



GFOSS
OPEN TECHNOLOGIES ALLIANCE

GSOC 2025 PROPOSAL

PersonalAIs

Generative AI Agent for Personalized Music Recommendations

1. Basic Details

Name : Debanjan Rakshit

Email : rakshit.debanjan1@gmail.com

University: Maulana Azad National Institute of Technology Bhopal

Course: Electronics and Communication Engineering

Time zone: GMT +5:30 (Indian Standard Time IST)

Links : [Github](#), [Linkedin](#)

2. Contents

1. Basic Details	1
2. Contents	2
3. About Me	3
4. Past Experiences and Projects	3
5. Certifications:	4
6. Availability	4
7. Why GFOSS ?	5
8. Primary Objective.....	5
9. Primary Features and Goal	5
10. Additional Features	7
11. Major Tech Stack to be used:	8
12. Understanding the Endpoints in Spotify API	8
13. Implementation:	9
14. Process Diagram:	18
15. Building Chat Like Experience/App	19
16. Feedback System Implementation (A challenge).....	22
17. Scalability and Feasibility:	24
18. Expectation from GSOC and GFOSS:	25
19. Timeline/Project Plan:	25

3. About Me

I am currently a sophomore at Maulana Azad National Institute of Technology Bhopal pursuing Electronics and Communications Engineering (2023-27). My keen interests lies in tinkering with programming languages, late night debugging and taking up challenging projects that pushes my boundaries and broadens my area of knowledge.

I have a solid foundation in the core concepts of Machine Learning Algorithms, Artificial Intelligence, Generative AI, Data Structures, Algorithm Design and Analysis.

I am well versed in languages like C/C++, Python, JavaScript and frameworks like Pytorch, Tensorflow, ScikitLearn, Langchain, FastAPI, RestAPIs, ReactJS, HTML, CSS and deployment services like MS Azure, Amazon AWS.

I am a self-taught developer and an enthusiastic individual. I have participated in quite a few hackathons and major achievements have been listed below.

4. Past Experiences and Projects:

1. Winner at Smart India Hackathon 2024 :

- Made an end to end fully deployed web service for the Ministry of Information and Broadcasting, Government of India and deployed using Amazon EC2.
- It was named SLIFTEX (Similarity and Linguistic Filtering for Title Examination).
- It could detect similarities across multiple local languages of India.

Objective:

The objective of the project was to develop an online platform (service) for the ministry so that they are able to filter out titles that are semantically, phonetically and lexically similar to the existing titles of newspaper, journals etc.

2. Innovation Award(3rd) at Smart Innovate Hackathon 2025 :

- Made a smart AI Chat application named PLANTID (Plant Identification and Disease Diagnosis)
- Made an interactive chat application that could detect crops and plant disease using the images uploaded by user and from the symptoms provided by them
- It could understand multilingual symptoms input and also provided accurate and most probable disease prediction using the Geodata and the location of the user.
- Deployed using Microsoft Azure.





3. Summer Internship Project at Engineer Core :

- Made a simple Random Forest Classifier model that could classify emotions of people using social media based on various parameters.

4. Crop Price Predictor :

- Made a Crop Price Prediction model for the Ministry of Foods, Government of India using time series analysis algorithms like ARIMA and also explored LSTM model for providing better results.
- Got selected for Smart India Hackathon 2024

5. Certifications:

- ❖ FreeCodeCamp Data Analysis with Python 
- ❖ FreeCodeCamp Machine Learning with Python 
- ❖ FreeCodeCamp JavaScript Algorithms and Data Structures 
- ❖ Machine Learning from Effervescence IIIT Allahabad 

6. Availability

After carefully reviewing the GSOC timeline, I concluded that I would be able to easily dedicate 40-50 hrs a week towards the project building. Due to college end semester exams from 8th May to 20th May I may not be able to attend to the project but after that I would be able to contribute to the project with full dedication.

7. Why GFOSS ?

The project PersonalAI : Generative AI Agent for Personalised Music Recommendation aligns with my area of interest and also aligns with my past experiences of various projects that I have made. My previous experiences and my knowledge in the field of Generative AI, LLMs makes me take up this project for GSOC 2025. This project focuses on the real-life application of **Generative AI and Large Language Models (LLMs)** that adds meaning to the technology by enhancing **usability, accessibility, and efficiency**. The goal is to leverage AI-driven solutions to simplify complex tasks, improve user interactions, and create **practical, real-world applications** that contribute to **digital empowerment** and **societal impact**.

By integrating **state-of-the-art AI models**, this project aims to demonstrate how Generative AI can go beyond theoretical advancements and be effectively applied in **various small and big** applications of daily use.

8. Primary Objective

The main goal of this project is to create an AI-based music recommendation agent using Generative AI and Large Language Models (LLMs). The smart system will evaluate a variety of parameters like genres, mood, listening history, and contextual preferences to provide highly customized music recommendations. With the power of existing LLMs, the agent will not only recommend songs but also learn from user behaviour and enhance its recommendations over time. The aim is to create an intuitive and interactive music discovery experience, where users can just browse songs that align with their moods, activities, or music preferences.

9. Primary Features and Goal

- **User Login:**

Users will have to login so that using SPOTIFY API we are able to fetch the user activities like:

1. User's Playlist
2. User's Top Tracks and Artist
3. Recently Played Tracks
4. Saved Music Information
5. Followed Artists

This will help us to generate better recommendations using user activities .

For example: The user wants to listen to some energetic songs and the top listened artist is Metallica then we can recommend some good Metallica songs.

- **User selects mood and genres :**

A feature to let the user select the mood and genre according to which they want recommendations. But solely this wouldn't be used to generate the recommendations rather this would be combined with the user history data to make better recommendations.

For example: The user selects mood as romantic and if he has recently played track as some metal song then we can recommend some romantic song played by a metal band as it will best fit his taste.

- **Explore Playlist Options :**

An explore playlist options where the LLM will create playlists as suggestions for the user so that they are able to get more personalised results rather than simply music as recommendations. It would further reduce the hassle to add songs that the AI would generate as a suggestion to some playlist.

The AI may take suggestions from already existing public playlist to get better results.

For example: The LLM may generate a list of 10 songs by itself. The user may select or deselect few of the songs and ask the model to generate more songs that will help us create a customised playlist for the user.

- **Mood Recognition from User Activity or Input:**

If the user shares input of example songs and what they want to listen that will allow the LLM to understand better and provide better recommendations.

For Example : Make a Playlist similar to Blinding Lights by Weekend that will automatically understand the mood of the user.

- **Open Sourced LLM models :**

Using Open Sourced LLM models either locally using Ollama or using ChatGroq's free API Service.

Probable use of LLM Models involves meta llama, deepseek's r1 model etc.

- **Emotion Classification :**

Emotional classification can be done either by using embeddings trained on emotional databases or by letting the LLM itself decide the genres as they have been already pretrained on large datasets, whichever provides the better outcomes and results.

For example: MusicGen is transformer model available on Hugging Face.

10. Additional Features

- **Dynamic Evolution of Playlist:**

Based on the user activities we can continuously collect information about the user activities and that could help us provide much better options.

Refining playlist can also be implemented based on suggestions provided by the LLM.

- **Voice Based Command Integrations:**

We can also work on voice-based command integrations so that it can make the agent more user interactive.

- **User feedback-based improvement system:**

After a user have been given some suggestions, the user can like or dislike, select or deselect the suggested songs that will let the LLM know what songs to recommend further to the user.

These are the basic features that I find that the Agent must have so that it can be user interactive, engaging and serves the purpose of the user. I have also listed some additional

features that can be worked on after the primary objectives have been achieved. Moreover the website for our agent should be easy to use, not much complicated and must grasp the attention of the user so that the user can retain on our website.

11. Major Tech Stack to be used:

- **Frontend:** HTML, CSS, ReactJS, JavaScript
- **Backend:** NodeJS, ExpressJS, MongoDB (if required for storing user database)
- **API Server:** FastAPI or RESTAPI
- **Generative AI:** LangChain (for integrating LLMs), ChatGroq (for free LLM API service) or Ollama (for running open-source LLM locally), Hugging Face Model for transformers (if required like MusicGen)
- **Deployment Service:** MS Azure
- **User Music History:** Spotify API

12. Understanding the Endpoints in Spotify API

Sr No.	Endpoints	Method	Parameters	Uses
1	accounts.spotify.com/authorize	GET	client_id, response_type, redirect_uri, scope, state	To get the user to login into their Spotify account on our platform
2	accounts.spotify.com/api/token	POST	grant_type, code, redirect_uri, client_id, client_secret	Exchange the authorization code for an access token.
3	/me	GET		Get user profile
4	/me/playlists	GET		Get user playlists
5	/me/top/artists	GET		Get top artists

6	/me/top/tracks	GET		Get top tracks
7	/me/player/recently-played	GET		Get recently played tracks
8	/v1/me/playlists	GET		Get current user's playlists.
9	/v1/users/{user_id}/playlists	POST	user_id	Create a new playlist for a user.
10	/v1/playlists/{playlist_id}	GET	playlist_id	Get information about a playlist
11	/v1/playlists/{playlist_id}/tracks	GET	Playlist_id	Get the tracks inside a playlist
12	/v1/search?q={query}&type={type}	GET	Query, type	Search for a song, artist, playlist
13	/v1/recommendations	GET		Get song recommendations based on seed artists, genres, or tracks.
14	/v1/artists/{artist_id}/related-artists	GET	Artist_id	Get similar artists to a given artist.
15	/v1/me/following	GET		Get the artists followed by the user.
16	/v1/me/albums	GET		Get saved albums
17	/v1/me/tracks	GET		Get saved tracks

13. Implementation:

- **User Authentication:**

User needs to login into their spotify accounts so that we are able to fetch their details.

We need to implement Spotify's OAuth Authentication.

```
SPOTIFY_CLIENT_ID=your_client_id
SPOTIFY_CLIENT_SECRET=your_client_secret
SPOTIFY_REDIRECT_URI=http://localhost:8000/callback
```

First of all we need to have the following saved in a .env file

After that the following authentication can be done using the login function as follows:

```
SCOPES = "user-read-private user-read-email playlist-read-private user-
top-read user-read-recently-played"

@app.get("/login")
def login():
    """Redirect user to Spotify login page."""
    auth_params = {
        "client_id": SPOTIFY_CLIENT_ID,
        "response_type": "code",
        "redirect_uri": SPOTIFY_REDIRECT_URI,
        "scope": SCOPES,
    }
    auth_url = f"{AUTH_URL}?{'&'.join([f'{k}={v}' for k, v in
auth_params.items()])}"
    return RedirectResponse(auth_url)
```

We'll need to create a callback endpoint so that the authentication can land at that endpoint:

```

@app.get("/callback")
def callback(code: str):
    """Handle Spotify callback and exchange authorization code for access
    token."""
    auth_payload = {
        "grant_type": "authorization_code",
        "code": code,
        "redirect_uri": SPOTIFY_REDIRECT_URI,
        "client_id": SPOTIFY_CLIENT_ID,
        "client_secret": SPOTIFY_CLIENT_SECRET,
    }

    response = requests.post(TOKEN_URL, data=auth_payload)

    if response.status_code != 200:
        raise HTTPException(status_code=400, detail="Error fetching
        token")

    token_info = response.json()
    access_token = token_info["access_token"]
    refresh_token = token_info["refresh_token"]

    # Store token (Use a database in production)
    tokens["access_token"] = access_token
    tokens["refresh_token"] = refresh_token

    return {"message": "Authentication successful!", "access_token":
    access_token}

```

- **User Data Retrieval:**

Using Spotify's Api we can now get various data of the users. Few example usages have been shown below

1. To get the playlist of the user

```
def playlists():  
    response = requests.get(f"{API_BASE_URL}me/playlists", headers =  
        get_headers())  
    return response.json()
```

2. To get the top tracks of the user

```
def top_tracks():  
    response = requests.get(f"{API_BASE_URL}me/top/tracks", headers =  
        get_headers())  
    return response.json()
```

3. To get the top artists of the user

```
def top_artists():  
    response = requests.get(f"{API_BASE_URL}me/top/artists", headers =  
        get_headers())  
    return response.json()
```

Furthermore we can obtain more user details that we need to complete the objectives.

- **Generating Suggestion from the LLMS:**

Now we can send the user details to the LLM so that it can generate the suggestion using the mood details and the preferences of the user

We can do so by making use of LangChain. The selected moods and preference of the user can be woven into a ChatPrompt by using LangChain's ChatPromptTemplate and then can be sent to the LLM that will generate suggestions automatically.

Example of such Chat Prompt is shown below:

```
"mood": "Happy",
  "genre": "Pop",
  "required": "playlists" # Options: playlists, artists, tracks
}

# Define JSON structure for recommendations
json_schema = """
{
  "recommended_playlists": [
    {
      "name": "string",
      "description": "string",
      "tracks": ["string", "string"]
    }
  ],
  "recommended_tracks": ["string"],
  "recommended_artists": ["string"]
}
"""
```

The above code stores the user data that we shall collect from the API and the json schema of the output data. Below is the code that shall invoke the LLM to parse the data and give the desired output

```
# Define prompt template using LangChain
- Genre: {genre}
- Required Recommendation Type: {required}

Generate a structured JSON response matching this schema:
{json_schema}
""")

# Initialize GPT-4 LLM
llm = ChatOpenAI(model="gpt-4", temperature=0.7)

# Format input
input_data = {
    **user_data,
    "json_schema": json_schema
}

# Invoke LLM using the latest method
response = llm.invoke(prompt.format(**input_data))

# Convert response to JSON
try:
    recommendations = json.loads(response.content)
    print(json.dumps(recommendations, indent=2))
except json.JSONDecodeError:
    print("Invalid JSON response from LLM:", response.content)
```

By using the ChatPromptTemplate feature we are able to provide the user input data, their mood, behaviour data and other features that the LLM can easily use to understand user mood and then produce a response.

- **Explore playlist feature:**

The above ChatPromptTemplate can be used to provide a list of songs that we can fetch from Spotify and then create and return as a playlist.

First we need to get Track ID of the recommended tracks

```
Get Spotify track IDs
def get_spotify_track_ids(track_names, access_token):
    """Fetch track IDs by searching songs on Spotify."""
    headers = {"Authorization": f"Bearer {access_token}"}
    track_ids = []

    for track in track_names:
        search_url =
f"https://api.spotify.com/v1/search?q={track}&type=track&limit=1"
        response = requests.get(search_url, headers=headers)

        if response.status_code == 200:
            data = response.json()
            if data["tracks"]["items"]:
                track_ids.append(data["tracks"]["items"][0]["id"])

    return track_ids
```

Then we need to create a playlist from the user account.

```
# Create Playlist
def create_spotify_playlist(user_id, playlist_name, access_token):
    """Create a playlist in the user's Spotify account."""
    url = f"https://api.spotify.com/v1/users/{user_id}/playlists"
    headers = {"Authorization": f"Bearer {access_token}", "Content-
Type": "application/json"}
```

```

    payload = {"name": playlist_name, "description": "Generated by AI",
               "public": False}

    response = requests.post(url, headers=headers, json=payload)

    if response.status_code == 201:
        return response.json()["id"]

```

Then we add the tracks to the playlist from the user account

```

#Add Songs to Playlist
def add_tracks_to_playlist(playlist_id, track_ids, access_token):
    """Add tracks to a created playlist."""
    url = f"https://api.spotify.com/v1/playlists/{playlist_id}/tracks"
    headers = {"Authorization": f"Bearer {access_token}", "Content-Type": "application/json"}

    track_uris = [f"spotify:track:{track_id}" for track_id in track_ids]

    payload = {"uris": track_uris}

    response = requests.post(url, headers=headers, json=payload)
    return response.status_code == 201

```

- **Emotional Classification**

If the user provides some music as a input in some cases the LLM may not be able to predict the emotion or mood accurately. In those cases we can use some MER Models (Music Emotion Recognition) to accurately predict the emotion and provide better results.

One such transformer available on Hugging Face is facebook/wav2vec2-large-xlsr-53

```

MODEL_NAME = "superb/wav2vec2-base-superb-emo"
processor = Wav2Vec2Processor.from_pretrained(MODEL_NAME)
model = Wav2Vec2ForSequenceClassification.from_pretrained(MODEL_NAME)

```



```

# Function to preprocess audio
def preprocess_audio(audio_path):
    waveform, sample_rate = librosa.load(audio_path, sr=16000)

    input_values = processor(waveform, return_tensors="pt",
sampling_rate=16000).input_values

    return input_values

# Predict emotion
def detect_emotion(audio_path):
    inputs = preprocess_audio(audio_path)

    with torch.no_grad():
        logits = model(inputs).logits
        predicted_class = torch.argmax(logits, dim=-1).item()

    emotions = ["neutral", "happy", "sad", "angry", "fearful",
"disgusted", "surprised"]

    return emotions[predicted_class]

# Example usage
song_path = "your_song.mp3" # Provide song path
emotion = detect_emotion(song_path)
print(f"Detected Emotion: {emotion}")

```

After getting the output emotion or genre result from the MER Model we can provide the inference of the MER Model to the LLM Model so that it can provide more refined result.

The above implementation section highlights how I plan to implement the various parts of the project bringing the entirety of the project to its final and complete form. These layout the plan and the path of how the project will be developed step by step. Each snippet represents a crucial component, ensuring seamless integration of features such as user authentication, playlist generation, and personalized recommendations.

14. Process Diagram:

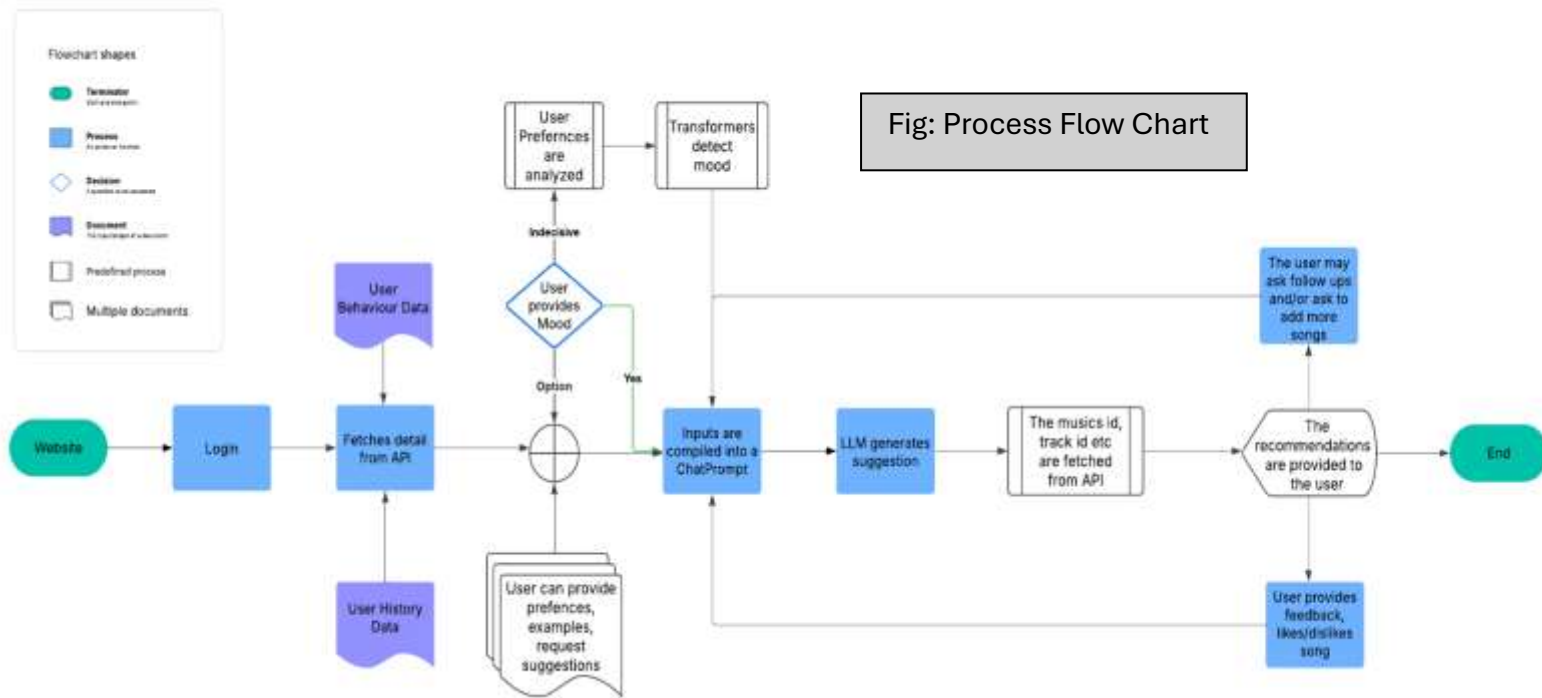


Fig: Process Flow Chart



Fig: Basic Architecture Diagram

15. Building Chat Like Experience/App

- **Backend Implementation :**
 - The **LangChain** will itself facilitate the use of LLM Model interconnecting the transformer model and the LLM.
 - The **transformer** will be provided as a **tool** to the **LLM Model** that it will access each time a call is made to the AI.
 - For transformer model a few examples that we can use are **facebook/wav2vec2-large-xlsr-53, superb/wav2vec2-base-superb-emo, audeering/wav2vec2-large-robust-12-ft-emotion-msp.**
 - For real-time chat we can build a **websocket** using FastAPI's WebSocket feature that will help us build real time connections.
 - To implement a good feedback system where the user may request for variations the websocket will play a very important role.
 - preserved as well.

Implementation of the FastAPI socket is as follows:

```
# WebSocket Connection Manager
class ConnectionManager:
    def __init__(self):
        self.active_connections: Dict[str, WebSocket] = {}

    async def connect(self, websocket: WebSocket, session_id: str):
        await websocket.accept()
        self.active_connections[session_id] = websocket
        logger.info(f"WebSocket connected: Session {session_id}")

    def disconnect(self, session_id: str):
        if session_id in self.active_connections:
            del self.active_connections[session_id]
            logger.info(f"WebSocket disconnected: Session {session_id}")
```

```

async def send_message(self, session_id: str, message: dict):
    if session_id in self.active_connections:
        await
self.active_connections[session_id].send_text(json.dumps(message))
    else:
        logger.warning(f"Failed to send message, no active session:
{session_id}")

manager = ConnectionManager()

```

```

#Main Connection the LLM
@app.post("/api/upload")

async def upload_image(file: UploadFile = File(...), symptoms:
Optional[str] = Form(None)):

    """Uploads an image and returns a file path for WebSocket
communication."""

    if not file.content_type.startswith("image/"):
        raise HTTPException(status_code=400, detail="Uploaded file must
be an image")

    try:
        file_ext = os.path.splitext(file.filename)[1]
        unique_filename = f"{uuid.uuid4()}{file_ext}"
        file_path = os.path.join("uploads", unique_filename)

        with open(file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        return {"success": True, "file_path": file_path, "symptoms":
symptoms}

    except Exception as e:
        logger.error(f"Error saving file: {e}")
        raise HTTPException(status_code=500, detail=f"Error saving
file: {str(e)}")

```

- **Frontend Implementation:**

- A simple UI similar to AI Chat applications like ChatGPT, Claude etc.
- Using JS, tailwind CSS, React this can be achieved easily.
- Socket.io can be used for establishing connection to the backend using WebSocket.

```
import { useEffect, useState } from "react";
import { io } from "socket.io-client";

const socket = io("http://localhost:8000");

const SocketComponent = () => {
  const [messages, setMessages] = useState([]);

  useEffect(() => {
    socket.on("connect", () => {
      console.log("Connected to WebSocket Server");
    });

    socket.on("message", (data) => {
      console.log("Received:", data);
      setMessages((prevMessages) => [...prevMessages, data]);
    });

    return () => {
      socket.disconnect();
    };
  }, []);

  const sendMessage = () => {
    socket.emit("chat", { message: "Hello from React!" });
  };
};
```

16. Feedback System Implementation (A challenge)

A good software system always requires a good feedback system as well.

- The user may or may not like the suggestions provided by the LLM/AI.
- The user must have a pathway so that they can provide their input.

For Example:

The user may say “I want to listen to some song like Blinding Light by The Weekend”.

For a reference these are the songs recommended by the ChatGPT.

Songs by The Weeknd (Similar Vibe)

- Save Your Tears – The Weeknd
- In Your Eyes – The Weeknd
- Out of Time – The Weeknd
- Take My Breath – The Weeknd

Other Artists with a Similar '80s Synth-Pop Sound

- Dua Lipa – Physical
- Dua Lipa – Levitating
- The Chainsmokers & Coldplay – Something Just Like This
- Tame Impala – The Less I Know The Better
- M83 – Midnight City
- Zedd & Grey ft. Maren Morris – The Middle
- Troye Sivan – My My My!
- Years & Years – King
- Robyn – Dancing On My Own

But the user may say that they like few of them and may not the rest of the others. The user may even want more suggestions.

This requires continuous reiteration of the same process but with added modification where the Chat Prompt needs to be modified and added up with few of the songs the user has selected.

Within the AI Chat this can be easily done due to the implementation of the Chat Memory and the WebSocket due to which the LLM will need not require to do the task again from scratch rather it will build up on its existing suggestions and further analyse user behaviour and hence providing better feedback outputs.

More detailed implementation involves:

1. Establishing sessions that will allow the user to interact more easily with the LLM as if they are using simple applications like WhatsApp.
2. **Persistent Context Awareness :**
By maintaining session ids, the AI will be able to track user interactions across multiple messages.
3. **Efficient Query Handling:**
The query handling will be much more efficient this way.
4. **Dynamic Responses:**
The responses will be much more dynamic making the application much more lively and user interactive.
5. **Optimized Resource Usage:**
Since the AI don't require to compute again from scratch it optimises resource usage.
6. **Seamless Integration with Frontend:**
The WebSocket-based architecture enables real-time updates without requiring multiple API calls. This results in a smoother, near-instantaneous chat experience compared to traditional REST-based implementations.
7. **Scalability & Multi-User Support:**
WebSocket allows handling multiple concurrent chat sessions without excessive server load. By assigning unique session IDs, different users can interact with the AI independently, each having their own contextual memory.

17. Scalability and Feasibility:

- **Modular & Microservices Architecture:**

Action: Break the application into independent services.

Implementation:

- Use Docker + Kubernetes to containerize and manage services.
- Implement RESTful APIs with FastAPI, or GraphQL for optimized queries.

- **Optimization for Scaling:**

Action: Implement caching, indexing, and partitioning to handle large-scale data.

Implementation:

- Use Redis or Memcached for caching frequently accessed data.
- Implement sharding and replication in MongoDB or MySQL for scalability.

- **Load Balancing & Auto Scaling:**

Action: Distribute user traffic across multiple servers.

Implementation:

- Use NGINX or AWS Load Balancer for distributing traffic.
- Implement auto-scaling with Kubernetes or AWS Auto Scaling.

- **Asynchronous Processing & Message Queues:**

Action: Move heavy computations to background tasks.

Implementation:

- Use Redis or Kafka for handling async tasks.
- Offload AI-based recommendations or data-heavy tasks to background workers.

- **API Rate Limiting & Caching:**

Action: Reduce server load by caching and limiting excessive requests.

Implementation:

- Implement Redis caching to store recent API responses.
- Use FastAPI's rate limiter (slowapi package) to prevent API abuse.

18. Expectation from GSOC and GFOSS:

Through Google Summer of Code (GSoC) and GFOSS, I aim to gain hands-on experience in open-source development, collaborate with experienced mentors, and contribute to a real-world AI-driven project.

From GSoC, I anticipate structured mentorship, exposure to large-scale project management, and the chance to work on production-level software. This will help me understand the best practices of collaborative coding, version control, and scalable AI implementations. I expect to deepen my understanding of open-source AI applications and contribute to a project that merges my passion for music and technology. I look forward to leveraging this opportunity to grow as a developer and contribute to a project that enhances music discovery and personalization for users worldwide.

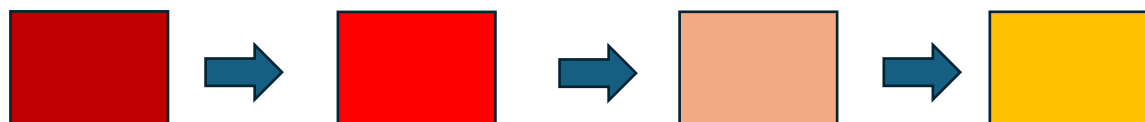
19. Timeline/Project Plan:

GSoC is round about 12 week duration starting from 9th May and the initial 2-3 weeks is provided for Community Bonding.

I plan to dedicate 60% of the time towards building the backend structure and developing the backend and 30% time towards developing the frontend, UI/UX of the software and remaining 10% of the time testing and fixing bugs.

Below is the timeline/project plan that I shall try to strictly follow so that the project is completed well within time and match the expectations of the mentor.

LEGEND Importance and time devoted and priority (highest -> lowest)



Phase	Start Date	End Date	Task	Priority
Community Bonding	9 th May	23 rd May		Yellow
	24 th May	25 th May	Gathering and Understanding Documentation	Light Orange

Phase 1	26 th May	23 rd June	Development of the Backend Server, Integrating the AI and the transformer	
Phase 2	24 th June	28 th June	Testing and fixing bugs of the backend server and ensuring proper functionality	
Phase 3	29 th June	1 st July	UI/UX Designing and Layout Designing	
	1 st July	25 th July	Frontend Development	
	26 th July	30 th July	Testing and debugging Frontend	
Phase 4	31 st July	7 th August	Frontend and Backend Integration	
	8 th August	10 th August	Final testing and debugging	
	11 th August	16 th August	Documentation and final PR	

I can assure you that if I get selected to work with GFOSS this summer, I will definitely dedicate the best of me to make this project successful and will love to continue working with GFOSS and their projects even after the summer.

Looking forward to working with you.

Thanks And Regards

Debanjan Rakshit