MUSICAL TIME MACHINE

SCRUM MASTER

ROSHNA SHIRIN M

MES24MCA-2046

Department of Computer Applications

MES College of Engineering, Kuttippuram

21/08/2025

PRODUCT OWNER

RESHMI K

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER APPLICATIONS

MES COLLEGE OF ENGINEERING, KUTTIPPURAM

TABLE OF CONTENTS

MUSICAL TIME MACHINE

- Musical Time Machine is a Flask-based web application integrated with Spotify API.
- Allows users to log in with their Spotify account using OAuth 2.0.
- Generates personalized playlists based on:
  - Time Travel (Billboard Hot 100 by year)
  - Mood (Happy, Sad, Chill, Party)
  - Language (English, Hindi, Tamil, Malayalam, etc.)
  - Genre (Pop, Rock, Jazz, EDM, etc.)
  - Recently Played history

MUSICAL TIME MACHINE

- Uses Spotipy (Python client for Spotify Web API) for playlist creation.
- Scrapes Billboard Hot 100 for historical top English songs.
- Adds support for regional languages using Spotify's search and market-based queries.
- Provides a simple and interactive web dashboard (HTML + CSS + Flask templates).
- API integration
- Web development with Flask
- Data scraping
- Multi-user authentication

OBJECTIVES

- To develop a web-based playlist generator using Flask and Spotify API.
- To implement secure user authentication with Spotify OAuth (multi-user support).
- To provide multiple ways for users to create playlists:
  - Based on Billboard charts (time travel feature).
  - By mood, language, and genre preferences.
  - From recently played songs.
- To integrate web scraping (Billboard Hot 100) for historical data.
- To practice and demonstrate concepts of API integration, data handling, and web development.
- To create a user-friendly dashboard for playlist generation.

EXISTING SYSTEM

- Users rely on Spotify app or website for listening to music and creating playlists.
- Spotify provides curated playlists (e.g., Daily Mix, Release Radar, Mood/Genre playlists).
- Recently Played section is available but not automatically converted into playlists.
- No option to time travel to past music charts (e.g., Billboard Hot 100 by date).
- Limited support for multi-language playlist creation in one place.
- Users need to manually search and add songs to playlists.

PROPOSED SYSTEM

- A Flask-based web application integrated with Spotify API.
- Provides multi-user login using Spotify OAuth 2.0.
- Allows users to generate playlists automatically based on:
  - Billboard Hot 100 (Time Travel)
  - Mood preferences
  - Language choices
  - Music genres
  - Recently played songs
- Uses web scraping (Billboard) and Spotify search for fetching songs.
- Eliminates the need for manual playlist creation.
- Offers a simple, user-friendly dashboard for easy playlist generation.

MOTIVATIONS

- Music is an essential part of daily life and people often look for personalized playlists
- Manually creating playlists is time-consuming and repetitive
- Existing platforms like Spotify lack a direct feature to explore past charts such as Billboard Hot 100
- Regional and language-based playlist creation is limited in the current system
- Need for a simple web-based tool that automatically generates playlists based on user preferences
- Opportunity to learn and apply concepts of Flask, API integration, and web scraping in a real-world scenario

FUNCTIONALITIES

- User Authentication
  Secure login using Spotify OAuth 2.0 for multiple users
- Time Travel Playlist
  Generates playlists from Billboard Hot 100 based on a selected date (year, month, day)
- Mood-based Playlist
  Creates playlists depending on the user's mood such as happy, sad, chill, or party
- Language-based Playlist
  Allows users to create playlists in preferred languages like English, Hindi, Tamil, Malayalam

FUNCTIONALITIES

•Genre-based Playlist

Generates playlists by selecting genres such as Pop, Rock, Jazz, Classical, EDM, Hip-Hop

•Recently Played Playlist

Converts a users recently played tracks into a new playlist

•Dashboard Interface

User-friendly web dashboard for selecting features and generating playlists

•Logout Feature

Secure logout option with session clearing and Spotify account sign-out

MODULE DESCRIPTION

•User Authentication Module
Handles Spotify login using OAuth 2.0 and manages user sessions
•Dashboard Module
Provides a central interface where users can choose playlist generation options
•Time Travel Module
Scrapes Billboard Hot 100 for a given date and creates a playlist with top songs from that period
•Mood-based Module
Generates playlists by mapping moods (happy, sad, chill, party) to suitable tracks

MODULE DESCRIPTION

•Language-based Module

Creates playlists based on the user's selected language by searching Spotify content

•Genre-based Module

Allows users to pick a genre (Pop, Rock, Jazz, Classical, EDM, Hip-Hop) and generates playlists

•Recently Played Module

Fetches the users recently played songs and compiles them into a playlist

•Logout Module

Clears user session data and provides an option to log out from both the app and Spotify

DEVELOPING ENVIRONMENT

- Operating System: Windows 10 / Linux (development and testing)
- Programming Language: Python 3.x
- Framework: Flask (web framework)
- Front End: HTML, CSS, Jinja2 templates
- Back End: Flask with Spotipy (Spotify API integration)
- Database: Not required (session-based storage only)
- Web Scraping: BeautifulSoup (for Billboard charts)
- APIs: Spotify Web API (via Spotipy)
- IDE / Editor: Visual Studio Code
- Package Manager: pip (Python package installer)
- Version Control: Git and GitHub

PRODUCT BACKLOG

ID	NAME	PRIORITY <high/medium/low>	ESTIMATE (Hours)	STATUS <Planned/In progress/Completed>
1	User Authentication	High	5	Completed
2	Dashboard	High	5	In Progress
3	Mood-based Playlist	High	4	In progress
4	Language-based Playlist	High	6	planned
5	Genre-based Playlist	High	3	planned
6	Recently Played Playlist	High	2	planned

PRODUCT BACKLOG

ID	NAME	PRIORITY <high/medium/low>	ESTIMATE (Hours)	STATUS <Planned/In progress/Completed>
7	Playlist Management	High	3	planned
8	Error Handling	High	7	planned
9	UI/UX Enhancements	Medium	4	planned
10	Documentation & Reports	High	6	planned
11	Future Enhancements (Optional)	low	4	planned

USER STORY

User Story ID	As a type of User	I want to <Perform some task>	So that i can <Achieve Some Goal>
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 date	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

USER STORY

User Story ID	As a type of User	I want to <Perform some task>	So that i can <Achieve Some Goal>
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

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

PROJECT PLAN

User StoryID	Task Name	Start Date	End Date	Days	Status
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

SPRINT BACKLOG

Backlog tem	Status And Completion Date	Original Estimation in Hours	Day 1 hrs	Day 2 hrs	Day 3 hrs	Day 4 hrs	Day 5 hrs	Day 6 hrs	Day 7 hrs	Day 8 hrs	Day 9 hrs	Day 10 hrs
SPRINT1												
User authentication	7/8/2025	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

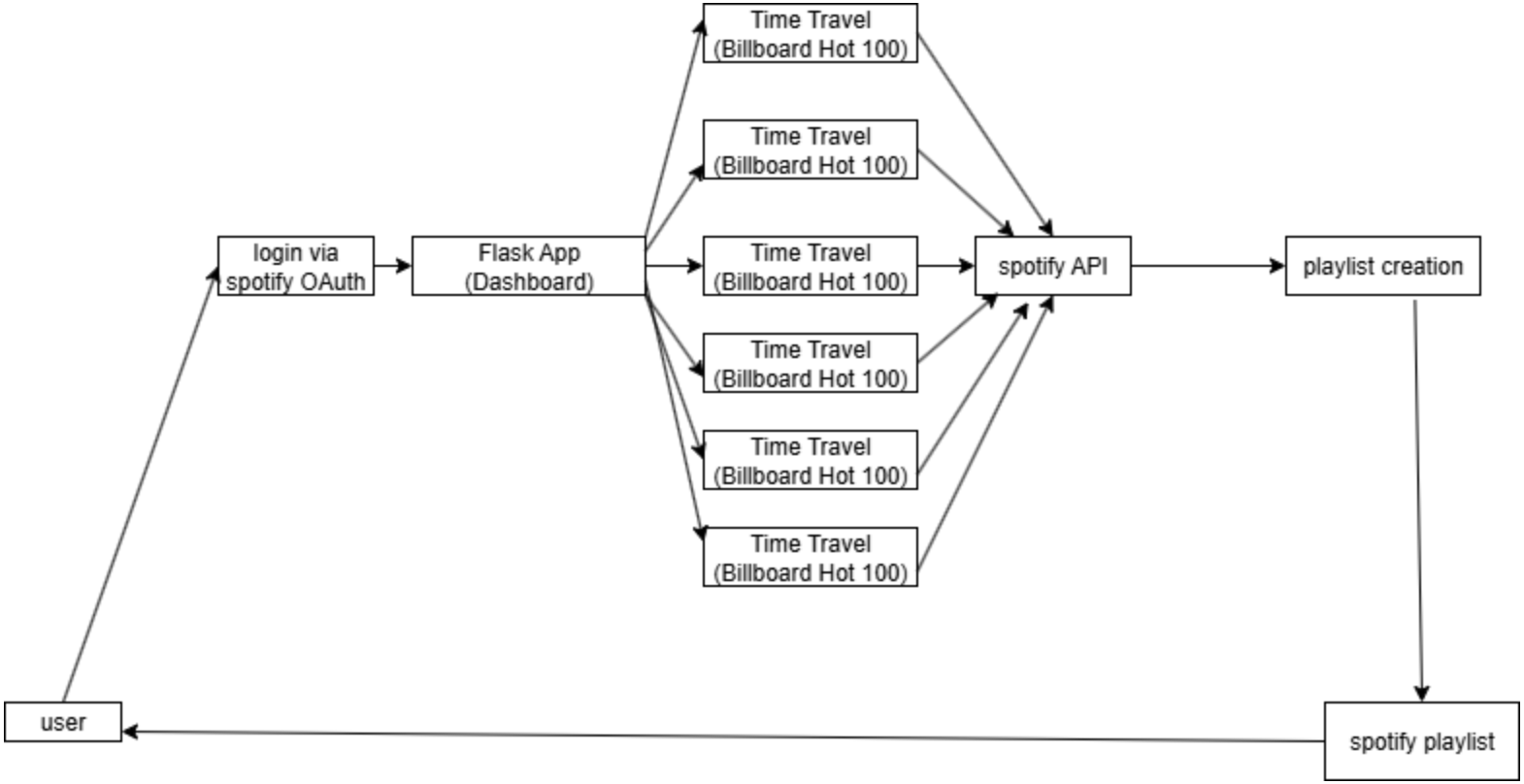
SPRINT BACKLOG

Backlog tem	Status And Completion Date	Original Estimation in Hours	Day 1 hrs	Day 2 hrs	Day 3 hrs	Day 4 hrs	Day 5 hrs	Day 6 hrs	Day 7 hrs	Day 8 hrs	Day 9 hrs	Day 10 hrs
SPRINT 2												
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

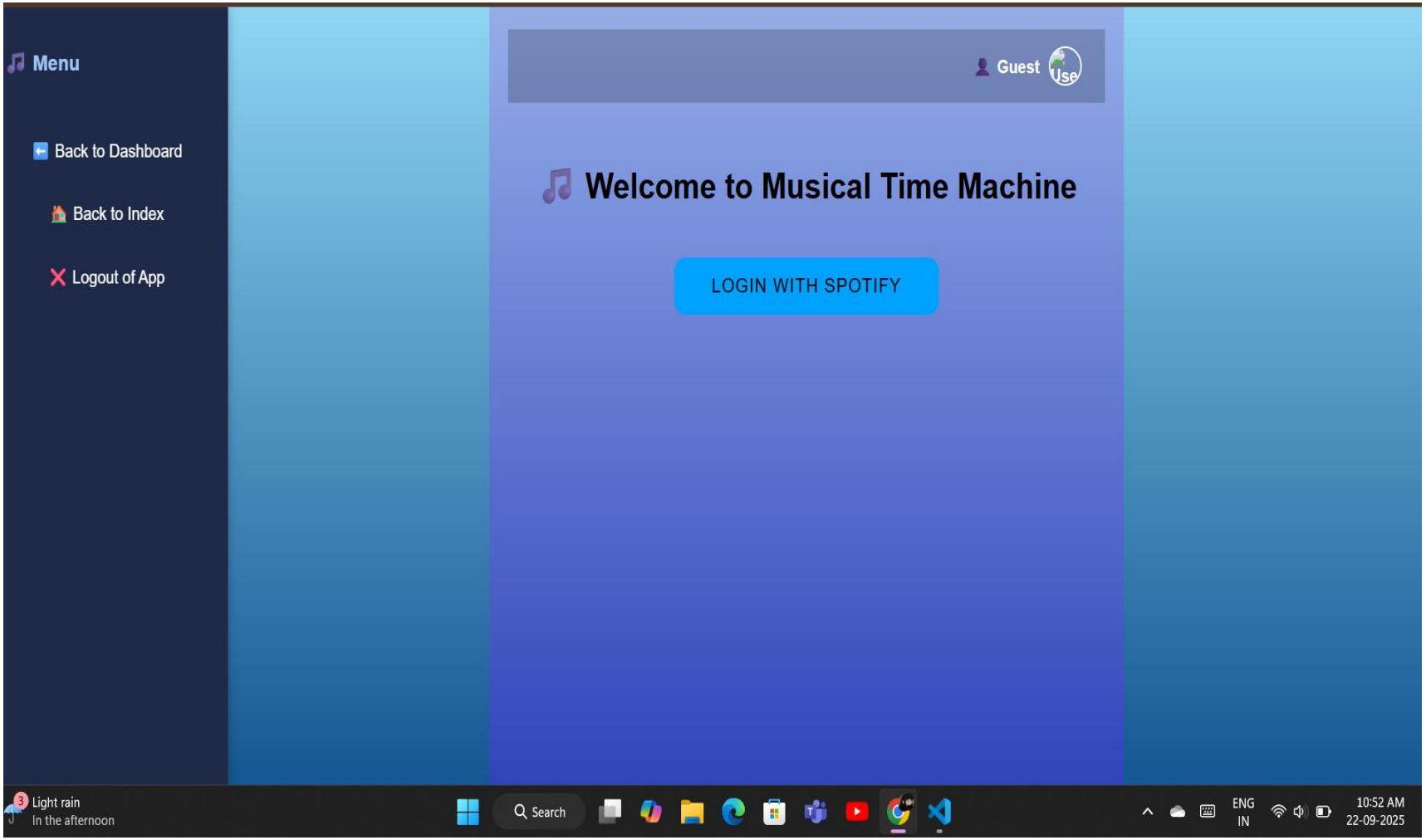
SPRINT BACKLOG

Backlog tem	Status And Completion Date	Original Estimation in Hours	Day 1 hrs	Day 2 hrs	Day 3 hrs	Day 4 hrs	Day 5 hrs	Day 6 hrs	Day 7 hrs	Day 8 hrs	Day 9 hrs	Day 10 hrs
SPRINT 3												
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

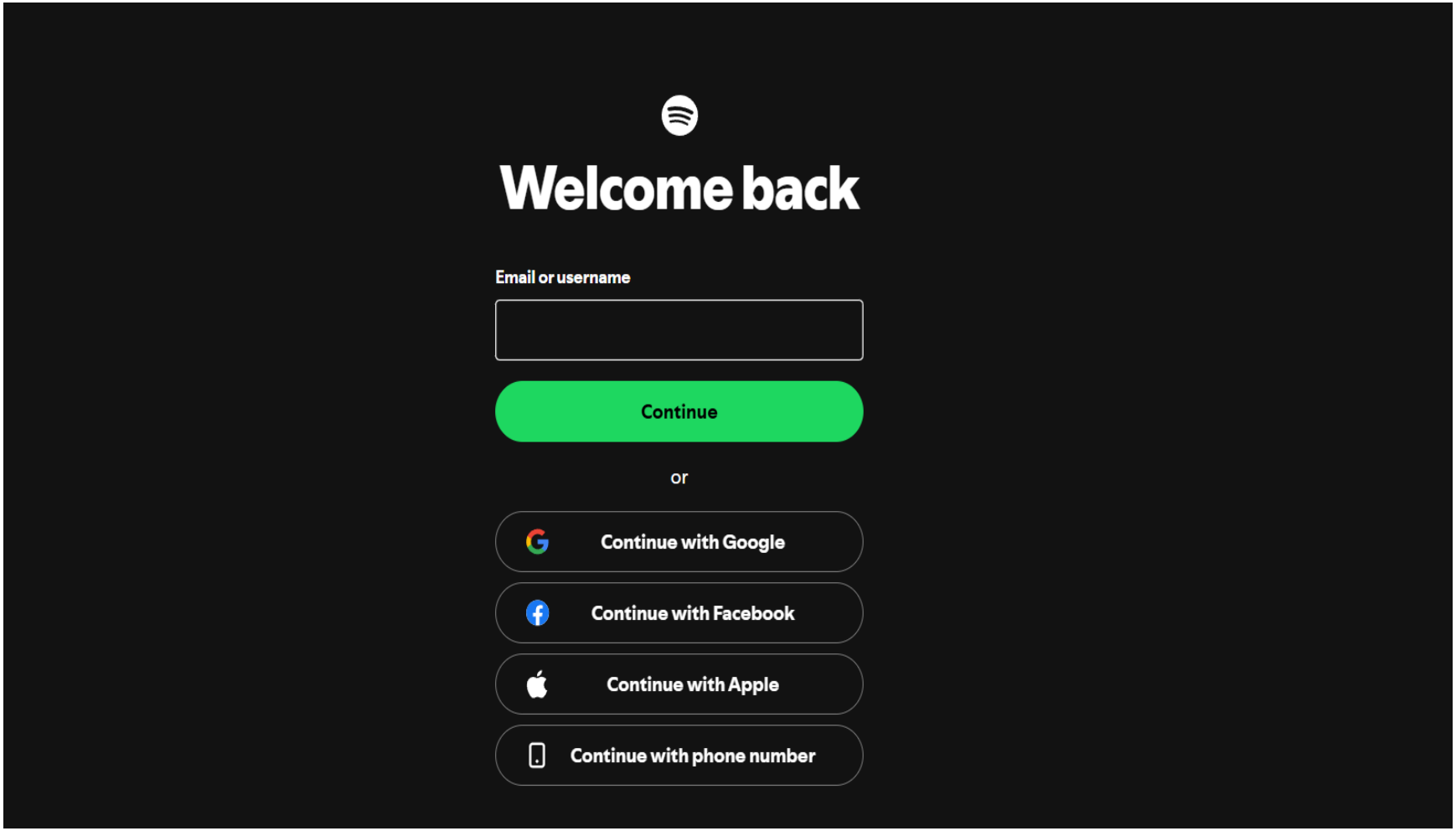
WORK FLOW DIAGRAM



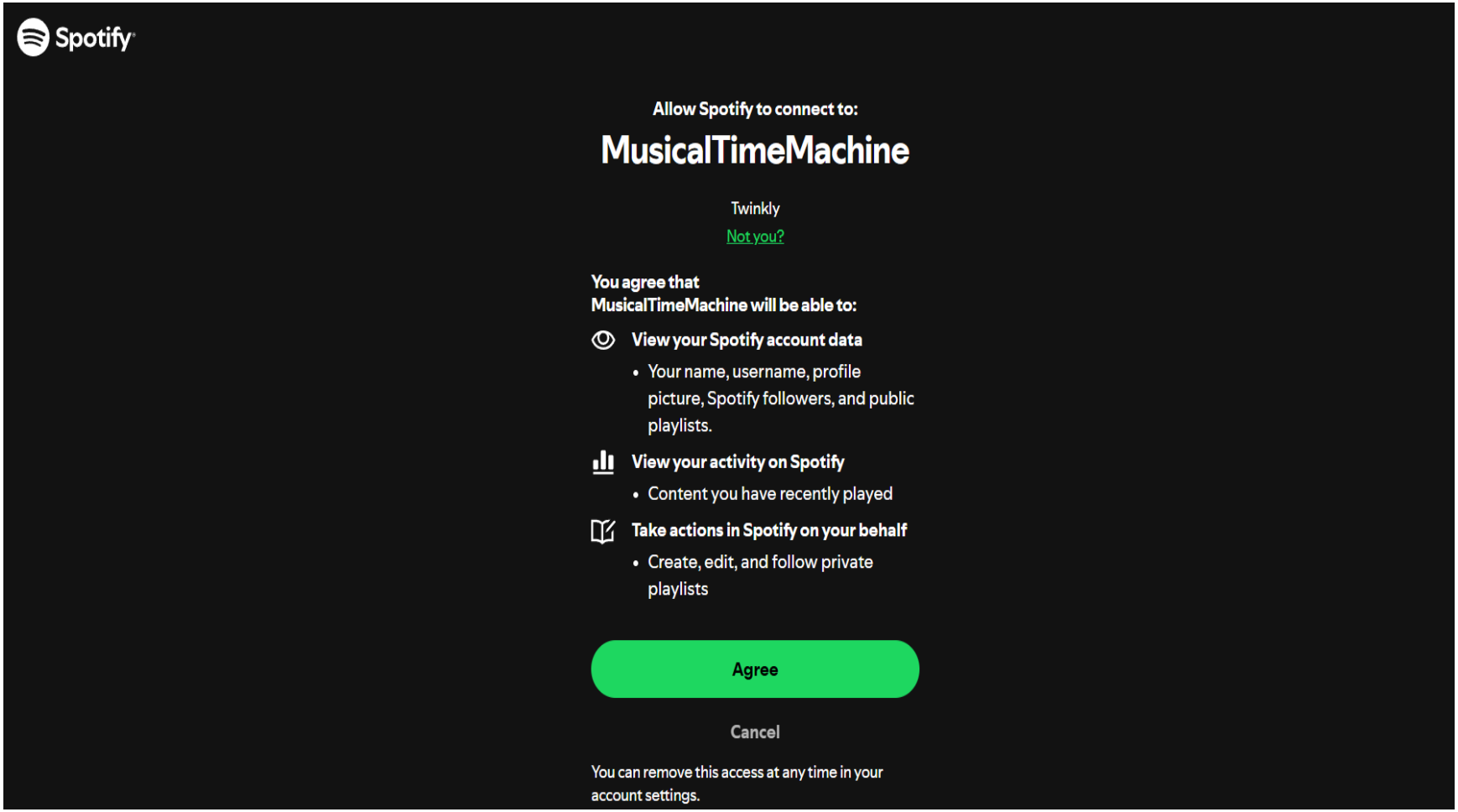
interface


Menu
Back to Dashboard
Back to Index
Logout of App
Guest
Welcome to Musical Time Machine
LOGIN WITH SPOTIFY
Light rain
In the afternoon
Search
ENG
IN
10:52 AM
22-09-2025

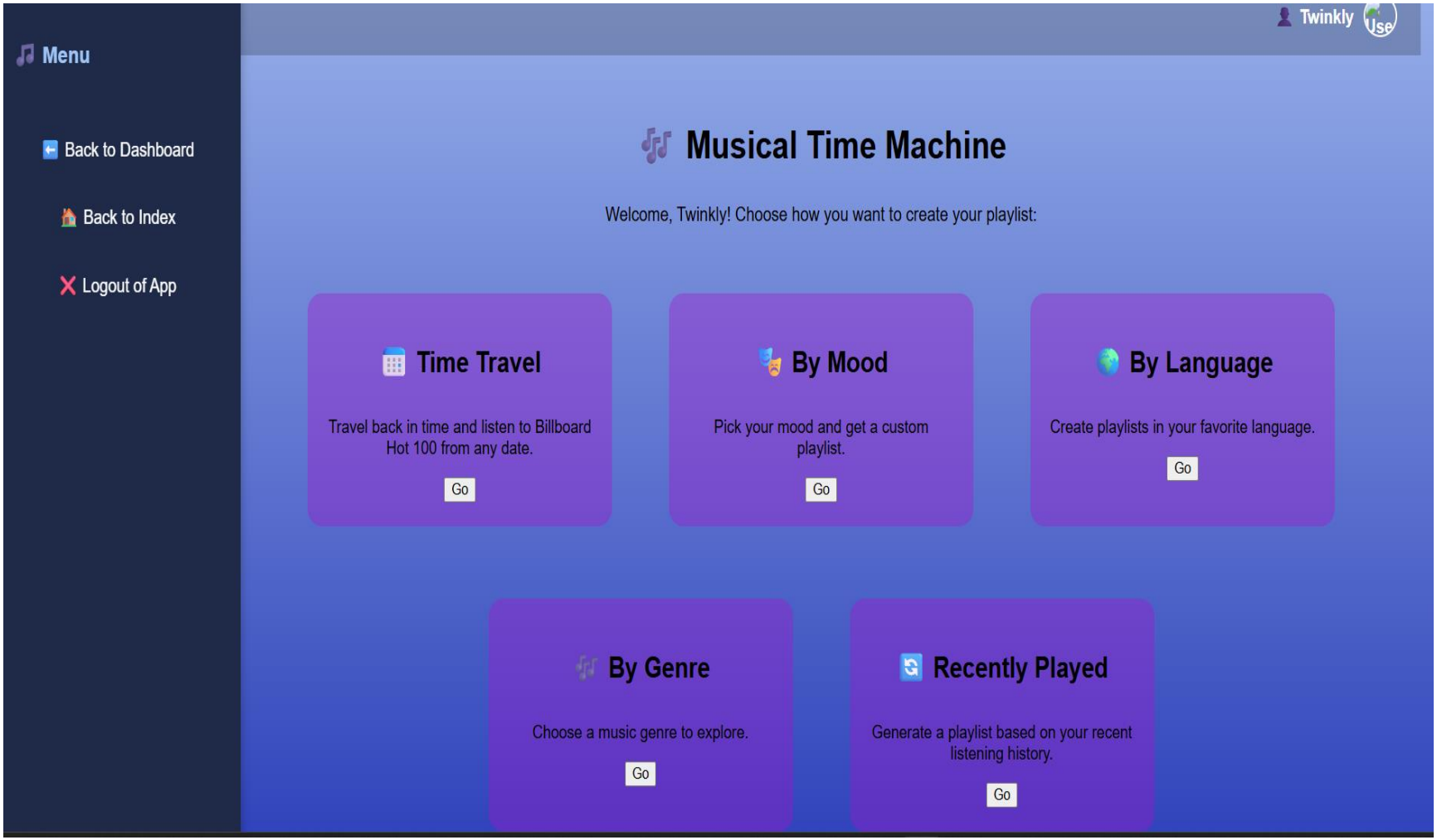
interface


Welcome back
Email or username
Continue
or
Continue with Google
Continue with Facebook
Continue with Apple
Continue with phone number

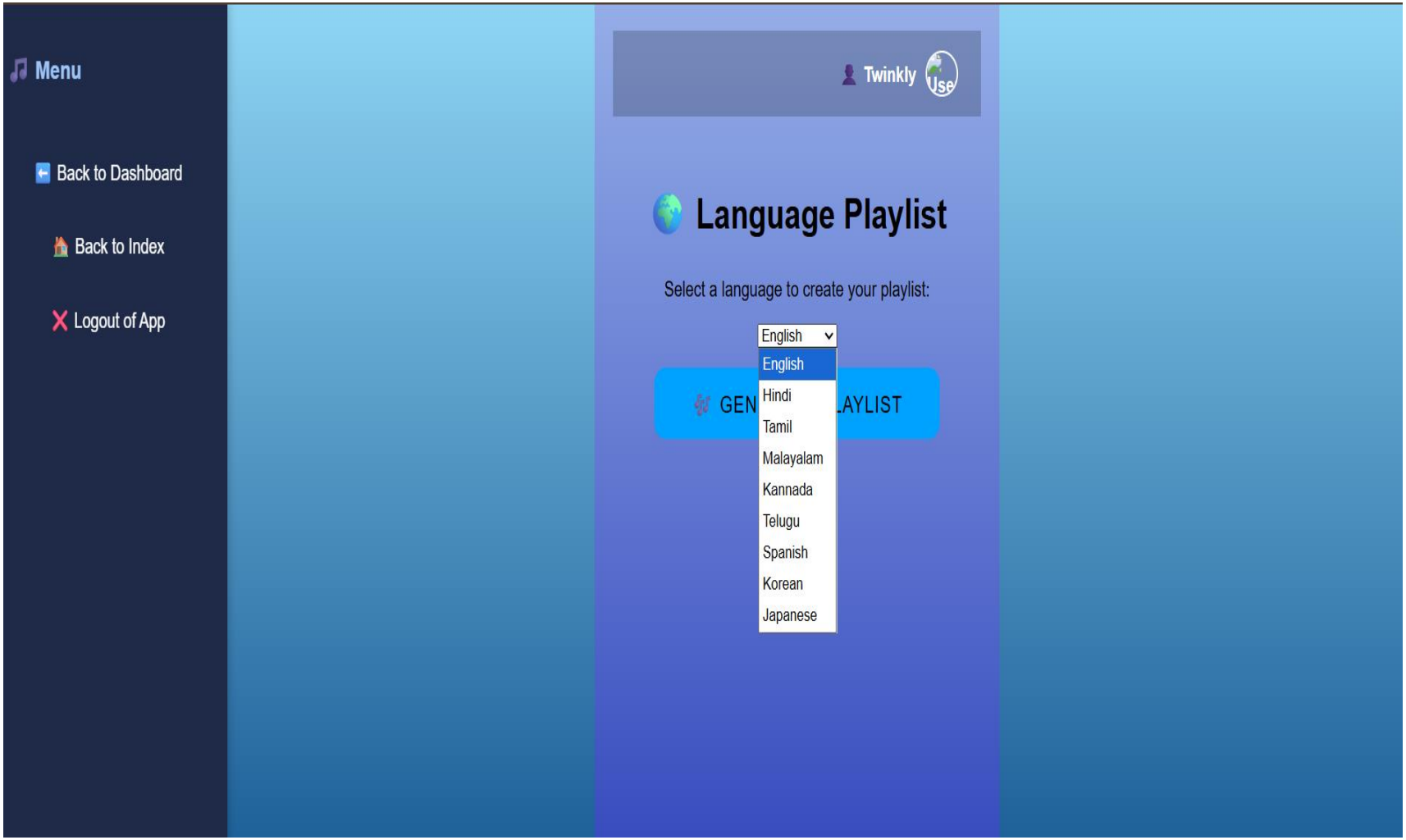
interface


Spotify
Allow Spotify to connect to:
MusicalTimeMachine
Twinkly
Not you?
You agree that
MusicalTimeMachine will be able to:
View your Spotify account data
• Your name, username, profile picture, Spotify followers, and public playlists.
View your activity on Spotify
• Content you have recently played
Take actions in Spotify on your behalf
• Create, edit, and follow private playlists
Agree
Cancel
You can remove this access at any time in your account settings.

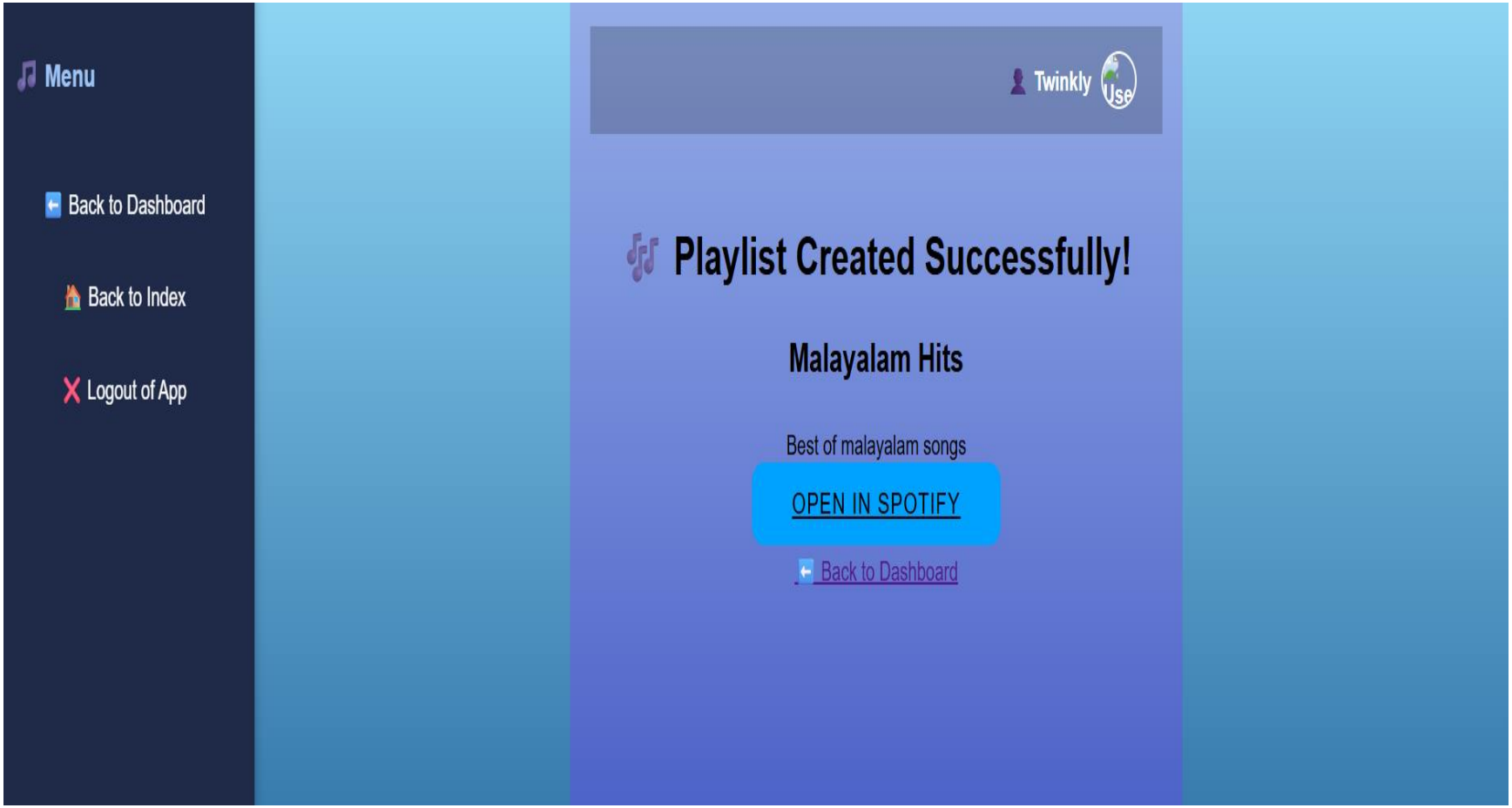
interface


Menu
Back to Dashboard
Back to Index
Logout of App
Twinkly
Musical Time Machine
Welcome, Twinkly! Choose how you want to create your playlist:
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

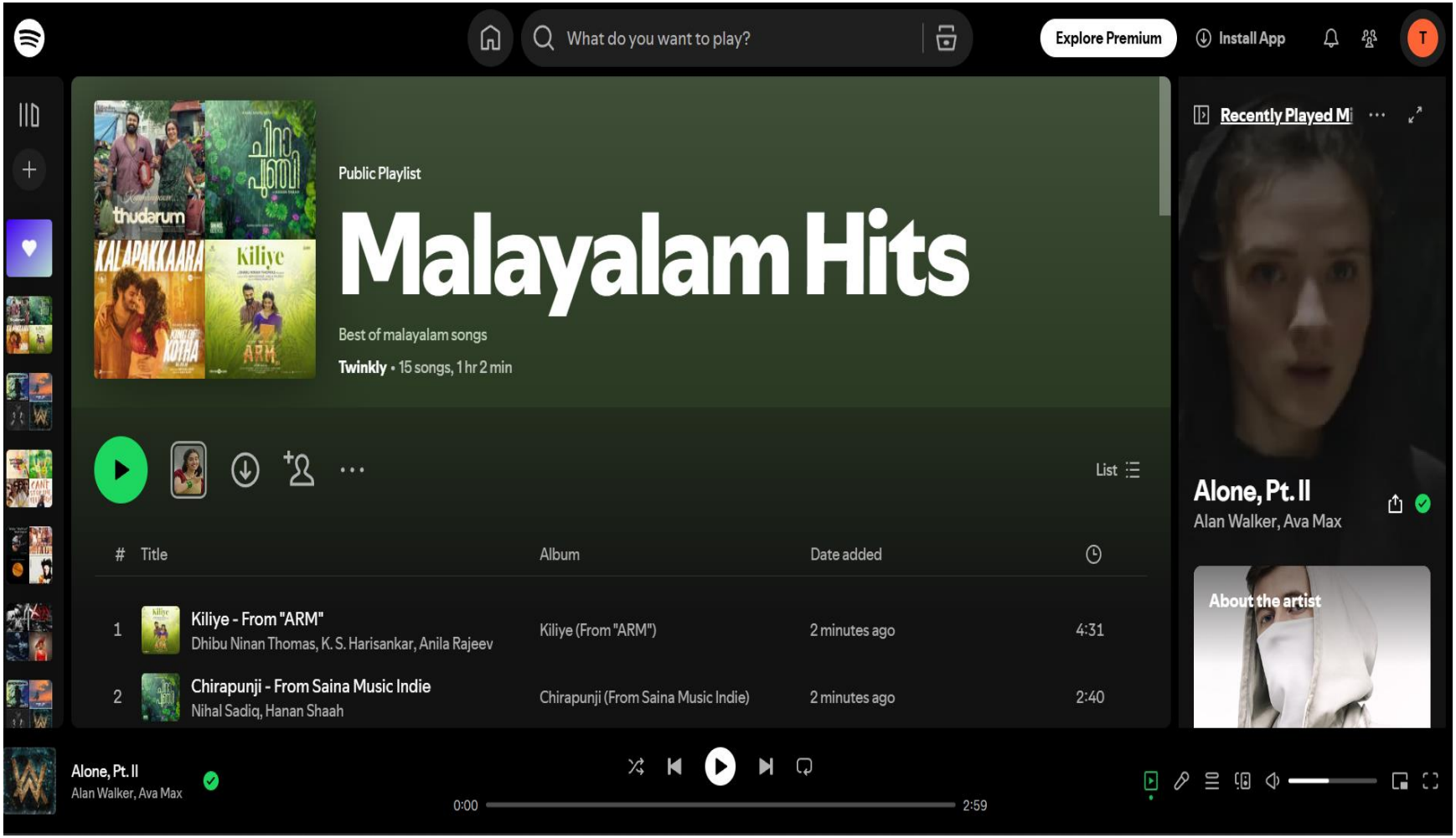
interface


Menu
Back to Dashboard
Back to Index
Logout of App
Twinkly
Language Playlist
Select a language to create your playlist:
English
English
Hindi
Tamil
Malayalam
Kannada
Telugu
Spanish
Korean
Japanese

interface


Menu
Back to Dashboard
Back to Index
Logout of App
Twinkly
Playlist Created Successfully!
Malayalam Hits
Best of malayalam songs
OPEN IN SPOTIFY
Back to Dashboard

interface


What do you want to play?
Explore Premium
Install App
Public Playlist
Malayalam Hits
Best of malayalam songs
Twinkly • 15 songs, 1 hr 2 min
List
Title
Album
Date added
1 Kiliye - From "ARM"
Dhibu Ninan Thomas, K. S. Harisankar, Anila Rajeev
Kiliye (From "ARM")
2 minutes ago
4:31
2 Chirapunji - From Saina Music Indie
Nihal Sadiq, Hanan Shaah
Chirapunji (From Saina Music Indie)
2 minutes ago
2:40
Recently Played Mi
Alone, Pt. II
Alan Walker, Ava Max
About the artist
Alone, Pt. II
Alan Walker, Ava Max
0:00
2:59

App.py

```python
from flask import Flask
from features.auth import auth_bp
from features.dashboard import dashboard_bp
from features.time_travel import time_travel_bp
from features.mood import mood_bp


app = Flask(__name__)
app.secret_key = "supersecretkey"


app.register_blueprint(auth_bp)
app.register_blueprint(dashboard_bp, url_prefix="/dashboard")
app.register_blueprint(time_travel_bp, url_prefix="/time_travel")
app.register_blueprint(mood_bp, url_prefix="/mood")
```

Main.py

```
1    from app import app
2    if __name__ == "__main__":
3    |
4        app.run(debug=True, port=8888)
```

Utils.py

```python
from flask import session
import spotipy


def get_spotify_client():
    """Return a Spotify client using stored session token"""
    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):
    """Create a private playlist for the logged-in user"""
    return sp.user_playlist_create(
        user=session["user_id"],
        name=name,
        public=False,
        description=description
    )
```

Time_travel.py

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


time_travel_bp = Blueprint("time_travel", __name__)


@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():
    date = request.form["date"]
    language = request.form["language"].lower()

    sp = get_spotify_client()
    if not sp:
        return redirect(url_for("auth.index"))

    year = date[:4]
    uris = []

```

Time_travel.py

```
28          language_market = {
29              "hindi": "IN",
30              "tamil": "IN",
31              "malayalam": "IN",
32              "kannada": "IN",
33              "telugu": "IN",
34              "spanish": "ES",
35              "korean": "KR",
36              "japanese": "JP",
37          }
38          market = language_market.get(language, "US")
39
40          if language == "english":
41
42              url = f"https://www.billboard.com/charts/hot-100/{date}"
43              response = requests.get(url)
44              soup = BeautifulSoup(response.text, "html.parser")
45              songs = [s.get_text(strip=True) for s in soup.select("li ul li h3")][:50]
46
47              for song in songs:
48                  try:
49                      result = sp.search(q=f"track:{song} year:{year}", type="track", limit=1, market=market)
50                      if result and result["tracks"]["items"]:
51                          uris.append(result["tracks"]["items"][0]["uri"])
52                  except Exception:
```

Time_travel.py

```
            except Exception:
                continue
    else:

        playlist_query = f"{language} hits {year}"
        try:
            res = sp.search(q=playlist_query, type="playlist", limit=5, market=market)
            playlists = res.get("playlists", {}).get("items", []) if res else []
        except Exception:
            playlists = []

        for pl in playlists:
            if len(uris) >= 40:
                break
            pl_id = pl.get("id")
            if not pl_id:
                continue
            try:
                pl_items = sp.playlist_items(pl_id, fields="items.track.album.release_date,items.track.uri", limit=50)
            except Exception:
                pl_items = {}
            for it in pl_items.get("items", []):
                track = it.get("track")
                if not track:
                    continue
```

Time_travel.py

```
                    continue
                rel = track.get("album", {}).get("release_date", "")
                rel_year = rel.split("-")[0] if rel else ""
                if rel_year == year or rel_year == "":
                    if track.get("uri") and track["uri"] not in uris:
                        uris.append(track["uri"])
                if len(uris) >= 40:
                    break


        if len(uris) < 20:
            track_query = f"{language} hits {year}"
            try:
                res = sp.search(q=track_query, type="track", limit=30, market=market)
                tracks = res.get("tracks", {}).get("items", []) if res else []
            except Exception:
                tracks = []
            for item in tracks:
                rel = item.get("album", {}).get("release_date", "")
                rel_year = rel.split("-")[0] if rel else ""
                if rel_year == year or rel_year == "":
                    if item.get("uri") and item["uri"] not in uris:
                        uris.append(item["uri"])
                if len(uris) >= 40:
                    break
```

Time_travel.py

```
100                     break
101
102         playlist = create_playlist(
103             sp,
104             f"{language.capitalize()} Hits - {date}",
105             f"Top {language} songs from {date}"
106         )
107
108         if uris:
109
110             for i in range(0, len(uris), 100):
111                 sp.playlist_add_items(playlist["id"], uris[i:i+100])
112             return render_template("playlist.html", playlist=playlist, added_count=len(uris))
113         else:
114             return render_template(
115                 "playlist.html",
116                 playlist=playlist,
117                 added_count=0,
118                 message="⚠️ No matching songs found for this language/date. Try another year or language."
119             )
```

Mood.py

```python
from flask import Blueprint, render_template, request, redirect, url_for
from utils import get_spotify_client, create_playlist

mood_bp = Blueprint("mood", __name__)

@mood_bp.route("/mood_page")
def mood_page():
    return render_template("mood.html")

@mood_bp.route("/mood", methods=["POST"])
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")
```

Mood.py

```
22        }
23        query = mood_map.get(mood, "music")
24
25        result = sp.search(q=query, type="track", limit=20)
26        uris = [item["uri"] for item in result["tracks"]["items"]]
27
28        playlist = create_playlist(sp, f"{mood.capitalize()} Vibes", f"A {mood} mood playlist")
29        if uris:
30            sp.playlist_add_items(playlist["id"], uris)
31
32        return render_template("playlist.html", playlist=playlist)
```

THANK YOU