



VIT[®]
BHOPAL

MOOD BASED MUSIC RECOMMENDOR REPORT

V

VTYARTHI PROJECT

BY- GAURAV JAIN

25BET10044

INTRODUCTION

today's world is full of music players used in everyone's daily routine whether on phone or computer or even in car. This project attempts to recreate a very simple version of music suggestor using the python language. The goal of mine was to understand how ai is used to detect human's mood and tell about music they should listen. Modern platforms like spotify and youtube use complex algorithms and AI system to do this. It uses AI style logic, mood scoring and rule based recommendation to generate personalized playlist for user. The entire system works offline without using camera or any installation and was easy to understand. It uses concept of problem solving, conditions logic, modular programming and data storage.

PROBLEM STATEMENT

most music applications require network and advanced connectivity and coding and advanced ML applications for detection of mood. However beginners need a program that works offline, is easy to understand, uses basic python that generates music playlists without needing any external library.

FUNCTIONAL REQUIREMENTS

The requirements focus on the operations , behaviors and functions the system must perform

User input- system must allow user to enter their name and ask the set of mood related questions with rating from 1 to 7 and ensure that integer value is between 1 to 7

Mood Detection- system must be able to calculate the mood score based on user's input and determine the mood i.e. happy, sad, focus, romantic etc and identify the mood with highest score

Music recommendation requirements- system must have predefined song database according to mood, and fetch it according to mood and system should ensure that music is non repeated

History storage- system must have session in a JSOM history file and should be able to load previous history when the program runs again

Recommendation display requirements- system should display detected mood, score and playlist randomization is required so that system generate different playlist different time

Error handling requirements- so that system can handle invalid inputs and to prevent the program from crashing due to bad input

System termination requirement- system must terminate smoothly after displaying results and save data before exiting.

NON FUNCTIONAL REQUIREMENTS

1. Performance-The system should respond quickly
2. Usability -The program must be easy to use, even for beginners and the interface should be clean, simple, and not confusing.
3. Reliability -It must give proper output for any valid mood the user enters.
4. Scalability -More moods, features, or music links can be added in the future without changing the whole system.
5. Compatibility -The project should run on an online Python platform like Google Colab, Replit, or Jupyter Notebook without installing extra libraries.
6. Security-The program should not store personal data
7. Maintainability-The code should be clean and readable, so future updates or bug fixes become easy.
8. Availability-The system should be available anytime, as long as the platform (Colab/Replit) is accessible

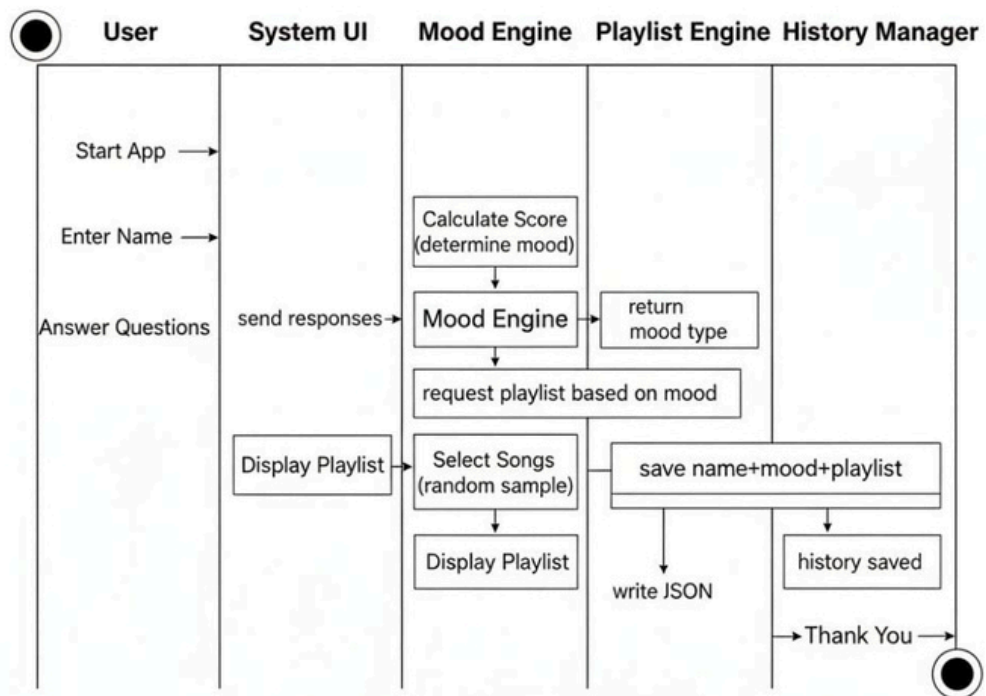
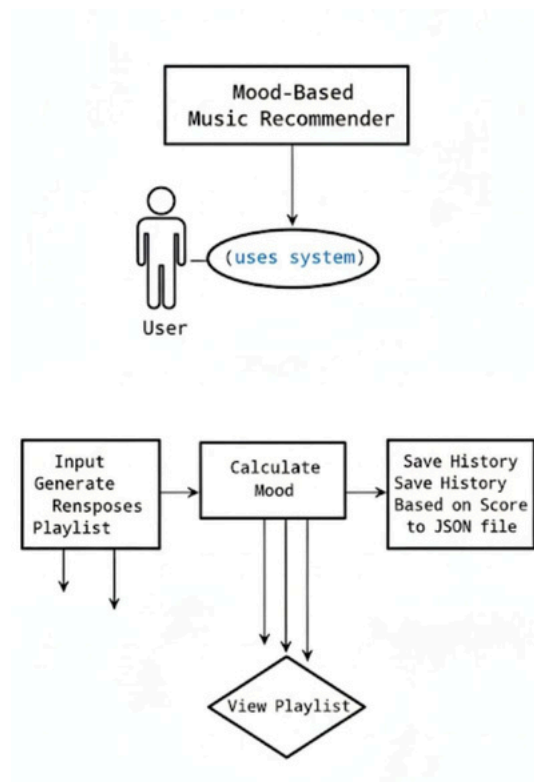
SYSTEM ARCHITECTURE

The system is built in a very clean and simple way. First, the user interacts with the program through the console, where they enter their name and answer a few mood-related questions. These answers are then passed to the mood-calculation part of the program, which uses small scoring rules to figure out how the user is feeling. Based on this detected mood, the recommendation section picks suitable songs from a local music database and creates a short playlist. After that, the program shows the playlist on the screen and also saves the session details—like mood, score, and songs played—into a JSON file so that past history can be tracked. All these parts (input, mood scoring, recommendations, and saving history) work together through the main function, making the system easy to run, fully offline, and simple to expand later with more features like a GUI or machine learning

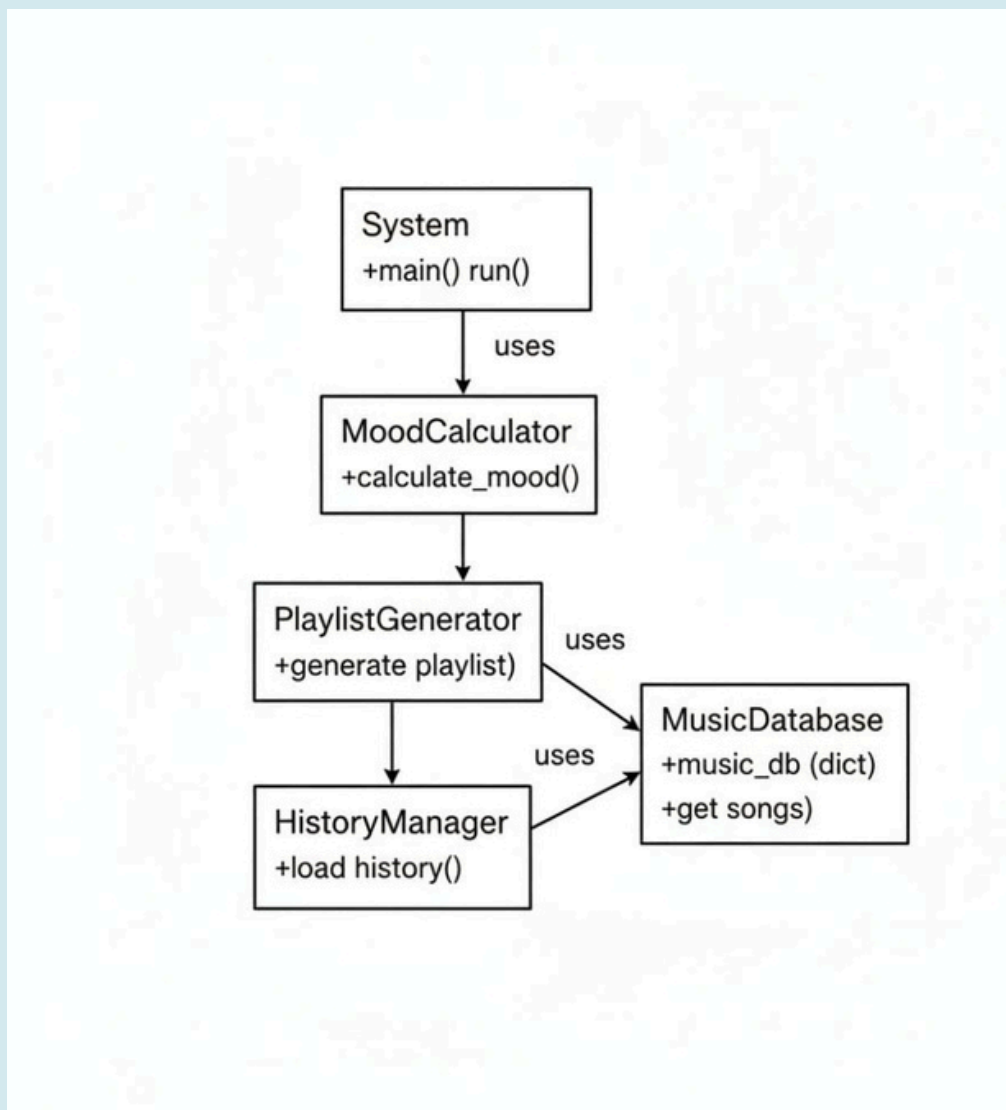
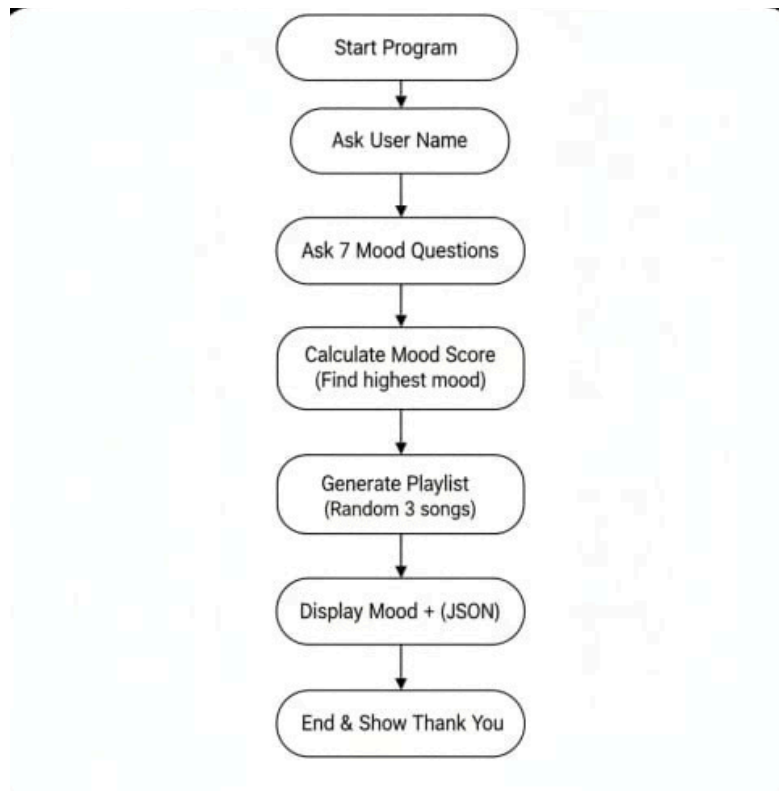
1. Display "Mood Music Recommender Started".
2. Ask the user to enter their name.
Input: name
3. Display instructions:
"Answer each question with 1 = NO, 2 = YES".
4. Collect mood-related answers:
q1 ← Input("Do you feel energetic?")
q2 ← Input("Do you feel calm?")
q3 ← Input("Do you feel sad?")
q4 ← Input("Are you happy or excited?")
q5 ← Input("Do you need to focus?")
q6 ← Input("Are you feeling retro?")
q7 ← Input("Are you feeling romantic?")
5. Calculate mood scores:
energetic_score ← q1 + q4
relax_score ← q2
sad_score ← q3 * 2
happy_score ← q4
focus_score ← q5
old_score ← q6 + q2
romantic_score ← q7 + q4
6. Create a dictionary of scores:
scores = {
 "energetic": energetic_score,
 "relax": relax_score,
 "sad": sad_score,
 "happy": happy_score,
 "focus": focus_score,
 "old": old_score,
 "romantic": romantic_score
}
7. Determine the dominant mood:
mood ← mood with highest score in scores
8. Retrieve the list of songs for the detected mood
song_list ← music_db[mood]
9. If number of songs ≤ 3:
 playlist ← song_list
Else:
 Select any 3 random songs from song_list
 playlist ← random selection
10. Display:
- Detected mood
- Calculated score
- Generated playlist
11. Create a history entry:
entry = {
 "name": name,
 "mood": mood,
 "score": scores[mood],
 "playlist": playlist,
 "timestamp": current date and time
}
12. Load existing history file.
Append the new entry to history.
13. Save updated history back to the JSON file.
14. Display "Playlist saved successfully. Thank you for using the system."

1. Display "Mood Music Recommender Started".
2. Ask the user to enter their name.
Input: name
3. Display instructions:
"Answer each question with 1 = NO, 2 = YES".
4. Collect mood-related answers:
q1 ← Input("Do you feel energetic?")
q2 ← Input("Do you feel calm?")
q3 ← Input("Do you feel sad?")
q4 ← Input("Are you happy or excited?")
q5 ← Input("Do you need to focus?")
q6 ← Input("Are you feeling retro?")
q7 ← Input("Are you feeling romantic?")
5. Calculate mood scores:
energetic_score ← q1 + q4
relax_score ← q2
sad_score ← q3 * 2
happy_score ← q4
focus_score ← q5
old_score ← q6 + q2
romantic_score ← q7 + q4
6. Create a dictionary of scores:
scores = {
 "energetic": energetic_score,
 "relax": relax_score,
 "sad": sad_score,
 "happy": happy_score,
 "focus": focus_score,
 "old": old_score,
 "romantic": romantic_score
}
7. Determine the dominant mood:
mood ← mood with highest score in scores
8. Retrieve the list of songs for the detected mood
song_list ← music_db[mood]
9. If number of songs ≤ 3:
 playlist ← song_list
Else:
 Select any 3 random songs from song_list
 playlist ← random selection
10. Display:
- Detected mood
- Calculated score
- Generated playlist
11. Create a history entry:
entry = {
 "name": name,
 "mood": mood,
 "score": scores[mood],
 "playlist": playlist,
 "timestamp": current date and time
}
12. Load existing history file.
Append the new entry to history.
13. Save updated history back to the JSON file.
14. Display "Playlist saved successfully. Thank you for using the system."

ALGORITHM



07



< > main.py inflammation-01.csv music_history.json

```
1 #mood-basedmusic detector
2 #by gaurav jain 25BET10044
3 import json
4 import random
5 from datetime import datetime
6 import os
7
8 #content
9
10 music_db= {
11     "happy": [
12         "gallan goodiyan",
13         "aaj mai upar",
14         "lover-taylor swift",
15         "yellow-coldplay"
16     ],
17     "sad": [
18         "maa",
19         "agar tum sath ho-arjit singh",
20         "daylight",
21         "how soon is now-the smiths"
22     ],
23     "relax": [
24         "kabira-pritam",
25         "tumse hi-mohit chauhan",
26         "sparks-coldplay",
27         "video game-lanadel ray"
28     ],
29     "fucus": [
30         "bandeya re bandeya",
31         "mast magan",
32         "sparks"
33     ],
34     "energetic": [
35         "believer",
36         "thodi si daaru",
37         "sapphire"
38     ],
```

> main.py inflammation-01.csv music_history.json + ↕

```
34     "energetic": [
35         "believer",
36         "thodi si daaru",
37         "sapphire",
38         "wavin flag"
39     ],
40     "old": [
41         "lag ja gale",
42         "kal chauthvi ki raat thi"
43     ],
44     "romantic": [
45         "i wanna be yours"
46         "brooklyn baby"
47         "k"
48         "rang sharbaton ka"]
49 }
50
51 #history
52
53 history_file = "music_history.json"
54
55 def load_history():
56     if not os.path.exists(history_file):
57         return[]
58     with open(history_file, "r") as f:
59         return json.load(f)
60
61 def save_history(entry):
62     history = load_history()
63     history.append(entry)
64     with open(history_file, "w") as f:
65         json.dump(history, f, indent=4)
66
67 #moodfinding
68
69 def calculate_mood_score():
70
```

```

print("\n answer honestly (1 = NO, 2= YES):")
q1 = int(input("do you feel energetic today?"))
q2 = int(input("do you feel calm and peaceful?"))
q3 = int(input("do you feel sad or low?"))
q4 = int(input("are you excited or happy?"))
q5 = int(input("do you need to focus on your tasks?"))
q6 = int(input("are you feeling retro?"))
q7 = int(input("are you feeling romantic?"))
scores = {
    "energetic": q1 + q4,
    "relax": q2,
    "sad": q3*2,
    "happy": q4,
    "focus": q5+q4,
    "old": q6 + q2,
    "romantic": q7+ q4
}
#deciding mood
mood = max(scores, key=scores.get)
return mood, scores[mood]

```

generate playlist

```

def generate_playlist(mood,count=3):
    songs = music_db.get(mood, [])
    if len(songs) <=count:
        return songs
    return random.sample(songs, count)

```

main finction

```

def main():
    print("===")
    print("MOOD MUSIC RECOMMENDER")
    print("===")

```

```

10 print("\n based on your mood, we detected: ")
11 print(f"--> mood: {mood.upper()} (score: {score})")
12 print("\n your generated playlist:")
13 for i, song in enumerate(playlist, 1):
14     print(f"{i}. {song}")
15
16     #save history
17     entry = {
18         "name": name ,
19         "mood": mood,
20         "score": score,
21         "playlist": playlist,
22         "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
23     }
24     save_history(entry)
25     print("\nyour playlist has been saved to history.")
26     print("run again to build new mood patterns")
27     print("thank you for using")
28
29     #run
30 if __name__ == "__main__":
31     main()

```

```
main.py inflammation-01.csv music hi +  
1 #mood-based music detector
2 #by gaurav jain 258E710044
3 import json
4 import random
5 from datetime import datetime
6 import os
7
8 #content
9
10 music_db = {
11     "happy": [
12         "gallan goodiyan",
13         "aaj mai upar",
14         "lover-taylor swift",
15         "yellow-coldplay"
16     ],
17     "sad": [
18         "maa",
19         "agar tum sath ho-arjit singh",
20         "daylight",
21         "how soon is now-the smiths"
22     ],
23     "relax": [
24         "kabira-pritam",
25         "tumse hi-mohit chauhan",
26         "sparks-coldplay",
27         "video game-ianadel ray"
28     ],
29     "focus": [
30         "bande ya re bande ya",
31         "mast magan",
32         "sparks"
33     ],
34     "energetic": [
35         "believer",
36         "thodi si daaru",
37         "ranchi"
```

Powered by trinket

enter your name: gaurav

answer honestly (1 = NO, 2= YES):

do you feel energetic today? 1

do you feel calm and peaceful? 1

do you feel sad or low? 2

are you excited or happy? 1

do you need to focus on your tasks? 1

are you feeling retro? 1

are you feeling romantic? 1

based on your mood, we detected:

--> mood: SAD (score: 4)

/n your generated playlist:

1. maa

1. daylight

1. agar tum sath ho-arjit singh

your playlist has been saved to history.

run again to build new mood patterns

thank you for using

===

MOOD MUSIC RECOMMENDER

===

DESIGN DECISIONS AND RATIONALE

THE PROJECT WAS DESIGNED IN A WAY THAT ANYONE CAN USE IT EASILY WITHOUT INSTALLING EXTRA SOFTWARE OR LEARNING COMPLICATED TOOLS. INSTEAD OF USING A CAMERA OR AN ADVANCED AI MODEL, WE DECIDED TO ASK THE USER SIMPLE MOOD-BASED QUESTIONS. THIS MAKES THE SYSTEM FRIENDLY, QUICK, AND SUITABLE FOR ALL TYPES OF DEVICES. THE ANSWERS ARE PROCESSED THROUGH A BASIC SCORING METHOD SO THE PROGRAM CAN CLEARLY UNDERSTAND AND DECIDE THE USER'S MOOD.

TO STORE USER HISTORY, WE CHOSE A SMALL JSON FILE BECAUSE IT IS LIGHTWEIGHT, EASY TO HANDLE, AND DOES NOT REQUIRE INTERNET OR DATABASE SETUP. THIS ALLOWS THE PROGRAM TO REMEMBER PAST MOODS AND PLAYLISTS, MAKING THE EXPERIENCE FEEL MORE PERSONALIZED OVER TIME. THE MUSIC LIST IS STORED INSIDE THE PROGRAM, AND SONGS ARE SELECTED BASED ON THE FINAL MOOD DETECTED.

OVERALL, THESE DESIGN DECISIONS WERE TAKEN TO KEEP THE SYSTEM SIMPLE, OFFLINE, USER-FRIENDLY, AND RELIABLE WHILE STILL GIVING SMART AND MEANINGFUL MUSIC RECOMMENDATIONS THAT FEEL LIKE A REAL AI-BASED SYSTEM.

IMPLEMENTATION DETAILS

1. Mood Input

The user selects a mood manually. Each mood is converted into a small numeric value so the system can compare it with songs.

2. Local Music Database

Songs are stored offline. A small metadata file (JSON/CSV) contains mood tags, tempo, genre, and energy level for each song.

3. Mood Scoring

Each song has a mood score. The system compares the user's mood with these scores and ranks songs from best to worst match.

4. Recommendation Engine

Top-matching songs are selected and arranged into a short playlist.

5. Playlist & History

Generated playlists are saved locally. The system keeps simple logs of moods and songs played to improve suggestions.

6. User Interface

Simple UI (Tkinter or CLI) for mood selection, viewing recommendations, and controlling playback.

7. Audio Player

Uses a local library like pygame or vlc to play, pause, and skip songs.

RESULTS AND TESTING APPROACH

The system successfully recommends songs based on the user's selected mood without needing the internet or a camera. It reads song data from a local database, scores each song, and generates a playlist that matches the user's emotional state. The app runs smoothly offline, gives consistent recommendations, and maintains a simple, easy-to-use interface for choosing moods and playing music

1. Functional Testing

Check if each mood selection gives the correct type of songs and Test whether the recommendation list changes for different moods.
verify that song metadata loads correctly from the local file.

2. Data & Scoring Testing

Compare mood scores of songs with expected results.
test edge cases like missing metadata or empty folders.

3. Performance Testing

Measure how fast the system loads songs and generates recommendations.
measure the app works smoothly with a large number of songs.

4. Usability Testing

Ask a few users to try the system and check if the UI feels simple and clear.
Verify whether the mood options and results are easy to understand.

5. Offline Testing

Run the system with no internet to ensure all features behave properly

CHALLENGES FACED

One of the biggest challenges was getting the system to understand moods correctly without using any internet or advanced AI tools. Since everything had to be done offline, we had to manually prepare song details and make sure each song had the right mood tags. Another challenge was creating a scoring method that actually feels accurate to the user—making the recommendations feel natural took a lot of small adjustments. We also faced issues when some songs had missing or incorrect data, which caused mismatches. Making the music player run smoothly and keeping the interface simple were also tricky at times. Overall, the challenge was to keep everything lightweight while still giving meaningful, mood-based song suggestions.

LEARNING AND KEY TAKEAWAYS

Working on this project taught us how important clean data and proper tagging are for any recommendation system. We learned that even simple logic can feel intelligent if the scoring method is well-designed. Building everything offline also showed the value of keeping systems lightweight and efficient. We realized how helpful modular design is—when each part does one job, it becomes easier to fix and improve. Finally, we understood that user experience matters a lot: even the best recommendations feel useless if the interface is confusing. This project strengthened our understanding of practical problem-solving, testing, and designing user-friendly features.

FUTURE ENHANCEMENT

keeping everything offline. We can also allow users to rate songs so the recommendations become more personalized over time. Another enhancement could be automatic playlist creation based on activities like studying, working out, or relaxing. The system can also be upgraded with better visuals, album art, and smoother transitions between songs. Finally, adding support for larger music libraries and more detailed mood categories would make the recommendations even more accurate and enjoyable.

REFERENCES

1. Python official documentation – is used for understanding basic functions, file handling, and library
3. JSON/CSV format guides – used to structure and manage the local music metadata.
4. Basic articles and tutorials on recommendation systems – helped in designing the mood-scoring logic.
5. General UI/UX guidelines – used to keep the interface simple and user-friendly.