshots 51

List of Figures

Figure 4.1:time travel interface 35

Figure 4.2:mood based playlist 36

Figure 4.3:genre playlist 37

Figure 4.4:language playlist 37

Figure 4.5:dashboard statistics 37

Figure 4.6:lyrics page 38

Figure 4.7:lyrics result 38

Figure 4.8:playlist created 39

Figure 4.9:playlist created in spotify 39

Figure 0.1:Workflow diagram 43

List of Tables

Table 3.1: List the software tools or languages used for the project development 17

Table 3.2: User Story 29

Table 3.3:product backlog 30

Table 3.4: Project plan 31

Table 3.5: Sprint backlog.....................32

Chapter 1. Introduction

Music streaming platforms have revolutionized how people discover and enjoy music. With millions of tracks available across genres, languages, and moods, users often struggle to find the right music that matches their preferences or mood. The increasing dependence on digital streaming services like Spotify has created the need for intelligent systems that personalize user experiences and automate playlist generation based on listening behaviour, genre, or emotional state.

The Musical Time Machine project aims to develop a dynamic web-based application that integrates with the Spotify Web API to create personalized playlists automatically. The system allows users to explore music by time period (year-based "Time Travel"), mood, genre, and language, along with advanced features such as Billboard data integration for retro English hits, custom search, and lyrics finder. A dashboard with interactive graphs provides insights into user listening patterns, such as top artists, song count, and favourite genres. The application's interface is designed with usability in mind, and an intuitive layout built using Flask and Chart.js.

This project combines concepts from Web Programming, API integration, and Data Visualization, applying real-time data fetched from Spotify's API. It bridges user interaction with intelligent automation converting user inputs like mood, year, or language into curated playlists. The inclusion of audio preview clips and album art enhances the listening experience directly within the application.

By developing this system, the project demonstrates how web applications can leverage modern APIs and data analytics to deliver personalized multimedia recommendations efficiently. It showcases an integration of backend logic (Python, Flask), data-driven UI visualization

(JavaScript, Chart.js), and API-based content management (Spotify Web API) offering an engaging and intelligent music exploration experience.

1.1 Motivation

In today’s fast-paced digital world, music plays an essential role in relaxation, productivity, and emotional balance. However, with the enormous amount of content available across streaming platforms, users often find it difficult to choose what to listen to next. Traditional playlist creation requires manual searching and filtering, which can be time-consuming and repetitive.

The motivation behind developing Musical Time Machine is to provide an intelligent, automated, and interactive music discovery experience. The project seeks to combine the power of the Spotify Web API with user-friendly visualization and data analysis to create meaningful, personalized playlists. By integrating features such as time-based playlist generation (Time Travel), mood and genre-based selection, and Billboard historical data, the system helps users explore music across eras, emotions, and languages effortlessly.

The increasing trend of using APIs for smart applications and the availability of real-time data from Spotify provided a strong motivation to implement this project. Additionally, the idea of using data visualization and analytics to show personal music listening habits like top artists or listening frequency adds a creative, modern dimension to the project.

1.2 Objectives

The main objectives of the Musical Time Machine project are as follows:

- To design and develop an intelligent web-based system for automated playlist creation using Spotify API integration.

- To provide multiple playlist creation options — based on year, mood, genre, language, and recently played songs.
- To integrate Billboard Hot 100 data for generating retro English playlists.
- To include a song preview player and display album covers for each track.
- To visualize user listening data using Chart.js graphs and charts.
- To ensure modular and maintainable project architecture using Flask Blueprints.

1.3 Contributions

The Musical Time Machine project contributes both technically and educationally in the following ways:

- Practical Implementation of API Integration: Demonstrates real-world use of the Spotify Web API for data fetching, playlist creation, and user authentication using OAuth 2.0.

- Innovative Time Travel Feature: Allows users to travel back in time musically by generating playlists from a specific year using Billboard data.

- Enhanced User Experience: Includes preview audio clips, and a sidebar navigation system for seamless usability.

- Data Analytics Integration: Uses Chart.js for visual dashboards that present user insights, engagement patterns, and top music trends.

- Modular Web Architecture: Implements a Flask Blueprint structure to keep the project modular, scalable, and easy to maintain.

- Cross-Disciplinary Learning: Combines knowledge from Web Programming, Data Visualization, Cloud APIs, and UI/UX Design.

1.4 Report Organization

The rest of this report is organized as follows:

Chapter 1 –Introduction: Provides the background, motivation, objectives, contributions and organization of the report.

Chapter 2 – System Study: Describes the existing system and its limitations, followed by the details of the proposed system and its functionalities.

Chapter 3 – Methodology: Explains the methodology used for the implementation, including design approach, software tools, and detailed description of different modules. Sprint details are also provided in this chapter.

Chapter 4 – Results and Discussions: Presents the implementation results, screenshots of the developed system, and discussion of outcomes with respect to project objectives.

Chapter 5 – Conclusion: Summarizes the overall contributions, highlights the benefits of the developed system, and suggests possible directions for future enhancements.

Chapter 2. System Study

The project “Musical Time Machine” is a web-based application that connects with the Spotify API to automatically generate personalized playlists based on the user’s preferences such as mood, genre, language, or a specific year.

It aims to simplify music discovery by allowing users to “travel through time” to explore songs from different eras and visualize their listening trends through interactive dashboard analytics.

2.1 Existing System

In the existing music streaming environment, users depend on platforms like Spotify, YouTube, or Apple Music to manually search and create playlists. These platforms do not offer features that combine historical data, mood-based recommendations, or multi-language discovery within a single interface. Users have to:

- Manually browse songs by year or genre.
- Depend on platform-curated playlists, which are not personalized.
- Lack insight into their listening behaviour or statistics.
- Have no option to revisit songs from specific time periods like “Top Hits from 1990s”.
- Hence, the existing system is time-consuming, non-personalized, and lacks integrated analytics or historical playlist generation features.

2.2 Proposed System

The proposed system, Musical Time Machine, is a Flask-based web application designed to enrich the music listening experience. It allows users to log in using their Spotify account and automatically generate playlists through various modes such as Time Travel, Mood, and Genre. The system fetches songs using the Spotify API and incorporates Billboard Hot 100 data for English tracks, offering a diverse and dynamic selection of music tailored to user preferences.

In addition to playlist creation, the application provides insightful listening statistics through a graphical dashboard. Users can view their weekly activity, top artists, and other trends, making it easier to understand and reflect on their music habits. By combining automation, data visualization, and an intuitive user interface, Musical Time Machine streamlines playlist management and transforms how users engage with their favorite tunes.

2.3 Functionalities of Proposed System

1. Spotify Authentication and Integration

The system enables users to securely log in using their Spotify account. This integration allows access to user-specific data such as listening history, favorite tracks, and playlists. It serves as the foundation for personalized playlist generation and statistical analysis.

2. Automated Playlist Creation

Musical Time Machine offers multiple modes for generating playlists automatically, including Time Travel (songs from a specific era), Mood-based selections, and Genre-focused compilations. By leveraging the Spotify API and Billboard Hot 100 data for English songs, the system curates diverse and engaging playlists tailored to individual preferences.

3. Listening Statistics Dashboard

Users can explore their music habits through a visually rich dashboard. This includes insights such as weekly listening activity, top artists, and track trends. The graphical representation helps users reflect on their musical journey and discover patterns in their listening behavior.

4. Enhanced User Experience

Combining automation, data visualization, and an intuitive user interface, the system simplifies playlist management and enhances user engagement. It transforms the way users interact with music by making discovery and organization effortless and enjoyable.

Chapter 3. Methodology

Software methodology is essential for building robust, scalable, and user-friendly applications. It provides a structured approach to development, ensuring that each phase—from planning to deployment is well-organized and efficient. For this project, a methodology was needed to manage multiple modules, integrate external APIs, and deliver a seamless user experience

3.1 Introduction

In any software development project, adopting a structured methodology is essential to ensure clarity, efficiency, and scalability throughout the development lifecycle. A well-defined software methodology provides a roadmap for planning, designing, implementing, testing, and maintaining the system. It helps teams manage complexity, allocate resources effectively, and adapt to changing requirements with minimal disruption.

For the Musical Time Machine project, which integrates multiple modules, external APIs, and user-centric features, a software methodology was crucial to maintain modularity, streamline development, and deliver a seamless user experience. By following a systematic approach, the project could evolve iteratively, incorporate feedback, and ensure that each component from playlist generation to dashboard analytics was built with precision and purpose.

3.2 Software Tools

The development of Musical Time Machine involved several software tools and languages. These were selected based on their compatibility, ease of use, and ability to support rapid development. The list of tools and languages used in this project is given in **Table 3.1**.

Table 3.1: List the software tools or languages used for the project development

Operating System	Windows 11
Front End	JavaScript, HTML, CSS,
Back End	Python
Framework	Flask (Python Web Framework)
Visualization	Chart.js
Database	No internal DB (uses Spotify data)
IDE	Visual Studio Code
Version Control	GitHub
Library	Spotipy ,Requests,BeautifulSoup (bs4)

3.2.1 Python

Python was chosen as the backend language due to its simplicity, readability, and extensive support for web development and API integration. It allowed rapid prototyping and seamless communication with external services like Spotify. Python’s rich ecosystem of libraries made it ideal for handling user sessions, playlist creation, and data processing.

3.2.2 Flask

Flask was selected as the web framework for its lightweight and modular architecture. It supports blueprint-based development, which helped organize features like Time Travel, Mood-based playlists, and Dashboard analytics. Flask also simplifies routing, template rendering, and integration with third-party APIs, making it perfect for building scalable web applications.

3.2.3 Spotify Web API & Billboard Hot 100

These external sources were used to fetch music data. The Spotify Web API provided access to user playlists, track metadata, and audio features, while Billboard Hot 100 was scraped to retrieve trending English songs by year. Together, they enabled rich playlist generation and historical music exploration.

3.2.4 Chart.js

Chart.js was used to visualize user statistics such as top artists and weekly listening activity. Its interactive and responsive charts enhanced the dashboard experience, making data insights more engaging and accessible.

3.2.5 Visual Studio Code

VS Code was the primary IDE used for development. Its support for Python, HTML, and JavaScript, along with integrated Git features and extensions, made it a powerful tool for writing, debugging, and managing the project codebase.

3.2.6 GitHub

GitHub was used for version control and collaboration. It helped track changes, manage branches, and maintain a clean development workflow throughout the project lifecycle.

3.2.7 Spotipy

Spotipy is a lightweight Python library used to interact with the Spotify Web API.

It allows developers to access user playlists, tracks, artists, and music recommendations programmatically.

In this project, Spotipy was used to authenticate users through OAuth, fetch top artists, generate playlists automatically, and retrieve song data such as preview URLs and release years.

It simplified Spotify API integration without manually handling complex HTTP requests.

3.2.8 Requests

The Requests library is a simple and powerful Python package used for making HTTP requests.

It enables communication with external APIs and web pages in an easy-to-read syntax.

In this project, it was primarily used to retrieve Billboard Hot 100 chart data for the "Time Travel" feature, allowing the application to fetch top songs from specific years

3.2.9 Beautifulsoup(bs4)

BeautifulSoup is a Python library used for web scraping and HTML/XML parsing.

It helps extract structured data from web pages by navigating and searching through HTML elements easily.

In this project, BeautifulSoup was used to scrape song titles from Billboard's website for the selected year, which were then used to find corresponding tracks on Spotify.

3.3 Module Description

System modules are individual components that perform specific tasks within the application. Each module in Musical Time Machine is designed to be independent yet interconnected through Flask routing and shared functions.

- Authentication Module
- Time Travel Module
- Mood playlist Module
- Genre Module
- Language Module
- Lyrics Finder Module

- Search Module
- Dashboard Module

3.3.1 Authentication Module

- This module handles Spotify OAuth login. It ensures that users are authenticated before accessing any features. It stores session tokens securely and manages user identity across modules.

code for authentication

```
@auth_bp.route("/login")

def login():

    auth_url = sp_oauth.get_authorize_url()

    print("Redirecting to Spotify:", auth_url)

    return redirect(auth_url)

@auth_bp.route("/callback")

def callback():

    code = request.args.get("code")

    token_info = sp_oauth.get_access_token(code, as_dict=True)

    session["token_info"] = token_info

    sp = spotipy.Spotify(auth=token_info["access_token"])

    user = sp.current_user()

    session["user_id"] = user["id"]

    session["user_name"] = user["display_name"]

    return redirect(url_for("dashboard.dashboard"))
```

- This function initializes the Spotify client using the user's token, enabling secure API calls.

3.3.2 Time Travel Module

Generates playlists based on songs from a specific year. It uses Billboard Hot 100 data and Spotify search to compile tracks.

Explanation:

- This function scrapes Billboard and returns a list of song titles and artists for the selected year.

code for Time travel

```
@time_travel_bp.route("/time_travel", methods=["POST"])

def time_travel():

    year = request.form["year"]

    language = request.form["language"].lower()

    song_count = int(request.form.get("song_count", 30))  # slider value

    source = request.form.get("source", "spotify")        # only relevant if English

    sp = get_spotify_client()

    if not sp:

        return redirect(url_for("auth.index"))

    uris = []

    # Language → Market mapping

    language_market = {

        "hindi": "IN",
        "tamil": "IN",
        "malayalam": "IN",
        "telugu": "IN",
        "kannada": "IN",
        "spanish": "ES",
        "korean": "KR",
        "japanese": "JP",
        "english": "US"
    }
    market = language_market.get(language, "US")
    # For English → let user pick Billboard or Spotify
    if language == "english":
        if source == "billboard":
            songs = get_billboard_hot_100(year, max_songs=song_count)
            for song in songs:
                try:
                    result = sp.search(q=f"track:{song} year:{year}", type="track", limit=1,
market="US")
                    if result and result["tracks"]["items"]:
                        uris.append(result["tracks"]["items"][0]["uri"])
                except Exception:
```

```
            continue
    else:  # Spotify
        query = f"english hits {year}"
        uris = fetch_tracks(sp, query, year, "US", max_songs=song_count)
```

3.3.3 Mood Playlist Module

Creates playlists based on user-selected moods like Happy, Chill, Sad, or Party. It uses Spotify's audio features and mood tags to filter songs.

code for mood playlist

```
def mood():
    mood = request.form["mood"]
    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))
    mood_map = {
        "happy": "happy upbeat",
        "sad": "sad emotional",
        "chill": "relax calm",
        "party": "party dance"
    }
    query = mood_map.get(mood, "music")
    result = sp.search(q=query, type="track", limit=20)
    uris = [item["uri"] for item in result["tracks"]["items"]]
    playlist = create_playlist(sp, f"{mood.capitalize()} Vibes", f"A {mood} mood playlist")
    if uris:
        for i in range(0, len(uris), 100):
            sp.playlist_add_items(playlist["id"], uris[i:i+100])
        tracks = [sp.track(uri) for uri in uris]
```

3.3.4 Genre Module

Allows users to select genres such as Pop, Rock, Jazz, or Hip-hop. The module queries Spotify's genre endpoints to fetch relevant tracks.

code for genre module

```
def genre():
    genre = request.form["genre"]
    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))

    query = f"{genre} music"
    result = sp.search(q=query, type="track", limit=20)
    uris = [item["uri"] for item in result["tracks"]["items"]]
    playlist = create_playlist(sp, f"{genre.capitalize()} Collection", f"{genre} based playlist")
    if uris:
        sp.playlist_add_items(playlist["id"], uris)
        tracks = [sp.track(uri) for uri in uris]
```

3.3.5 Language Module

Generates playlists based on regional or international languages. It filters Spotify tracks using language-based metadata.

code for language module

```
def language():

    language = request.form["language"]

    sp = get_spotify_client()

    if not sp:

        return redirect(url_for("auth.index"))

    query = f"{language} music"

    result = sp.search(q=query, type="track", limit=20)

    uris = [item["uri"] for item in result["tracks"]["items"]]

    playlist = create_playlist(sp, f"{language.capitalize()} Hits", f"Best of {language} songs")

    if uris:

        sp.playlist_add_items(playlist["id"], uris)

        tracks = [sp.track(uri) for uri in uris]
```

3.3.6 Lyrics Finder Module

Fetches lyrics using a third-party API. It takes song title and artist as input and returns the lyrics.

code for lyrics finder module

```
def lyrics():

    artist = request.form.get("artist")

    song = request.form.get("song")
    try:
        url = f"https://api.lyrics.ovh/v1/{artist}/{song}"
        response = requests.get(url)
        data = response.json()
        lyrics_text = data.get("lyrics", "Lyrics not found 😢")
    except Exception:
        lyrics_text = "Error fetching lyrics."
```

3.3.7 Search Module

Enables manual search and custom playlist creation. Users can input song names or artists to build personalized playlists.

code for search module

```
def search():
    q = request.form.get("query", "").strip()
    if not q:
        flash("Enter something to search")
        return redirect(url_for("search.search_page"))
    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))
    res = sp.search(q=q, type="track", limit=50)
    tracks = res.get("tracks", {}).get("items", [])
    items = [{
        "id": t["id"],
        "name": t["name"],
        "artists": ", ".join([a["name"] for a in t["artists"]]),
        "uri": t["uri"],
        "preview_url": t.get("preview_url")
    } for t in tracks]
    return render_template("search_results.html", query=q, items=items)
```

3.3.8 Dashboard Module

Displays user statistics using Chart.js. It shows top artists, songs listened, and weekly activity in graphical format.

```
def dashboard():
    if "token_info" not in session:
        return redirect(url_for("auth.index"))
    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))
    try:
        # Recently played songs (for weekly counts)
        recently_played = sp.current_user_recently_played(limit=50)
        # Extract daily counts
```

```
from collections import Counter
import datetime
dates = [
    item["played_at"][:10] for item in recently_played.get("items", [])
]
date_counts = Counter(dates)
# Ensure we cover last 7 days
today = datetime.date.today()
labels = [(today - datetime.timedelta(days=i)).isoformat() for i in range(6, -1, -1)]
weekly_counts = [date_counts.get(day, 0) for day in labels]
# Top artists distribution
top_artists = sp.current_user_top_artists(limit=5, time_range="short_term")
artist_labels = [a["name"] for a in top_artists["items"]]
artist_counts = [a["popularity"] for a in top_artists["items"]]  # use popularity score
stats = {
    "songs_this_week": len(recently_played["items"]) if recently_played else 0,
    "top_artist": artist_labels[0] if artist_labels else "Unknown",
    "weekly_labels": labels,
    "weekly_counts": weekly_counts,
    "artist_labels": artist_labels,
    "artist_counts": artist_counts,
```

}

3.4 User Story

User stories help capture the requirements from the perspective of end users. They describe what different users expect from the system in a concise manner. Table **3.2** lists the user stories for this Musical Time Machine.

Table 3.2: User Story

User story id	As a type of user	I want to	So that I can
1	USER	Log in with Spotify	Access my playlists securely
2	USER	Log out	Protect my account
3	USER	See a dashboard	choose how to create a playlist
4	USER	Select a year	Get songs from that time period
5	USER	Pick a mood	Listen to music that matches my feeling
6	USER	Choose a language	Hear songs in my preferred language
7	USER	Select a genre	Explore music I like
8	USER	User recently played	Save recent songs as a playlist
9	USER	Save playlists automatically	Access them later on spotify
10	USER	Get alerts when no songs are found	Try another option quickly

11	USER	Open playlist link	Play it directly on spotify

3.5 Product Backlog

Table 3.3:product backlog

ID	NAME	PRIORITY	ESTIMATE	STATUS
1	User Authentication	High	5	completed
2	Dashboard	High	5	completed
3	Mood-based Playlist	High	4	completed
4	Language-based Playlist	High	6	completed
5	Genre-based Playlist	High	3	completed
6	Recently Played Playlist	High	2	completed
7	Playlist Management	High	3	completed
8	Error Handling	High	7	completed
9	UI/UX Enhancements	Medium	4	completed
10	Documentation & Reports	High	6	completed
11	Future Enhancements (Optional)	Low	4	completed

3.6 Project Plan

Table 3.4: Project plan

User StoryID	Task Name	Start Date	End Date	Days	Status
1	Sprint 1	7/08/2025	8/08/2025	6	Completed
2		15/08/2025	18/08/2025		Completed
3	Sprint 2	22/08/2025	24/08/2025	14	Completed
11		25/08/2025	26/08/2025		Completed
8		27/08/2025	28/08/2025		Completed
10		29/08/2025	30/08/2025		Completed
7		1/09/2025	5/09/2025		Completed
6	Sprint 3	14/09/2025	15/09/2025	12	Completed
9		16/09/2025	17/09/2025		Completed

4		19/09/2025	22/09/2025		Completed
5		25/09/2025	28/09/2025		Completed

3.7 Sprint Backlog

Table 3.5: Sprint backlog

Backlog term	date	Original estimation in hr.	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
SPRINT1												
User authentication	7/8/25	2	1	1	0	0	0	0	0	0	0	0
dashboard	15/8/25	4	1	1	1	1	0	0	0	0	0	0

SPRINT2												
Mood based playlist	24/8/25	3	1	1	1	0	0	0	0	0	0	0
Playlist management	26/8/25	4	1	1	1	1	0	0	0	0	0	0
Documents and reports	28/8/25	6	1	1	1	1	1	1	0	0	0	0
Error handling	30/8/25	4	1	1	1	1	0	0	0	0	0	0
More features	5/9/25	5	1	1	1	1	1	0	0	0	0	0

SPRINT3												
Recently Played Playlist	15/9/25	3	1	1	1	0	0	0	0	0	0	0
UI/UX Enhancements	17/9/25	4	1	1	1	1	0	0	0	0	0	0

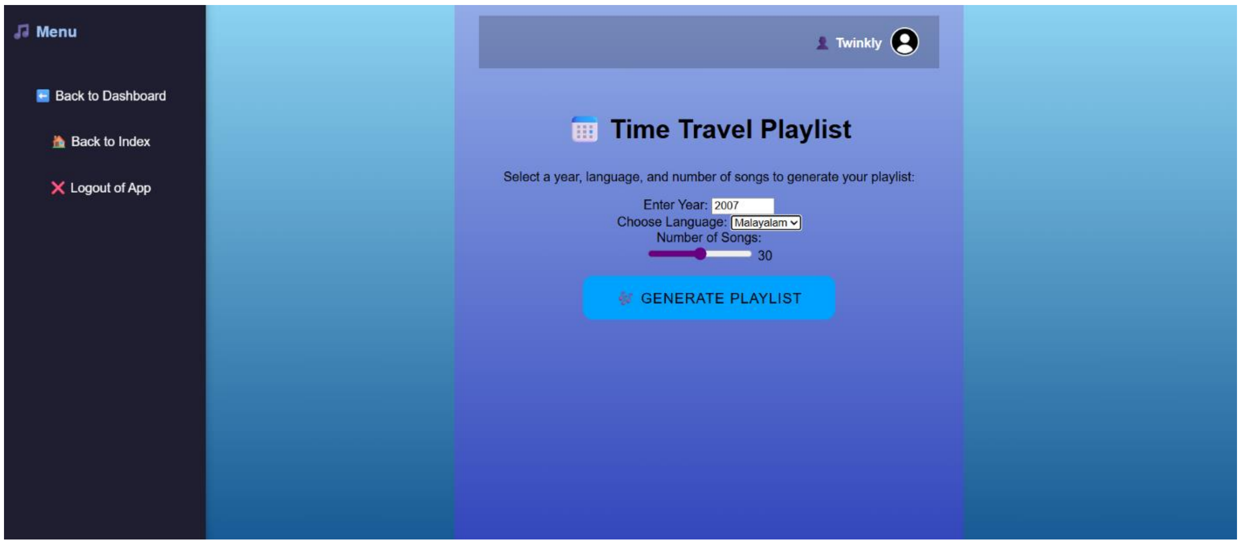
Language-based Playlist	22/9/25	5	1	1	1	1	1	0	0	0	0	0
Genre-based Playlist	28/9/25	3	1	1	1	0	0	0	0	0	0	0
Total		43	11	11	10	7	3	1	0	0	0	0

Chapter 4. Results and Discussions

This chapter presents the outcomes of the Musical Time Machine project and discusses how each core feature contributes to the overall functionality and user experience. The results are demonstrated through selected screenshots of key modules and interfaces, showcasing how the system dynamically interacts with Spotify APIs, processes user inputs, and visualizes data. Each figure is accompanied by a brief explanation to highlight its role within the application.

Below are screenshots of the most significant forms and interfaces used in the project. Each figure includes valid data and demonstrates how users interact with the system to generate playlists, view statistics, and explore music content.

4.1 Results



Time Travel Playlist

Select a year, language, and number of songs to generate your playlist:

Enter Year: 2007
Choose Language: Malayalam
Number of Songs: 30

Figure 4.1:time travel interface

Figure 4.1: Time Travel Playlist Generator Form This form allows users to select a specific year to generate a playlist based on Billboard Hot 100 songs from that era. Once the year is submitted, the system fetches relevant tracks and creates a playlist on the user's Spotify account.

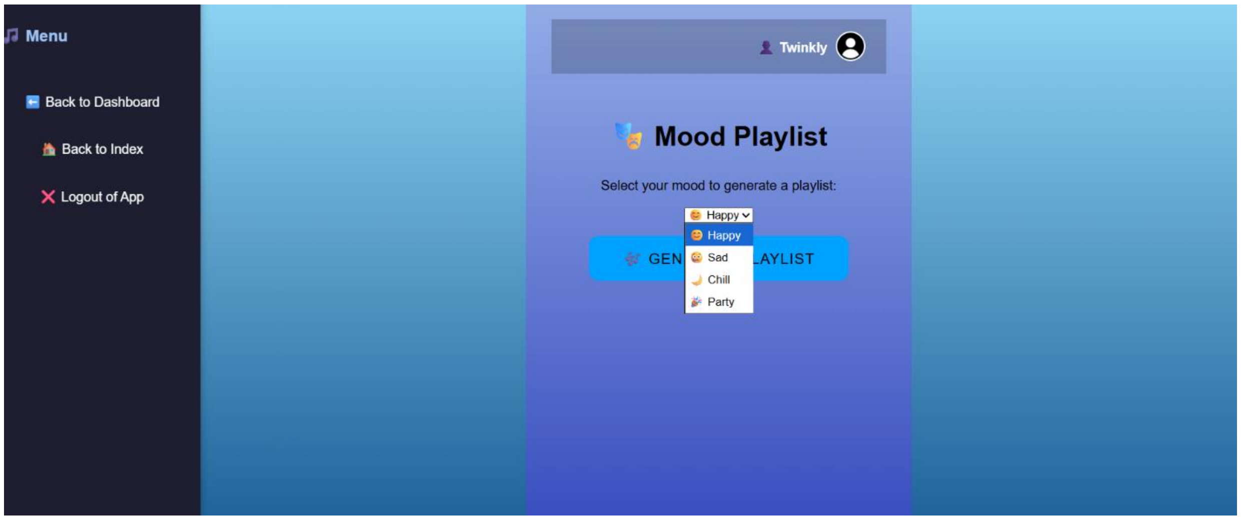


Figure 4.2:mood based playlist

Figure 4.2: Mood-Based Playlist Selection Interface Users can choose from predefined moods such as Happy, Chill, Sad, or Party. The system uses Spotify’s audio features to filter and compile tracks that match the selected mood.

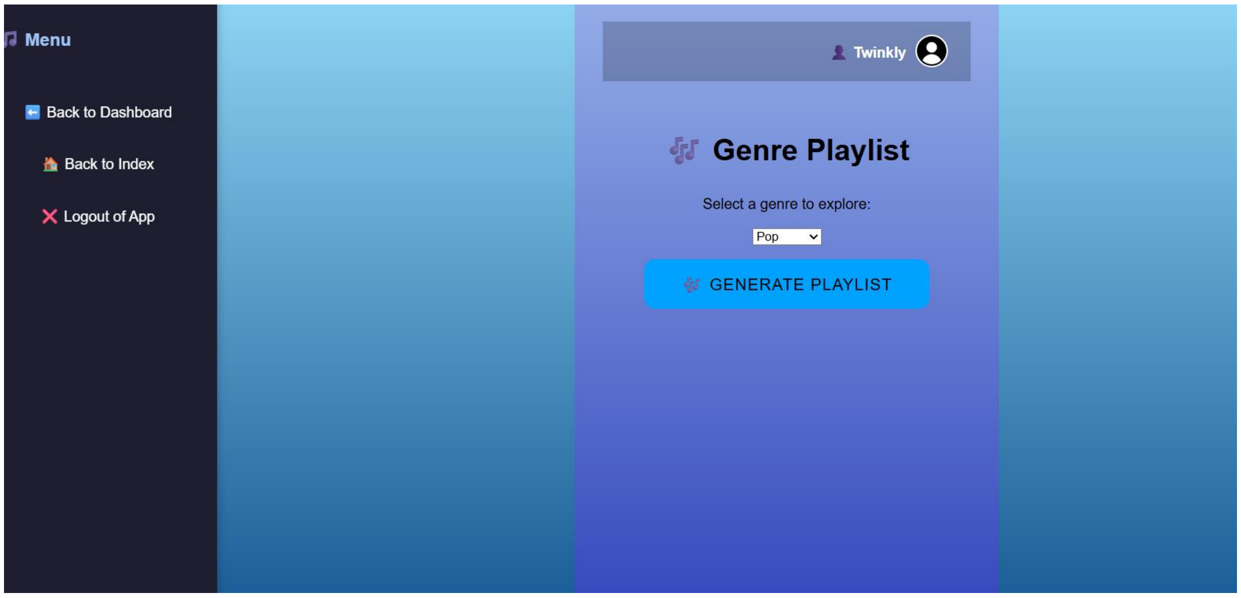
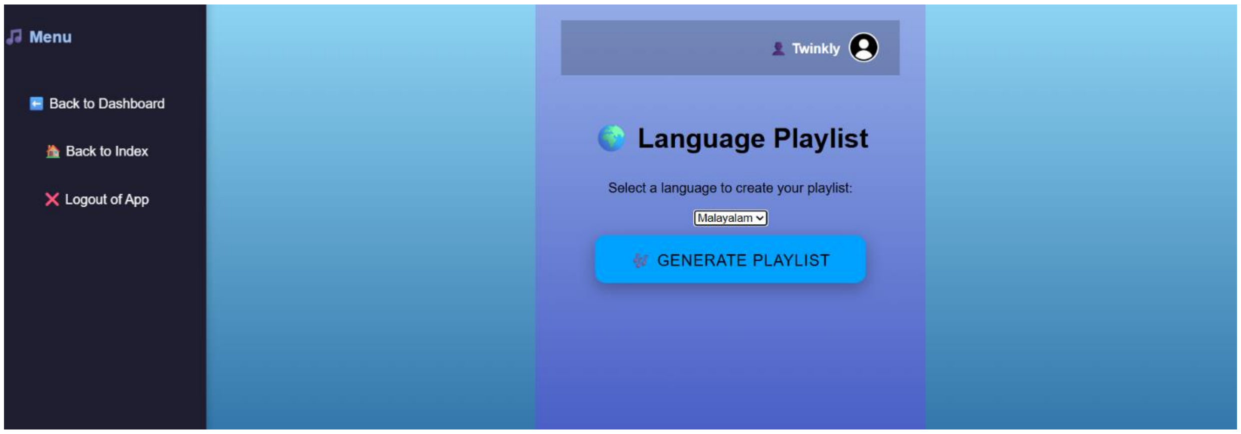


Figure 4.3:genre playlist

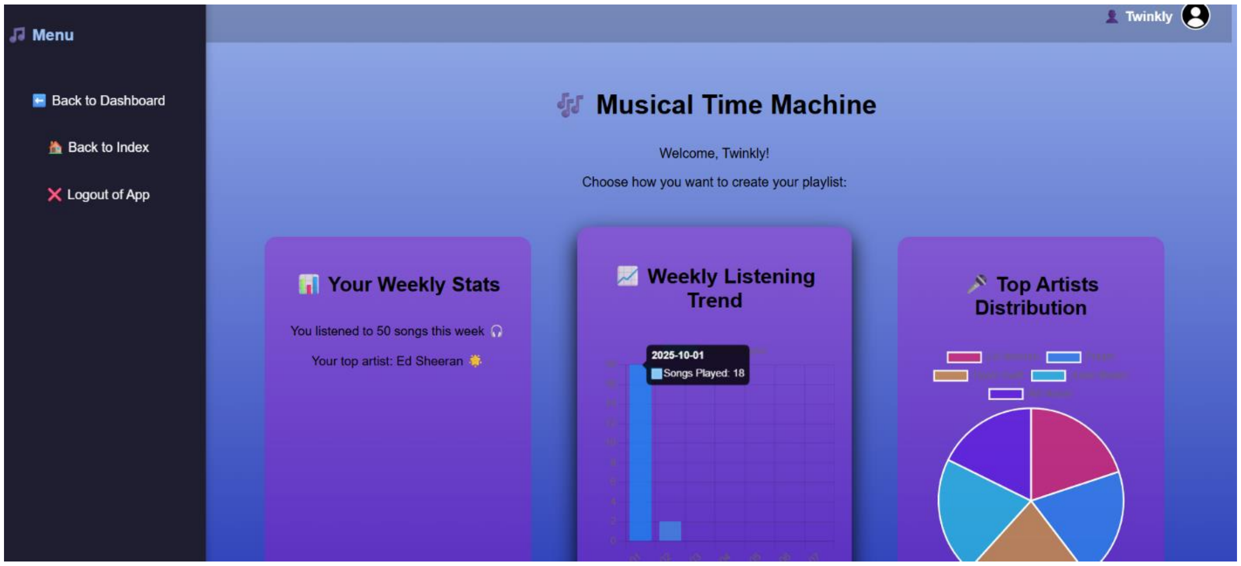


Select a language to create your playlist:

Malayalam

Figure 4.4:language playlist

Figure 4.3: Genre and Language Playlist Generator This form enables users to select a genre (e.g., Pop, Jazz, Rock) or a language (e.g., Hindi, Spanish) to create a customized playlist. The backend queries Spotify's genre and market endpoints to fetch relevant songs.



Welcome, Twinkly!

Choose how you want to create your playlist:

You listened to 50 songs this week

Your top artist: Ed Sheeran

Figure 4.5:dashboard statistics

Figure 4.4: Dashboard Analytics View The dashboard displays user statistics such as top artists, most played tracks, and weekly listening activity. It uses Chart.js to render interactive graphs, helping users visualize their music habits.

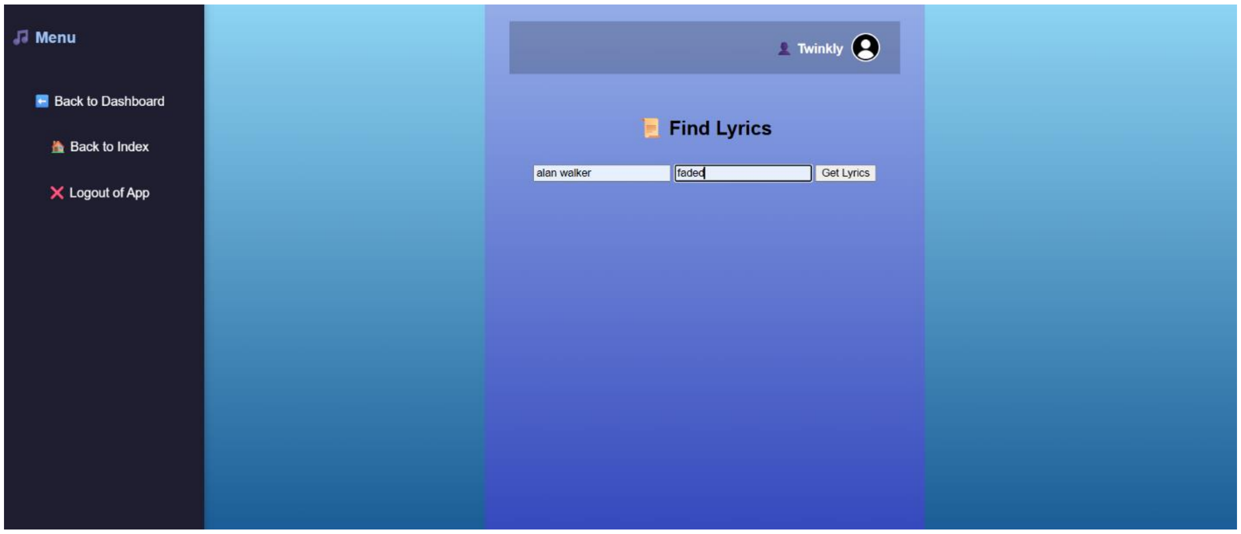
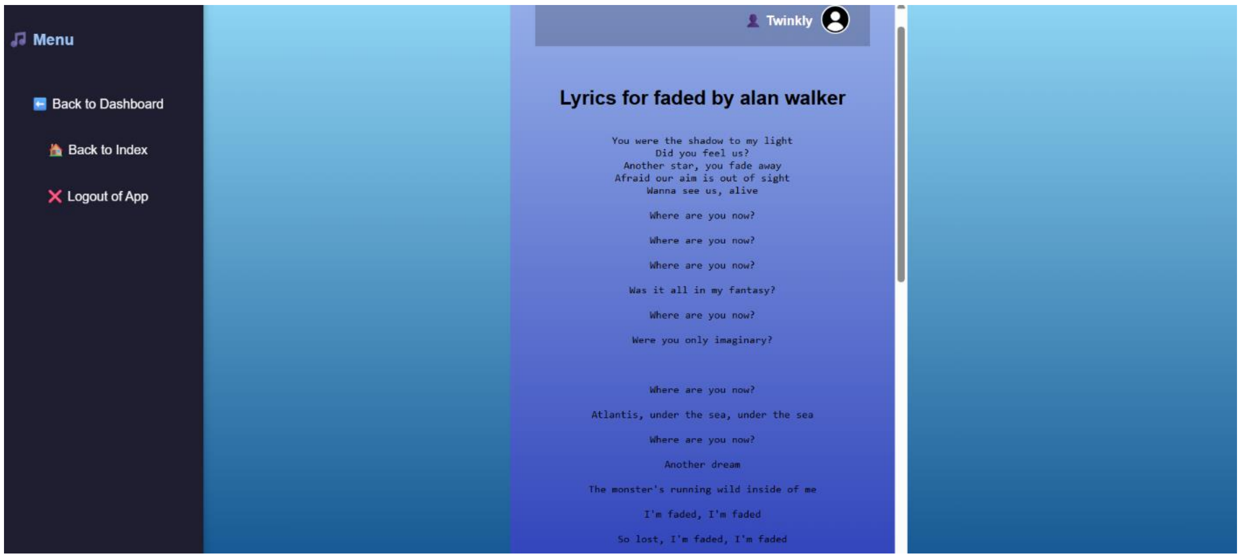


Figure 4.6:lyrics page

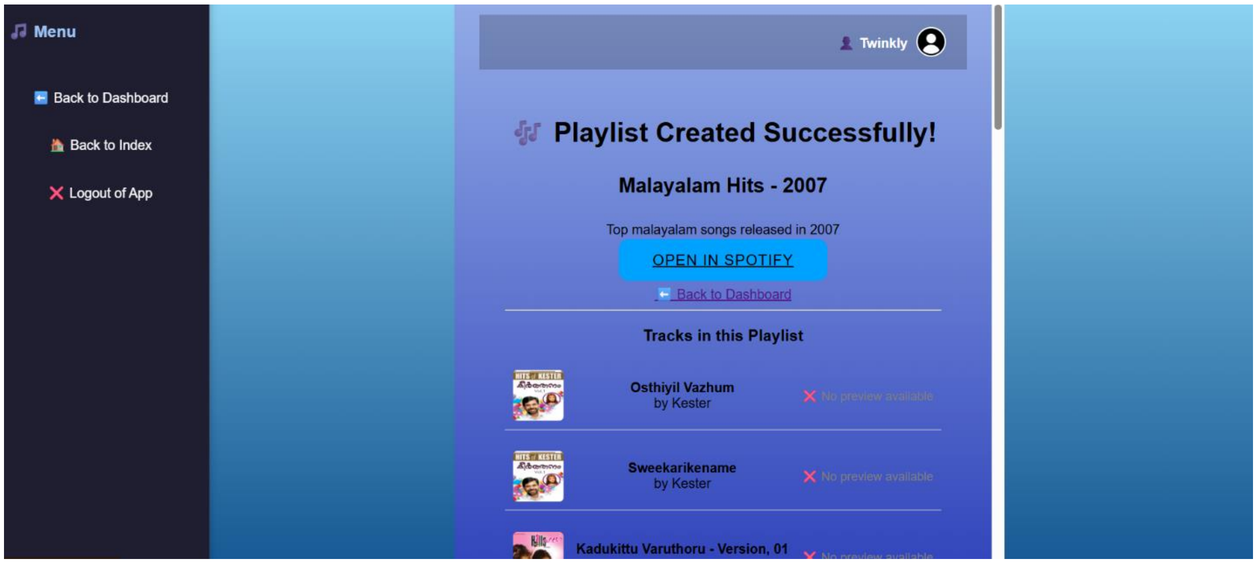


Lyrics for faded by alan walker

```
You were the shadow to my light
Did you feel us?
Another star, you fade away
Afraid our aim is out of sight
Wanna see us, alive

Where are you now?

Where are you now?

Where are you now?

Was it all in my fantasy?

Where are you now?

Were you only imaginary?


Where are you now?

Atlantis, under the sea, under the sea

Where are you now?

Another dream

The monster's running wild inside of me

I'm faded, I'm faded

So lost, I'm faded, I'm faded
```

Figure 4.7:lyrics result

Figure 4.5: Lyrics Finder Interface Users can input a song title and artist name to retrieve lyrics using a third-party API. This feature enhances the listening experience by providing lyrical context.



Playlist Created Successfully!

Malayalam Hits - 2007

Top malayalam songs released in 2007

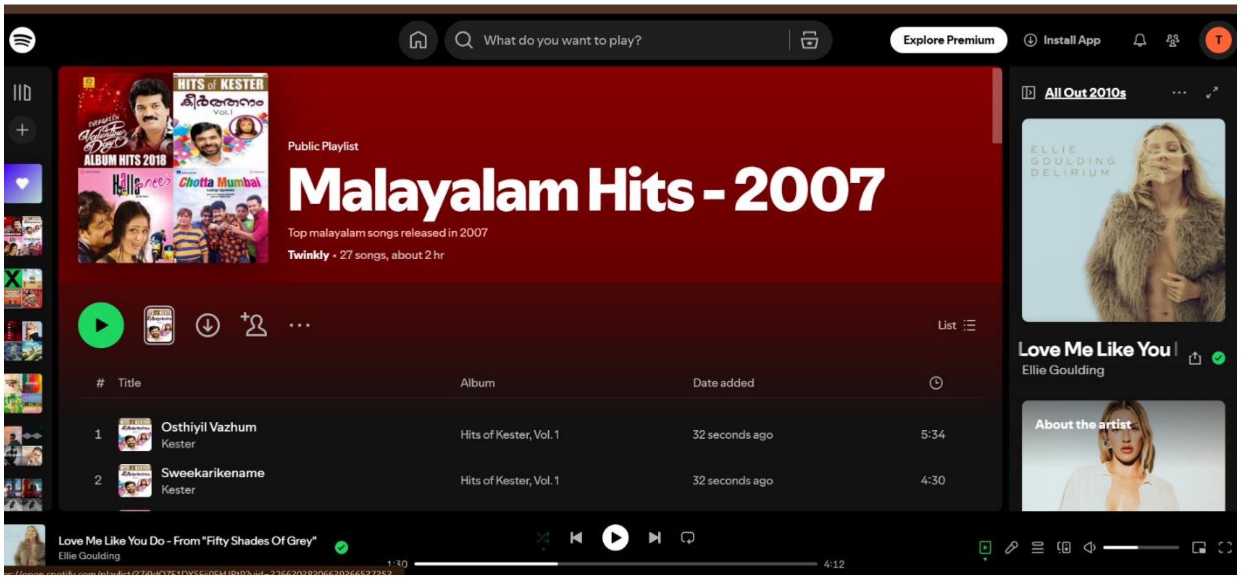
Tracks in this Playlist

Osthiyil Vazhum
by Kester — No preview available

Sweekarikename
by Kester — No preview available

Kadukittu Varuthoru - Version, 01 — No preview available

Figure 4.8:playlist created



Public Playlist

Malayalam Hits - 2007

Top malayalam songs released in 2007

Twinkly • 27 songs, about 2 hr

#	Title	Album	Date added	
1	Osthiyil Vazhum Kester	Hits of Kester, Vol. 1	32 seconds ago	5:34
2	Sweekarikename Kester	Hits of Kester, Vol. 1	32 seconds ago	4:30

Love Me Like You Do - From "Fifty Shades Of Grey"
Ellie Goulding

Figure 4.9:playlist created in spotify

Chapter 5. Conclusion

The Musical Time Machine project presents a compelling solution for enhancing the music listening experience through automation, personalization, and data-driven insights. Designed as a Flask-based web application, it allows users to log in with their Spotify account and generate playlists using multiple modes such as Time Travel, Mood, Genre, and Language. By integrating the Spotify Web API and Billboard Hot 100 data, the system offers a rich and diverse selection of music tailored to individual preferences. One of the standout features is the interactive dashboard, which visualizes user statistics like top artists and weekly activity using Chart.js. These functionalities not only simplify playlist creation but also encourage users to explore their musical habits in a more meaningful way.

Despite its achievements, the project faced certain limitations during development. The absence of a dedicated internal database restricted long-term data storage and user preference caching. Additionally, the lyrics module was only partially implemented due to constraints with third-party API access and rate limits. These challenges emerged primarily due to time constraints and limited access to premium API services. In future iterations, these limitations can be addressed by integrating a lightweight database such as SQLite for storing user metadata and exploring more reliable lyrics APIs with broader access. Overall, Musical Time Machine demonstrates a strong foundation for scalable growth and offers a user-centric platform that can be expanded with advanced features like music recommendations, collaborative playlists, and real-time listening analytics.

References

[1] Spotify for Developers – Spotify Web API Documentation

https://developer.spotify.com/documentation/web-api

[2] Spotipy Library (Python Wrapper for Spotify API) – GitHub Repository & Documentation

https://spotipy.readthedocs.io/

[3] Flask Framework Documentation – Python Micro Web Framework

https://flask.palletsprojects.com/

[4] 4. Python Requests Library – Official Documentation

https://docs.python-requests.org/

[5] 5. BeautifulSoup (bs4) – Web Scraping Library Documentation

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[6] 6. Billboard Hot 100 Charts – Music Chart Source

https://www.billboard.com/charts/hot-100

[7] 7. Chart.js Library – Open Source JavaScript Charting Library

https://www.chartjs.org/docs/latest/

[8] 8. Jinja2 Templating Engine – Used in Flask for HTML Templates

https://jinja.palletsprojects.com/

Appendix

Appendix A Workflow Diagram

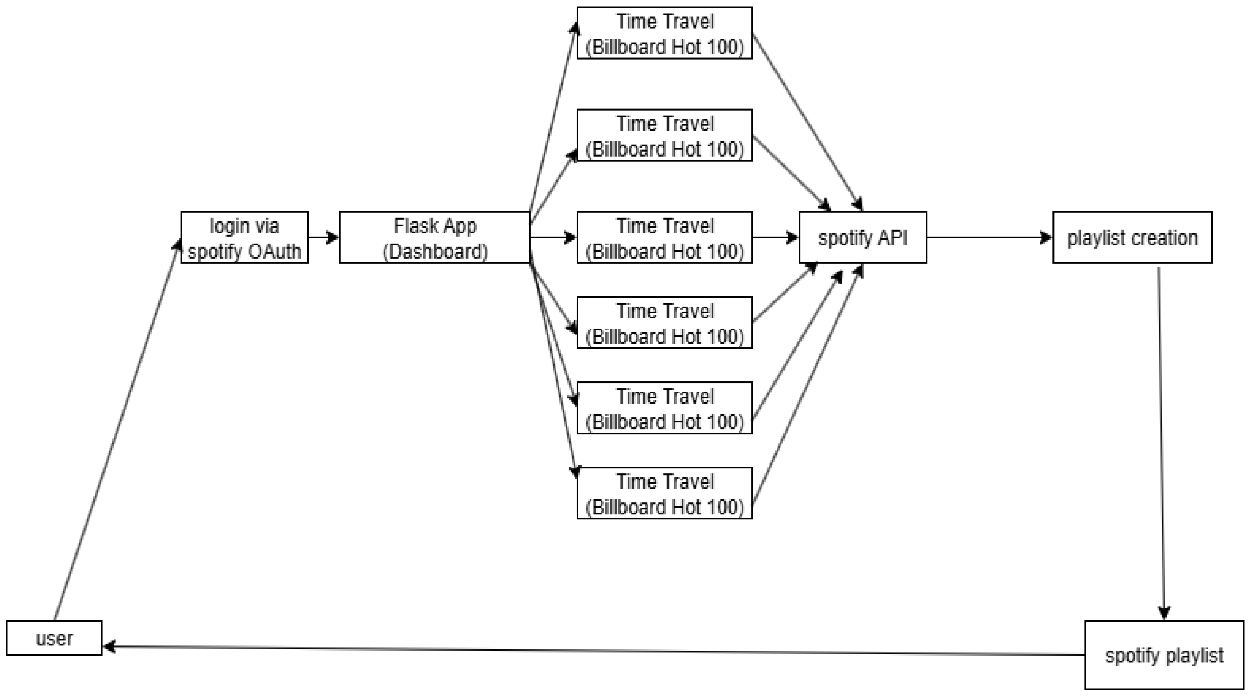

login via spotify OAuth
Flask App (Dashboard)
Time Travel (Billboard Hot 100)
Time Travel (Billboard Hot 100)
Time Travel (Billboard Hot 100)
Time Travel (Billboard Hot 100)
Time Travel (Billboard Hot 100)
Time Travel (Billboard Hot 100)
spotify API
playlist creation
spotify playlist
user

Figure 0.1:Workflow diagram

Appendix B Source code

Features/auth.py

```
from flask import Blueprint, redirect, render_template, request, session, url_for
import spotipy
import glob, os
from config import sp_oauth
auth_bp = Blueprint("auth", __name__)
@auth_bp.route("/")
def index():
    return render_template("index.html")
@auth_bp.route("/login")
def login():
    auth_url = sp_oauth.get_authorize_url()
    print("Redirecting to Spotify:", auth_url)
    return redirect(auth_url)
@auth_bp.route("/callback")
def callback():
    code = request.args.get("code")
    token_info = sp_oauth.get_access_token(code, as_dict=True)
    session["token_info"] = token_info
    sp = spotipy.Spotify(auth=token_info["access_token"])
    user = sp.current_user()
    session["user_id"] = user["id"]
    session["user_name"] = user["display_name"]
    return redirect(url_for("dashboard.dashboard"))
@auth_bp.route("/logout")
def logout():
    session.clear()
    for f in glob.glob(".cache*"):
        os.remove(f)
    return render_template("logout.html")
```

Features/dashboard.py

```
from flask import Blueprint, render_template, session, redirect, url_for
from utils import get_spotify_client
dashboard_bp = Blueprint("dashboard", __name__, url_prefix="/dashboard")
@dashboard_bp.route("/")
def dashboard():
    if "token_info" not in session:
        return redirect(url_for("auth.index"))
    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))
    try:
        # Recently played songs (for weekly counts)
        recently_played = sp.current_user_recently_played(limit=50)
        # Extract daily counts
        from collections import Counter
        import datetime
        dates = [
            item["played_at"][:10] for item in recently_played.get("items", [])
        ]
        date_counts = Counter(dates)
        # Ensure we cover last 7 days
        today = datetime.date.today()
        labels = [(today - datetime.timedelta(days=i)).isoformat() for i in range(6, -1, -1)]
        weekly_counts = [date_counts.get(day, 0) for day in labels]
        # Top artists distribution
        top_artists = sp.current_user_top_artists(limit=5, time_range="short_term")
        artist_labels = [a["name"] for a in top_artists["items"]]
        artist_counts = [a["popularity"] for a in top_artists["items"]]  # use popularity score
        stats = {
            "songs_this_week": len(recently_played["items"]) if recently_played else 0,
            "top_artist": artist_labels[0] if artist_labels else "Unknown",
            "weekly_labels": labels,
            "weekly_counts": weekly_counts,
            "artist_labels": artist_labels,
            "artist_counts": artist_counts,
        }
    except Exception as e:
        print("⚠ Error fetching dashboard stats:", e)
        stats = {
            "songs_this_week": 0,
            "top_artist": "Unknown",
            "weekly_labels": [],
            "weekly_counts": [],
            "artist_labels": [],
            "artist_counts": [],
        }
    return render_template("dashboard.jinja2", user_name=session.get("user_name", "Guest"),
stats=stats)
```

Features/helpers.py

```
import spotipy
from flask import session
def get_spotify_client():
    token_info = session.get("token_info", None)
    if not token_info:
        return None
    return spotipy.Spotify(auth=token_info["access_token"])
def create_playlist(sp, name, description):
    return sp.user_playlist_create(
        user=session["user_id"],
        name=name,
        public=False,
        description=description
    )
```

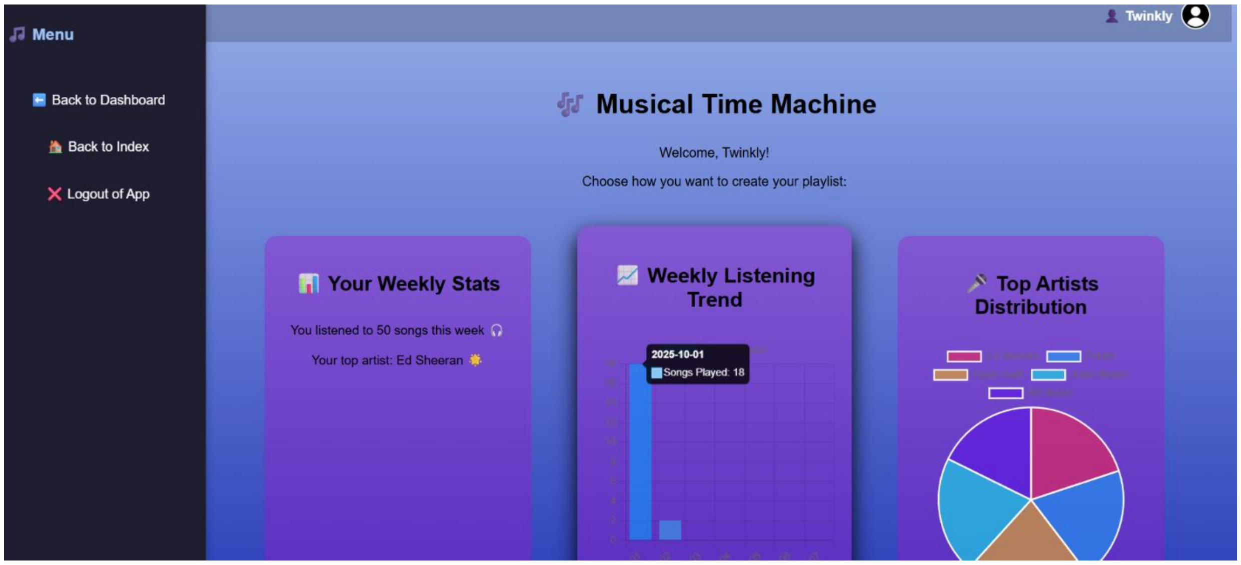
Features/time_travel.py

```
from flask import Blueprint, render_template, request, redirect, url_for
from utils import get_spotify_client, create_playlist
import requests
from bs4 import BeautifulSoup

time_travel_bp = Blueprint("time_travel", __name__)
def get_billboard_hot_100(year, max_songs=50):
    """Scrape Billboard Hot 100 for Dec 31 of given year"""
    url = f"https://www.billboard.com/charts/hot-100/{year}-12-31"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    songs = [s.get_text(strip=True) for s in soup.select("li ul li h3")]
    return songs[:max_songs]
def fetch_tracks(sp, query, year, market="US", max_songs=50):
    """Search Spotify in multiple batches and filter by year"""
    uris = []

    for offset in range(0, 200, 50):
        results = sp.search(q=query, type="track", limit=50, offset=offset, market=market)
        for item in results["tracks"]["items"]:
            release_date = item["album"].get("release_date", "")
            release_year = release_date.split("-")[0] if release_date else ""
            if release_year == year and item["uri"] not in uris:
                uris.append(item["uri"])
            if len(uris) >= max_songs:
                return uris
    return uris
```

```
@time_travel_bp.route("/time_travel_page")
def time_travel_page():
    return render_template("time_travel.html")



@time_travel_bp.route("/time_travel", methods=["POST"])
def time_travel():
    year = request.form["year"]
    language = request.form["language"].lower()
    song_count = int(request.form.get("song_count", 30))  # slider value
    source = request.form.get("source", "spotify")        # only relevant if English
    sp = get_spotify_client()
    if not sp:

        return redirect(url_for("auth.index"))
    uris = []
    # Language → Market mapping
    language_market = {
        "hindi": "IN",
        "tamil": "IN",
        "malayalam": "IN",
        "telugu": "IN",
        "kannada": "IN",
        "spanish": "ES",
        "korean": "KR",
        "japanese": "JP",
        "english": "US"
    }

    market = language_market.get(language, "US")
    # For English → let user pick Billboard or Spotify
    if language == "english":
        if source == "billboard":
            songs = get_billboard_hot_100(year, max_songs=song_count)
            for song in songs:
                try:
                    result = sp.search(q=f"track:{song} year:{year}", type="track", limit=1,
market="US")

                    if result and result["tracks"]["items"]:
                        uris.append(result["tracks"]["items"][0]["uri"])
                except Exception:
                    continue
        else:  # Spotify
            query = f"english hits {year}"
            uris = fetch_tracks(sp, query, year, "US", max_songs=song_count)
    else:

        # Non-English → use Spotify search
        query = f"{language} hits {year}"
        uris = fetch_tracks(sp, query, year, market, max_songs=song_count)
  # fallback if too few
        if len(uris) < song_count // 2:
            alt_query = f"{language} top songs {year}"
            extra = fetch_tracks(sp, alt_query, year, market, max_songs=song_count - len(uris))
            uris.extend(extra)

    # Create playlist
    playlist = create_playlist(
        sp,
        f"{language.capitalize()} Hits - {year}",
        f"Top {language} songs released in {year}"
    )

    tracks = []
    if uris:
        for i in range(0, len(uris), 100):
            sp.playlist_add_items(playlist["id"], uris[i:i+100])
        tracks = [sp.track(uri) for uri in uris]

        return render_template(
            "playlist.html",
            playlist=playlist,
            tracks=tracks,
            added_count=len(uris)
        )
    else:

     return render_template(
        "playlist.html",
        playlist=playlist,
        tracks=[],   # empty if no tracks
        added_count=0,
        message=f"⚠ No matching {language} songs found for {year}. Try another year."
    )
```

Appendix C Screenshots

Dashboard



Menu

Back to Dashboard

Back to Index

Logout of App

Twinkly

Musical Time Machine

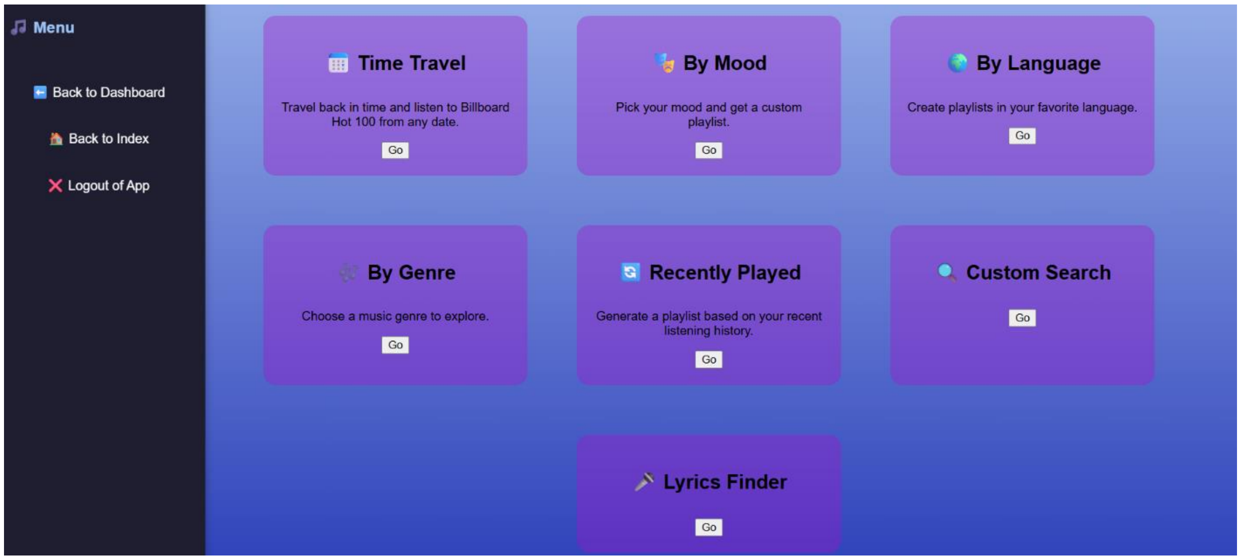
Welcome, Twinkly!

Choose how you want to create your playlist:

Your Weekly Stats

You listened to 50 songs this week

Your top artist: Ed Sheeran



Menu

Back to Dashboard

Back to Index

Logout of App

Time Travel

Travel back in time and listen to Billboard Hot 100 from any date.

Go

By Mood

Pick your mood and get a custom playlist.

Go

By Language

Create playlists in your favorite language.

Go

By Genre

Choose a music genre to explore.

Go

Recently Played

Generate a playlist based on your recent listening history.

Go

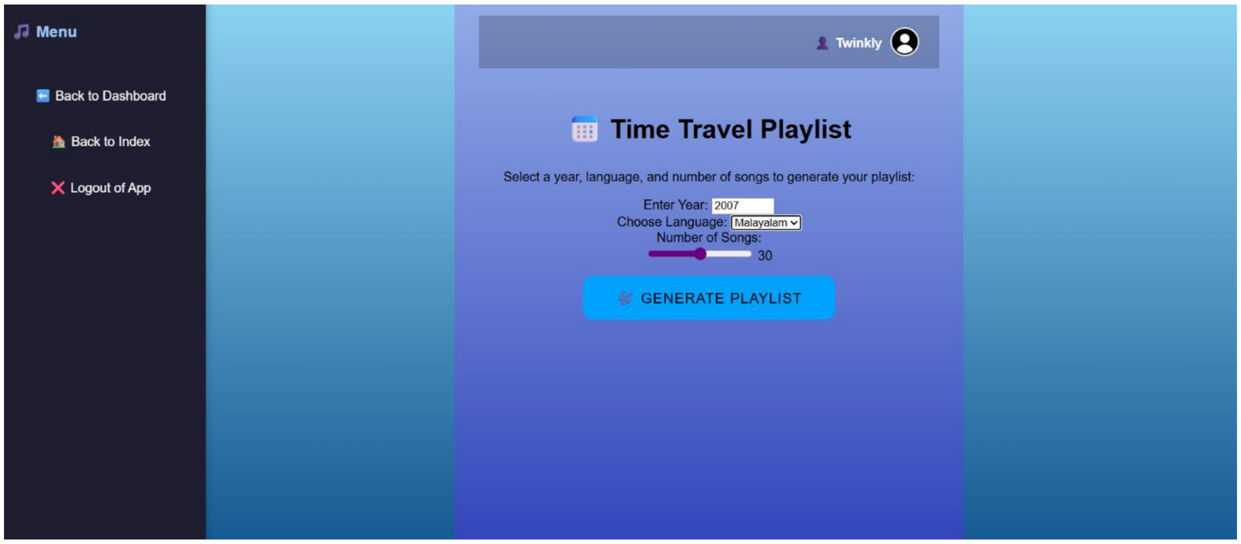
Custom Search

Go

Lyrics Finder

Go

time travel

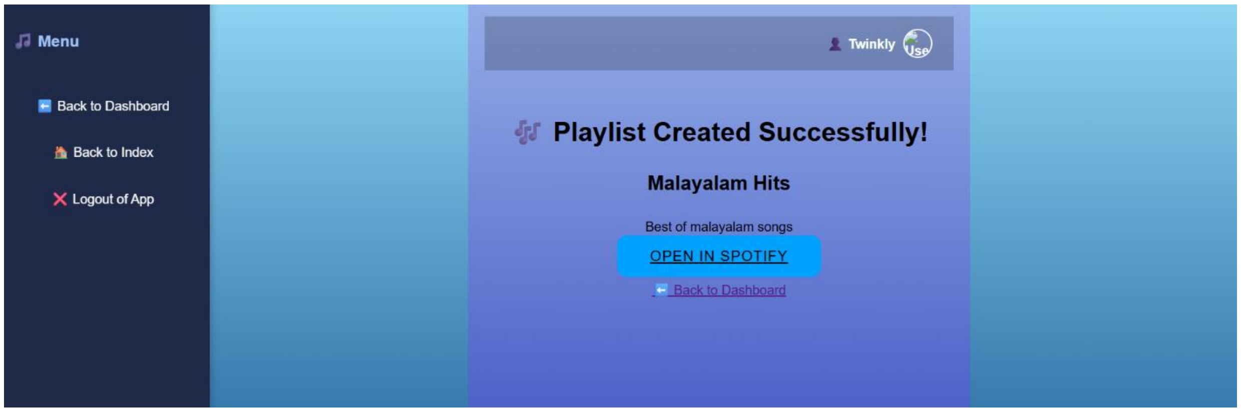


Time Travel Playlist

Select a year, language, and number of songs to generate your playlist:

Enter Year: 2007
Choose Language: Malayalam
Number of Songs: 30

Playlist page

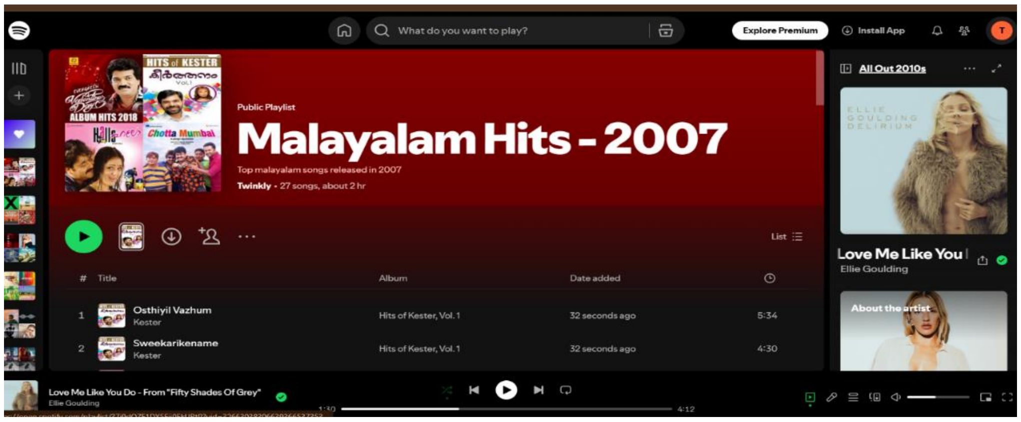


Playlist Created Successfully!

Malayalam Hits

Best of malayalam songs

Spotify playlist



Public Playlist

Malayalam Hits - 2007

Top malayalam songs released in 2007

Twinkly • 27 songs, about 2 hr

#	Title	Album	Date added	
1	Osthiyil Vazhum Kester	Hits of Kester, Vol. 1	32 seconds ago	5:34
2	Sweekarikename Kester	Hits of Kester, Vol. 1	32 seconds ago	4:30

Love Me Like You Do - From "Fifty Shades Of Grey"
Ellie Goulding
1:30 4:12