UID EXPERIMENT: 3

CLI (Command Line Interface):

Source Code:

```
import os

PLAYLIST_FILE = "playlist.txt"

def load_playlist():
    if not os.path.exists(PLAYLIST_FILE):
        return []
    with open(PLAYLIST_FILE, "r") as file:
        return [line.strip() for line in file.readlines()]

def save_playlist(playlist):
    with open(PLAYLIST_FILE, "w") as file:
        for song in playlist:
            file.write(song + "\n")

def view_playlist():
    playlist = load_playlist()
    if not playlist:
        print("\n🎵 Playlist is empty.\n")
    else:
        print("\n🎵 Your Playlist:")
        for i, song in enumerate(playlist, start=1):
            print(f"{i}. {song}")
        print()

def add_song():
    song = input("Enter song name to add: ")
    playlist = load_playlist()
    playlist.append(song)
    save_playlist(playlist)
    print("✅ Song added!\n")
```

```python
def remove_song():
    playlist = load_playlist()
    view_playlist()
    if not playlist:
        return
    try:
        index = int(input("Enter song number to remove: "))
        removed = playlist.pop(index - 1)
        save_playlist(playlist)
        print(f"❌ Removed: {removed}\n")
    except (ValueError, IndexError):
        print("⚠ Invalid selection.\n")

def main():
    while True:
        print("==== CLI Playlist Manager ====")
        print("1. View Playlist")
        print("2. Add Song")
        print("3. Remove Song")
        print("4. Exit")

        choice = input("Choose an option: ")

        if choice == "1":
            view_playlist()
        elif choice == "2":
            add_song()
        elif choice == "3":
            remove_song()
        elif choice == "4":
            print("Goodbye 👋")
            break
        else:
            print("⚠ Invalid choice. Try again.\n")

if __name__ == "__main__":
    main()
```

How It Works – CLI Playlist Manager

This program is a simple Command-Line Playlist Manager that allows users to view, add, and remove songs. The playlist is stored in a file called playlist.txt, so the data is saved even after the program closes.

🔹 File Handling

- The program checks if playlist.txt exists.
- Songs are stored line-by-line in the file.
- When changes are made, the file is updated automatically.

🔹 Main Functions

- load_playlist()
  Reads songs from the file and returns them as a list.
- save_playlist()
  Saves the updated playlist back into the file.
- view_playlist()
  Displays all songs with numbering.
  Shows a message if the playlist is empty.
- add_song()
  Takes song input from the user and adds it to the playlist.
- remove_song()
  Removes a selected song using its number.
  Handles invalid inputs safely.

How to Run (CLI)

Open terminal in the file location and run:

python playlist_manager.py

GUI(Graphical User Interface):

Source Code:

```
import customtkinter as ctk
import random
import math

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("dark-blue")

app = ctk.CTk()
app.geometry("1200x750")
app.title("BOLT MUSIC")

# ---------------- COLORS ---------------- #
BG = "#0a0a0f"
SIDEBAR = "#11111a"
NEON_BLUE = "#00f2ff"
NEON_PINK = "#ff00c8"
NEON_PURPLE = "#9d00ff"

app.configure(fg_color=BG)

# ---------------- SIDEBAR ---------------- #
sidebar = ctk.CTkFrame(app, width=220, fg_color=SIDEBAR)
sidebar.pack(side="left", fill="y")

logo = ctk.CTkLabel(sidebar,
                    text="⚡ BOLT MUSIC",
                    font=ctk.CTkFont(size=22, weight="bold"),
                    text_color=NEON_PINK)
logo.pack(pady=30)

menu = ["Dashboard", "Discover", "Favourites", "Playlists"]

for item in menu:
```

```
    btn = ctk.CTkButton(
        sidebar,
        text=item,
        fg_color="transparent",
        hover_color="#1b1b2a",
        text_color=NEON_BLUE,
        anchor="w"
    )
    btn.pack(fill="x", padx=20, pady=6)

# ----------------- MAIN AREA ----------------- #
main = ctk.CTkFrame(app, fg_color=BG)
main.pack(side="right", expand=True, fill="both", padx=30, pady=20)

title = ctk.CTkLabel(main,
                     text="YOUR MIXES",
                     font=ctk.CTkFont(size=30, weight="bold"),
                     text_color=NEON_BLUE)
title.pack(anchor="w", pady=10)

# ----------------- ANIMATED CARDS ----------------- #
card_frame = ctk.CTkFrame(main, fg_color="transparent")
card_frame.pack(pady=30)

mixes = ["Favourites Mix", "Friends Mix", "Chill Mix", "New Music Mix"]
neon_colors = [NEON_BLUE, NEON_PINK, NEON_PURPLE, "#00ff99"]

cards = []

def create_card(parent, text, color):
    card = ctk.CTkFrame(parent,
                        width=220,
                        height=220,
                        corner_radius=25,
                        fg_color="#141422")
    card.pack_propagate(False)

    label = ctk.CTkLabel(card,
                         text=text,
                         font=ctk.CTkFont(size=16, weight="bold"),
```

```
                        text_color=color)
    label.pack(expand=True)

    # Hover animation
    def on_enter(e):
        card.configure(fg_color="#1c1c30")

    def on_leave(e):
        card.configure(fg_color="#141422")

    card.bind("<Enter>", on_enter)
    card.bind("<Leave>", on_leave)

    return card

for i in range(4):
    card = create_card(card_frame, mixes[i], neon_colors[i])
    card.grid(row=0, column=i, padx=20)
    cards.append(card)

# ---------------- PULSE ANIMATION ---------------- #
pulse_value = 0

def pulse():
    global pulse_value
    pulse_value += 0.1
    glow = int((math.sin(pulse_value) + 1) * 127)
    color = f"#{glow:02x}00ff"
    title.configure(text_color=color)
    app.after(50, pulse)

pulse()

# ---------------- PLAYLIST AREA ---------------- #
playlist_label = ctk.CTkLabel(main,
                              text="YOUR TRACKS",
                              font=ctk.CTkFont(size=20, weight="bold"),
                              text_color=NEON_PINK)
playlist_label.pack(anchor="w", pady=20)
```

```
playlist_box = ctk.CTkTextbox(main,
                              height=180,
                              corner_radius=20,
                              fg_color="#101018",
                              text_color=NEON_BLUE)
playlist_box.pack(fill="x")

# ----------------- ADD SONG ----------------- #
entry_frame = ctk.CTkFrame(main, fg_color="transparent")
entry_frame.pack(fill="x", pady=15)

song_entry = ctk.CTkEntry(entry_frame,
                          placeholder_text="Add a track into the
system...",
                          height=40,
                          corner_radius=25,
                          fg_color="#151525",
                          text_color="white")
song_entry.pack(side="left", expand=True, fill="x", padx=5)

def add_song():
    song = song_entry.get()
    if song.strip():
        playlist_box.insert("end", f"⚡  {song}\n")
        song_entry.delete(0, "end")

add_btn = ctk.CTkButton(entry_frame,
                        text="UPLOAD",
                        width=140,
                        corner_radius=25,
                        fg_color=NEON_PURPLE,
                        hover_color=NEON_PINK,
                        command=add_song)
add_btn.pack(side="right", padx=5)

# ----------------- MINI PLAYER ----------------- #
player = ctk.CTkFrame(app,
                      height=80,
                      fg_color="#0d0d14",
                      corner_radius=0)
```

```
player.pack(side="bottom", fill="x")

now_playing = ctk.CTkLabel(player,
                           text="SYSTEM IDLE",
                           text_color=NEON_BLUE)
now_playing.pack(side="left", padx=30)

controls = ctk.CTkFrame(player, fg_color="transparent")
controls.pack(side="right", padx=30)

for icon in ["⏮", "⏯", "⏭"]:
    btn = ctk.CTkButton(controls,
                        text=icon,
                        width=45,
                        corner_radius=30,
                        fg_color=NEON_BLUE,
                        hover_color=NEON_PINK)
    btn.pack(side="left", padx=8)

app.mainloop()
```

How It Works:

This program is a modern **Graphical User Interface (GUI) Music Dashboard** built using **CustomTkinter**. It creates a sleek, dark-themed music app interface with animations and interactive components.

🔹 1. App Setup

- Uses customtkinter for a modern UI.
- Dark mode and neon color theme are enabled.
- Main window is created with custom size and background color.

🔹 2. Sidebar Navigation

- A fixed sidebar contains:
    - App logo (⚡ BOLT MUSIC)
    - Navigation buttons (Dashboard, Discover, Favourites, Playlists)
- Buttons have hover effects for smooth interaction.

🔹 3. Main Content Area

- Displays **“YOUR MIXES”** section.
- Four animated mix cards are created dynamically using a function.
- Cards change color when hovered (interactive animation effect).

🔹 4. Pulse Animation

- Uses math.sin() to create a glowing neon pulse effect.
- The title text color continuously changes.
- app.after() runs the animation loop smoothly.

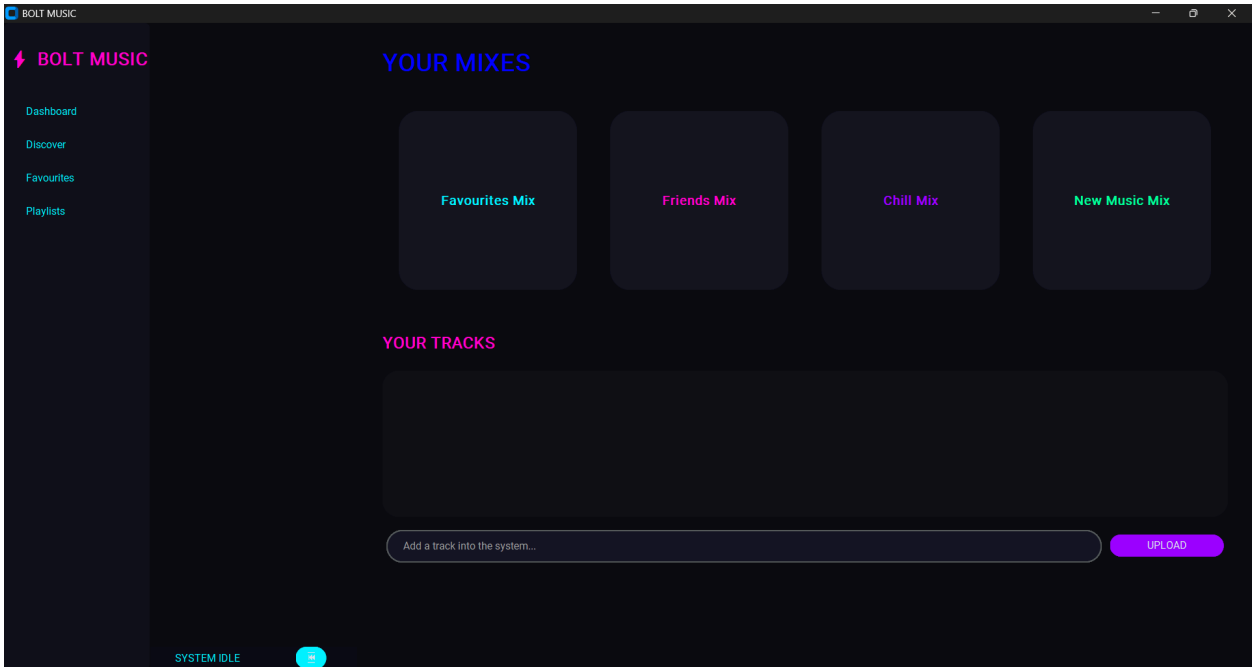
🔹 5. Playlist Section

- “YOUR TRACKS” area displays songs inside a textbox.
- User can type a song name into the input field.
- Clicking **UPLOAD**:
    - Adds the song to the playlist box.

- Clears the input field.

🔹 6. Mini Player

- Bottom bar simulates a music player.
- Shows current status (“SYSTEM IDLE”).
- Includes control buttons (Previous, Play/Pause, Next).



VUI(Voice User Interface):

Source Code:

```
import speech_recognition as sr
import pyttsx3
import sounddevice as sd
import numpy as np
import os

PLAYLIST_FILE = "playlist.txt"

engine = pyttsx3.init()
recognizer = sr.Recognizer()

def speak(text):
    print("Assistant:", text)
    engine.say(text)
    engine.runAndWait()

def load_playlist():
    if os.path.exists(PLAYLIST_FILE):
        with open(PLAYLIST_FILE, "r") as f:
            return [line.strip() for line in f.readlines()]
    return []

def save_playlist(playlist):
    with open(PLAYLIST_FILE, "w") as f:
        for song in playlist:
            f.write(song + "\n")

def record_audio(duration=5, samplerate=16000):
    speak("Listening")
    recording = sd.rec(int(duration * samplerate),
                       samplerate=samplerate,
                       channels=1,
                       dtype='int16')
```

```
    sd.wait()
    return recording

def listen():
    try:
        audio_data = record_audio()

        audio = sr.AudioData(
            audio_data.tobytes(),
            16000,
            2
        )

        command = recognizer.recognize_google(audio)
        print("You said:", command)
        return command.lower()

    except Exception as e:
        speak("I didn't catch that")
        return ""

def add_song(song):
    playlist = load_playlist()
    playlist.append(song)
    save_playlist(playlist)
    speak(f"{song} added")

def remove_song(song):
    playlist = load_playlist()
    if song in playlist:
        playlist.remove(song)
        save_playlist(playlist)
        speak(f"{song} removed")
    else:
        speak("Song not found")

def view_playlist():
    playlist = load_playlist()
    if playlist:
        speak("Your playlist contains")
```

```
        for song in playlist:
            speak(song)
    else:
        speak("Playlist is empty")

# ---------------- MAIN LOOP ---------------- #

speak("Voice playlist activated")

while True:
    command = listen()

    if "add" in command:
        song = command.replace("add", "").strip()
        if song:
            add_song(song)

    elif "remove" in command:
        song = command.replace("remove", "").strip()
        if song:
            remove_song(song)

    elif "show" in command or "view" in command:
        view_playlist()

    elif "exit" in command or "stop" in command:
        speak("Shutting down")
        break
```

How It Works – Voice Playlist Manager (VUI)

This program is a Voice User Interface (VUI) Playlist Manager that allows users to control their playlist using voice commands instead of typing. It uses speech recognition for input and text-to-speech for responses.

🔹 1. Libraries Used

- speech_recognition → Converts voice into text.
- pyttsx3 → Converts text into speech (assistant voice).
- sounddevice → Records audio from the microphone.
- numpy → Handles audio data format.
- os → Manages playlist file storage.

🔹 2. Voice Output (Text-to-Speech)

The speak(text) function:

- Prints the assistant message.
- Uses pyttsx3 to speak the text aloud.
- Allows the system to respond verbally.

🔹 3. Playlist Storage

- Songs are stored in playlist.txt.
- load_playlist() reads songs from the file.
- save_playlist() updates the file after changes.
- Ensures data is saved permanently.

🔹 4. Voice Input Process

Step 1: Record Audio

record_audio():

- Records 5 seconds of microphone input.
- Uses sounddevice to capture audio.
- Returns raw audio data.

Step 2: Convert Speech to Text

listen():

- Converts recorded audio into AudioData.
- Uses Google Speech Recognition API.
- Returns recognized text in lowercase.
- If recognition fails, assistant says:

  “I didn’t catch that”

🔹 5. Voice Commands Supported

Inside the main loop, the system listens continuously and checks for keywords:

- "add" → Adds a song
  Example:

  “Add Shape of You”

- "remove" → Removes a song
  Example:

  “Remove Believer”

- "show" / "view" → Reads all songs in playlist
- "exit" / "stop" → Shuts down the assistant

🔹 6. Program Flow

1. Assistant says:

   “Voice playlist activated”

2. System listens for command.
3. Recognizes speech.

4. Matches keywords.
5. Performs action (add/remove/view).
6. Speaks response.
7. Repeats until user says exit.