AI-Mental-Healthcare

Alhaan

2025-05-06 19:31:43

Contents

Project Overview

No project description provided.

Codebase Statistics

- **Total files:** 28
- **Total lines of code:** 3366
- **Languages used:** plaintext (4), python (9), css (3), javascript (4), html (8)
- **Generated by:** code2pdf v0.2

Directory Structure

```
Simplified_Agentic_AI-Mental-Heatlh-checker/
    └── .gitignore
    └── docker-compose.yml
    └── file_structure_and_flow.txt
    └── run.py
    ├── backend/
        └── ai_chat.py
        └── database.py
        └── Dockerfile
        └── main.py
        └── models.py
        └── requirements.txt
        └── sentiment.py
        └── transcription.py
        ├── audio_logic/
            ├── sample_audio/
        ├── chatbot_test/
            └── chat_test.py
    ├── ml_data/
        └── ml.py
    ├── static/
        └── ai_chat.css
        └── audio_room.js
        └── meeting-room.js
        └── meeting_room.css
        └── scripts.js
        └── styles.css
        └── voice_chatbot.js
        ├── audio/
        ├── images/
        ├── video/
    ├── templates/
        └── ai_chat.html
        └── audio_room.html
        └── index.html
        └── login.html
        └── meeting_room.html
        └── signup.html
        └── therapy.html
        └── voice_chatbot.html
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/.gitignore

```
__pycache__/
backend/__pycache__/

venv/
venv/bin/
venv/Include/

.env
.vscode

audio_logic

audio/
images/
video/
mindsentry.db
__init__.py
Readme.md
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/docker-compose.yml

```
version: "3.9"
```

```
services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ./backend:/app
    working_dir: /app
    command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
    restart: always
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/file_structure_and_flow.txt

```
File Structure and Flow of the Application:

1. Backend:
   - main.py:
     * Defines API endpoints and WebSocket functionality.
     * Handles sentiment analysis, audio transcription, and chat room management.
     * Flow:
       - Sentiment Analysis: sentiment.py -> main.py
       - Audio Transcription: transcription.py -> main.py
       - WebSocket Chat: main.py (manages WebSocket connections)

   - transcription.py:
     * Handles audio transcription using the Whisper model.
     * Flow: Audio file -> transcription.py -> main.py

   - sentiment.py:
     * Performs sentiment analysis using Hugging Face Transformers.
     * Flow: Text input -> sentiment.py -> main.py

2. Frontend:
   - templates/index.html:
     * Main landing page for sentiment analysis and audio transcription.
     * Allows navigation to the therapy page and meeting rooms.

   - templates/therapy.html:
     * Provides a relaxation page with soothing sounds and calming videos.

   - templates/meeting_room.html:
     * UI for the meeting room with WebSocket chat functionality.

   - static/scripts.js:
     * Handles frontend logic for sentiment analysis, audio transcription, and
 ↪  WebSocket chat.

   - static/meeting-room.js:
     * Manages WebSocket communication for the meeting room.

   - static/room-navigation.js:
     * Handles navigation to the meeting room.

   - static/styles.css:
     * Provides styling for the frontend pages.

3. Entry Point:
   - run.py:
     * Starts the FastAPI application.

4. Additional Notes:
   - WebSocket communication is managed via the /ws/{room_id} endpoint in main.py.
   - Sentiment analysis and transcription results are displayed on the frontend.
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/run.py

```python
"""
File: run.py
Purpose: Entry point to start the FastAPI application.
"""
import uvicorn

if __name__ == "__main__":
    uvicorn.run("backend.main:app", host="127.0.0.1", port=8000, reload=True)
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/ai_chat.py

```python
import os
```

```python
from cerebras.cloud.sdk import Cerebras

client = Cerebras()

async def get_ai_response(message: str, history: list = None) -> dict:
    """
    Get response from Cerebras AI using the official SDK with context.
    """
    try:

        if history is None:
            history = []


        if not history:
            history.append({
                "role": "system",
                "content": "You are a compassionate AI assistant dedicated to
                 ↪  providing supportive and insightful mental health guidance.
                 ↪  Give the reply very breifly"
            })


        history.append({
            "role": "user",
            "content": message
        })


        chat_completion = client.chat.completions.create(
            messages=history,
            model="llama3.1-8b"
        )

        response = chat_completion.choices[0].message.content


        history.append({
            "role": "assistant",
            "content": response
        })

        return {"response": response, "history": history}
    except Exception as e:
        print(f"Error calling Cerebras API: {str(e)}")
        return {"response": "I apologize, I'm having trouble responding right
        ↪  now.", "history": history}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/database.py

```
"""
File: database.py
Purpose: Sets up the SQLite database connection and session management.
"""
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

DATABASE_URL = "sqlite:///mindsentry.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})  #
↪  SQLite concurrency fix
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/Dockerfile

```
FROM python:3.9-slim-bullseye

WORKDIR /app


RUN apt-get update && apt-get install -y \
    ffmpeg \
    git \
    build-essential \
    && apt-get clean && rm -rf /var/lib/apt/lists/*


COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt


COPY . .

EXPOSE 8000

CMD ["uvicorn", "backend.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/main.py

```
"""
File: main.py
Purpose: This is the main backend file for the FastAPI application. It defines API
 ↪  endpoints and WebSocket functionality.

Flow:
1. Sentiment Analysis:
   - Endpoint: /analyze
   - Flow: sentiment.py -> main.py

2. Audio Transcription:
   - Endpoint: /transcribe
   - Flow: transcription.py -> main.py

3. WebSocket for Chat Rooms:
   - Endpoint: /ws/{room_id}
   - Flow: main.py (manages WebSocket connections and room participants)

4. HTML Templates:
   - Endpoint: /
   - Endpoint: /therapy
   - Endpoint: /meeting-room
   - Flow: main.py -> templates/*.html
"""
```

```
from fastapi import FastAPI, UploadFile, File, WebSocket, WebSocketDisconnect,
 ↪  HTTPException, status, Form, Depends
from pydantic import BaseModel
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.templating import Jinja2Templates
from fastapi.requests import Request
from fastapi.staticfiles import StaticFiles
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session
from sqlalchemy import MetaData
from .sentiment import analyze_sentiment
from .transcription import transcribe_audio
from .database import engine, get_db
from .models import Base, User, Text
from passlib.context import CryptContext
import os
from typing import Dict, List
from random import choice
from .ai_chat import get_ai_response


templates = Jinja2Templates(directory="templates")

app = FastAPI()

app.mount("/static", StaticFiles(directory="static"), name="static")


app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

metadata = MetaData()
metadata.reflect(bind=engine)
metadata.drop_all(bind=engine)
Base.metadata.create_all(bind=engine)
```

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


def get_current_user(request: Request):
    user_id = request.cookies.get("user_id")
    if not user_id:
        return None
    return {"uid": int(user_id), "logged_in": True}

class TextInput(BaseModel):
    text: str

rooms = {}

@app.get("/", response_class=HTMLResponse)
def read_root(request: Request):
    user = get_current_user(request)
    if not user:
        return RedirectResponse(url="/login")
    return templates.TemplateResponse("index.html", {"request": request, "user":
     ↪  user})

@app.get("/login", response_class=HTMLResponse)
async def login_page(request: Request):
    user = get_current_user(request)
    if user:
        return RedirectResponse(url="/")
    return templates.TemplateResponse("login.html", {"request": request})

@app.get("/signup", response_class=HTMLResponse)
async def signup_page(request: Request):
    user = get_current_user(request)
    if user:
        return RedirectResponse(url="/")
    return templates.TemplateResponse("signup.html", {"request": request})

@app.post("/signup")
async def signup(username: str = Form(...), password: str = Form(...), db: Session
 ↪  = Depends(get_db)):
    db_user = db.query(User).filter(User.username == username).first()
    if db_user:
        raise HTTPException(status_code=400, detail="Username already registered")
    hashed_password = pwd_context.hash(password)
    new_user = User(username=username, password=hashed_password)
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return RedirectResponse(url="/login", status_code=status.HTTP_302_FOUND)

@app.post("/login")
async def login(request: Request, username: str = Form(...), password: str =
 ↪  Form(...), db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.username == username).first()
    if not db_user or not pwd_context.verify(password, db_user.password):
        raise HTTPException(status_code=400, detail="Invalid credentials")
    response = RedirectResponse(url="/", status_code=status.HTTP_302_FOUND)
    response.set_cookie(key="user_id", value=str(db_user.id), httponly=True)  #
 ↪  Updated to use id
    return response

@app.get("/logout")
async def logout(request: Request):
    response = RedirectResponse(url="/login", status_code=status.HTTP_302_FOUND)
    response.delete_cookie("user_id")
    return response

@app.get("/therapy", response_class=HTMLResponse)
def therapy_page(request: Request):
    return templates.TemplateResponse("therapy.html", {"request": request})

@app.get("/meeting-room", response_class=HTMLResponse)
def meeting_room(request: Request):
    return templates.TemplateResponse("meeting_room.html", {"request": request})

@app.get("/ai-chat", response_class=HTMLResponse)
def ai_chat_page(request: Request):
    """
    Serve the AI chat page.
    """
    return templates.TemplateResponse("ai_chat.html", {"request": request})

@app.get("/voice-chatbot", response_class=HTMLResponse)
def voice_chatbot_page(request: Request):
    """
    Serve the Voice Chatbot page.
    """
    return templates.TemplateResponse("voice_chatbot.html", {"request": request})

@app.post("/api/ai-chat")
async def voice_chat(input: TextInput):
    """
    Handle voice chat messages using the AI chat system.
    """
    try:
        response_data = await get_ai_response(input.text, [])
        return {"response": response_data["response"]}
    except Exception as e:
        print(f"Error in voice chat: {str(e)}")
        raise HTTPException(status_code=500, detail="Error processing voice chat
         ↪  response")

# Post_request to analyze sentiment

@app.post("/analyze")
async def analyze_text(input: TextInput, db: Session = Depends(get_db), user: dict
 ↪  = Depends(get_current_user)):
    if not user:
        raise HTTPException(status_code=401, detail="Not logged in")
    result = analyze_sentiment(input.text)
    text_entry = Text(
        user_id=user["uid"],  # Updated to use user_id
        text=input.text,
        tscore=result["confidence"],
        sentiment=result["sentiment"],
        label="text"
    )
    db.add(text_entry)
    db.commit()
    return result

@app.post("/transcribe")
async def transcribe_audio_endpoint(file: UploadFile = File(...), db: Session =
 ↪  Depends(get_db), user: dict = Depends(get_current_user)):
    if not user:
        raise HTTPException(status_code=401, detail="Not logged in")
    temp_path = f"temp_audio_{file.filename}"
    try:
        with open(temp_path, "wb") as buffer:
            buffer.write(await file.read())
        transcription = transcribe_audio(temp_path)  # Pass temp_path directly
        if "text" in transcription:
            result = analyze_sentiment(transcription["text"])
            text_entry = Text(
                user_id=user["uid"],  # Fixed typo from uid to user_id
                text=transcription["text"],
                tscore=result["confidence"],
                sentiment=result["sentiment"],
                label="uploaded" if file.filename else "direct_audio"
            )
            db.add(text_entry)
            db.commit()
            return {"transcription": transcription["text"], "sentiment": result}
        return transcription
    finally:
        if os.path.exists(temp_path):
            os.remove(temp_path)

connected_clients: Dict[str, List[WebSocket]] = {}

USER_COLORS = ["#FF5733", "#33FF57", "#3357FF", "#FF33A1", "#A133FF", "#33FFF5"]

client_nicknames: Dict[str, Dict[WebSocket, tuple]] = {}


async def broadcast_to_room(room_id: str, data: dict):
    for client in connected_clients[room_id]:
        await client.send_json(data)

@app.websocket("/ws/{room_id}")
async def websocket_endpoint(websocket: WebSocket, room_id: str = "general"):
    await websocket.accept()

    if room_id not in connected_clients:
        connected_clients[room_id] = []
        client_nicknames[room_id] = {}

    nickname = await websocket.receive_text()
    color = choice(USER_COLORS)
    connected_clients[room_id].append(websocket)
    message_history = []
    ai_chat_mode = False
    client_nicknames[room_id][websocket] = (nickname, color, message_history,
↪   ai_chat_mode)

    await websocket.send_json({
        "type": "system",
        "message": "Connected to the room.",
    })

    is_alone = len(connected_clients[room_id]) == 1
    if is_alone:
        await websocket.send_json({
            "type": "system",
            "message": "You're alone in the room. Would you like to chat with our
            ↪   AI chatbot?",
            "showAiOption": True
        })

    await broadcast_to_room(room_id, {
        "type": "join",
        "nickname": nickname,
        "color": color,
        "isAlone": is_alone
    })

    try:
        while True:
            message = await websocket.receive_text()

            await broadcast_to_room(room_id, {
                "type": "message",
                "nickname": nickname,
                "message": message,
                "color": color,
                "isAI": False
            })
    except WebSocketDisconnect:
        connected_clients[room_id].remove(websocket)
        is_alone = len(connected_clients[room_id]) == 1
        await broadcast_to_room(room_id, {
            "type": "leave",
            "nickname": nickname,
            "color": color,
            "isAlone": is_alone
        })
        del client_nicknames[room_id][websocket]

        if not connected_clients[room_id]:
            del connected_clients[room_id]
            del client_nicknames[room_id]


async def api_ai_chat(input: TextInput):
    """
    Handle AI chat messages.
    """
    try:
        response_data = await get_ai_response(input.text, [])
        return {"response": response_data["response"]}
    except Exception as e:
        print(f"Error in AI chat: {str(e)}")
        raise HTTPException(status_code=500, detail="Error processing AI response")


audio_room_clients = {}
@app.websocket("/ws/audio-room")
async def audio_room_websocket(websocket: WebSocket):
    await websocket.accept()
    username = None
    try:
        while True:
            data = await websocket.receive_json()
            if data["type"] == "join":
                username = data["username"]
                audio_room_clients[username] = websocket
                for client in audio_room_clients.values():
                    await client.send_json({"type": "join", "username": username})

                if len(audio_room_clients) == 1:
                    await websocket.send_json({
                        "type": "system",
                       "message": "You're alone in the room. Would you like to chat
                         ↪  with our AI chatbot?",
                        "showAiOption": True
                    })
            elif data["type"] == "leave":
                if username in audio_room_clients:
                    del audio_room_clients[username]
                    for client in audio_room_clients.values():
                        await client.send_json({"type": "leave", "username":
                         ↪  username})
            else:
                receiver = data.get("receiver")
                if receiver in audio_room_clients:
                    await audio_room_clients[receiver].send_json(data)
    except WebSocketDisconnect:
        if username in audio_room_clients:
            del audio_room_clients[username]
            for client in audio_room_clients.values():
                await client.send_json({"type": "leave", "username": username})
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/models.py

```
"""
File: models.py
Purpose: Defines database models using SQLAlchemy.
"""
from sqlalchemy import Column, Integer, String, Float, ForeignKey
from .database import Base

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, nullable=False)
    password = Column(String, nullable=False)

class Text(Base):
    __tablename__ = "texts"
    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    text = Column(String, nullable=False)
    tscore = Column(Float, nullable=False)
    sentiment = Column(String, nullable=False)
    label = Column(String, nullable=False)

class Chat(Base):
    __tablename__ = "chat"
    id = Column(Integer, primary_key=True, index=True)
```

```python
    cscore = Column(Float, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    summary = Column(String, nullable=False)

class Therapy(Base):
    __tablename__ = "therapy"
    id = Column(Integer, primary_key=True, index=True)
    sentiment_score = Column(Float, nullable=False)
    audio = Column(String, nullable=True)
    video = Column(String, nullable=True)
    quotes = Column(String, nullable=True)
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/sentiment.py

```python
"""
File: sentiment.py
Purpose: Performs sentiment analysis using the Hugging Face Transformers pipeline.

Flow:
- Text is sent to the /analyze endpoint in main.py.
- This file processes the text and returns the sentiment and confidence score.
"""

from transformers import pipeline

sentiment_pipeline = pipeline("sentiment-analysis")

def analyze_sentiment(text: str):
    """
    Analyze the sentiment of the given text.

    Args:
        text (str): The text to analyze.

    Returns:
        dict: A dictionary containing the sentiment ('positive' or 'negative') and
↪   the confidence score.
    """
    result = sentiment_pipeline(text)[0]
    sentiment = result["label"].lower()
    confidence = round(result["score"], 2)
    return {"sentiment": sentiment, "confidence": confidence}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/transcription.py

```
"""
File: transcription.py
Purpose: Handles audio transcription using the Whisper model.

Flow:
- Audio file is uploaded via the /transcribe endpoint in main.py.
- This file processes the audio and returns the transcription.
"""

import whisper
import os
from fastapi import UploadFile

# use this for higher accuracy - large-v3-turbo
# base model is for standard use
# tiny or small are lightweight models for faster processing


model = whisper.load_model("base")

def transcribe_audio(file_path: str):
    try:
        result = model.transcribe(file_path)
        return {"text": result["text"]}
    except Exception as e:
        return {"error": str(e)}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/backend/chatbot_test/chat_test.py

```
import os
from cerebras.cloud.sdk import Cerebras

client = Cerebras(
    api_key=os.environ.get("CEREBRAS_API_KEY"),  # This is the default and can be
↪   omitted
```

```python
)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "hello, how are you?",
        }
    ],
    model="llama3.1-8b",
)

print(chat_completion)
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/ml_data/ml.py

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/ai_chat.css

```css
.chat-container {
    display: flex;
    flex-direction: column;
    height: 100vh;
    max-width: 800px;
    margin: 0 auto;
    background-color: #f7f7f8;
    border: 1px solid #e0e0e0;
    border-radius: 10px;
    overflow: hidden;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.chat-box {
    flex: 1;
    overflow-y: auto;
    padding: 20px;
    background-color: #ffffff;
    display: flex;
    flex-direction: column;
    gap: 15px;
}

.message {
    padding: 12px 16px;
    border-radius: 12px;
    max-width: 75%;
    word-wrap: break-word;
    font-size: 14px;
    line-height: 1.5;
    box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.message.user {
    background-color: #d1e7dd;
    align-self: flex-end;
    text-align: left;
    color: #155724;
}

.message.ai {
    background-color: #f8f9fa;
    align-self: flex-start;
    text-align: left;
    color: #333;
    border: 1px solid #e0e0e0;
}


.chat-input {
    display: flex;
    padding: 10px;
    background-color: #f1f1f1;
    border-top: 1px solid #e0e0e0;
}

.chat-input input {
    flex: 1;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 8px;
```

```css
    font-size: 14px;
    outline: none;
    transition: border-color 0.3s;
}

.chat-input input:focus {
    border-color: #007bff;
}

.chat-input button {
    margin-left: 10px;
    padding: 12px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s;
}

.chat-input button:hover {
    background-color: #0056b3;
}

.chat-box::-webkit-scrollbar {
    width: 8px;
}

.chat-box::-webkit-scrollbar-thumb {
    background-color: #ccc;
    border-radius: 4px;
}

.chat-box::-webkit-scrollbar-thumb:hover {
    background-color: #aaa;
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/audio_room.js

```javascript
const userGrid = document.getElementById("user-grid");
const toggleMicButton = document.getElementById("toggle-mic");
const debugLog = document.getElementById("debug-log");
const chatbotPrompt = document.getElementById("chatbot-prompt");
const chatWithBotButton = document.getElementById("chat-with-bot");
const stayHereButton = document.getElementById("stay-here");

let localStream = null;
let peers = {};
let isMicOn = false;
let signalingSocket = null;
let username = prompt("Enter your name to join the audio room:");


function connectToSignalingServer() {
    signalingSocket = new WebSocket("ws://127.0.0.1:8000/ws/audio-room");

    signalingSocket.onopen = () => {
        log("Connected to the room.");
        signalingSocket.send(JSON.stringify({ type: "join", username }));
    };

    signalingSocket.onmessage = async (event) => {
        const data = JSON.parse(event.data);
        if (data.type === "join") {
            addUserToGrid(data.username);
        } else if (data.type === "leave") {
            removeUserFromGrid(data.username);
        } else if (data.type === "offer") {
            await handleOffer(data.offer, data.sender);
        } else if (data.type === "answer") {
            await handleAnswer(data.answer, data.sender);
        } else if (data.type === "ice-candidate") {
            await handleIceCandidate(data.candidate, data.sender);
        } else if (data.type === "system" && data.showAiOption) {
            chatbotPrompt.classList.remove("hidden");
        }
    };

    signalingSocket.onclose = () => log("Disconnected from the room.");
}


function addUserToGrid(username) {
    if (document.getElementById(`user-${username}`)) return; // Avoid duplicates
    const userDiv = document.createElement("div");
    userDiv.id = `user-${username}`;
    userDiv.className = "flex flex-col items-center";
    userDiv.innerHTML = `
        <div class="avatar">${username[0].toUpperCase()}</div>
        <p class="mt-2">${username}</p>
    `;
    userGrid.appendChild(userDiv);
}


function removeUserFromGrid(username) {
    const userDiv = document.getElementById(`user-${username}`);
    if (userDiv) userDiv.remove();
}


toggleMicButton.addEventListener("click", async () => {
    if (!localStream) {
        localStream = await navigator.mediaDevices.getUserMedia({ audio: true });
        broadcastStream();
    }

    isMicOn = !isMicOn;
    localStream.getAudioTracks()[0].enabled = isMicOn;
    toggleMicButton.textContent = isMicOn ? "Microphone On" : "Microphone Off";
});


chatWithBotButton.addEventListener("click", () => {
    window.location.href = "/ai-chat";
});

stayHereButton.addEventListener("click", () => {
    chatbotPrompt.classList.add("hidden");
});


function broadcastStream() {
    Object.values(peers).forEach(peer => {
        localStream.getTracks().forEach(track => peer.addTrack(track,
 ↪  localStream));
```

```
    });
}


async function handleOffer(offer, sender) {
    const peerConnection = createPeerConnection(sender);
    await peerConnection.setRemoteDescription(new RTCSessionDescription(offer));
    const answer = await peerConnection.createAnswer();
    await peerConnection.setLocalDescription(answer);
    signalingSocket.send(JSON.stringify({ type: "answer", answer, sender:
 ↪  username, receiver: sender }));
}


async function handleAnswer(answer, sender) {
    const peerConnection = peers[sender];
    if (peerConnection) {
        await peerConnection.setRemoteDescription(new
         ↪  RTCSessionDescription(answer));
    }
}


async function handleIceCandidate(candidate, sender) {
    const peerConnection = peers[sender];
    if (peerConnection) {
        await peerConnection.addIceCandidate(new RTCIceCandidate(candidate));
    }
}

function createPeerConnection(peerId) {
    const peerConnection = new RTCPeerConnection();
    peers[peerId] = peerConnection;

    peerConnection.onicecandidate = (event) => {
        if (event.candidate) {
            signalingSocket.send(JSON.stringify({ type: "ice-candidate", candidate:
 ↪  event.candidate, sender: username, receiver: peerId }));
        }
    };

    peerConnection.ontrack = (event) => {
        const remoteAudio = new Audio();
        remoteAudio.srcObject = event.streams[0];
        remoteAudio.play();
    };

    return peerConnection;
}

function log(message) {
    debugLog.innerHTML += `<p>${message}</p>`;
}

connectToSignalingServer();
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/meeting-room.js

```
/*
File: meeting-room.js
Purpose: Handles WebSocket communication for the meeting room.

Flow:
- Connects to the WebSocket endpoint (/ws/{room_id}) in main.py.
- Sends and receives messages in real-time.
*/

let socket;
let nickname;

function setNickname() {
    const nicknameInput = document.getElementById("nickname-input");
    nickname = nicknameInput.value.trim();

    if (!nickname) {
        alert("Please enter a nickname.");
        return;
    }

    document.getElementById("nickname-section").style.display = "none";
    document.getElementById("chat-section").style.display = "block";

    connectToRoom();
}
```

```
function connectToRoom() {
    const urlParams = new URLSearchParams(window.location.search);
    let roomId = urlParams.get("room_id") || "general"; // Default to "general" if
     ↪  no room ID is provided

    document.getElementById("room-info").innerText = `You are in room: ${roomId}`;
    socket = new WebSocket(`ws://127.0.0.1:8000/ws/${roomId}`);

    // Debugging log to confirm socket initialization
    console.log("Connecting to WebSocket...");

    socket.onopen = () => {
        console.log("WebSocket connection established.");
        socket.send(nickname);
        const messages = document.getElementById("messages");
        const messageElement = document.createElement("div");
        messageElement.className = "system-message";
        messageElement.style.textAlign = "center"; // Center-align the message
        messageElement.innerText = "Connected to the room.";
        messages.appendChild(messageElement); // Ensure the message is appended to
↪   the chatbox
        messages.scrollTop = messages.scrollHeight; // Scroll to the latest message
    };

    socket.onmessage = (event) => {
        console.log("Message received:", event.data); // Debugging log
        const data = JSON.parse(event.data);
        const messages = document.getElementById("messages");
        const messageElement = document.createElement("div");

        if (data.type === "message") {
            // Ignore messages sent by the current user
            if (data.nickname === nickname) return;

            // Append all received messages to the chatbox
            messageElement.className = data.isAI ? "message ai" : "message"; // Use
↪   "ai" class for AI messages
            const senderSpan = document.createElement("span");
            senderSpan.style.color = data.color;
            senderSpan.style.fontWeight = "bold";
            senderSpan.innerText = data.nickname;
            const messageSpan = document.createElement("span");
            messageSpan.innerText = `: ${data.message}`;

            messageElement.appendChild(senderSpan);
            messageElement.appendChild(messageSpan);
            messages.appendChild(messageElement);
            messages.scrollTop = messages.scrollHeight; // Scroll to the latest
↪   message
        } else if (data.type === "join" || data.type === "leave") {
            if (data.nickname === nickname && data.type === "join") return; //
             ↪  Prevent duplicate join message for the current user
            messageElement.className = "system-message";
            messageElement.style.textAlign = "center"; // Center-align the message
            messageElement.innerText = data.type === "join"
                ? `□ ${data.nickname} joined the room.`
                : `□ ${data.nickname} left the room.`;

            messages.appendChild(messageElement);
            messages.scrollTop = messages.scrollHeight;

            // Show prompt if only one user is present
            if (data.isAlone) {
                const chatbotPrompt = document.getElementById("chatbot-prompt");
                chatbotPrompt.style.display = "block"; // Show the chatbot prompt
            }
        } else if (data.type === "system" && data.showAiOption) {
            // Handle system message with AI chat option
            const chatbotPrompt = document.getElementById("chatbot-prompt");
            chatbotPrompt.style.display = "block";
        }
    };

    socket.onclose = () => {
        console.log("WebSocket connection closed.");
        alert("Disconnected from the room.");
        window.location.href = "/";
    };

    socket.onerror = (error) => {
        console.error("WebSocket error:", error);
    };
}
```

```
function sendMessage() {
    const messageInput = document.getElementById("message-input");
    const message = messageInput.value.trim();
    if (message && socket) {
        // Show the user's message in the chatbox
        const messages = document.getElementById("messages");
        const messageElement = document.createElement("div");
        messageElement.className = "message user"; // Ensure it's styled as a user
↪   message
        messageElement.innerText = message;
        messages.appendChild(messageElement);

        // Send the message to the server
        socket.send(message);
        messageInput.value = ""; // Clear the input field
        messages.scrollTop = messages.scrollHeight; // Scroll to the latest message
    }
}

function leaveRoom() {
    if (socket) {
        socket.close();
    }
}

function redirectToAiChat() {
    window.location.href = "/ai-chat";
}

function stayInRoom() {
    const aiStatus = document.getElementById("ai-status");
    aiStatus.style.display = "none";
}
function redirectToNewAudioRoom() {
    const username = prompt("Enter your name to join the audio room:");
    if (!username) {
        alert("Name is required to join the audio room.");
        return;
    }

    const socket = new WebSocket("ws://127.0.0.1:8000/ws/audio-room");

    socket.onopen = () => {
        console.log("Connected to the audio room WebSocket.");
        socket.send(JSON.stringify({ type: "join", username }));
    };

    socket.onmessage = (event) => {
        const data = JSON.parse(event.data);
        if (data.type === "join") {
            console.log(`${data.username} joined the audio room.`);
        } else if (data.type === "leave") {
            console.log(`${data.username} left the audio room.`);
        } else if (data.type === "system") {
            console.log(`System message: ${data.message}`);
        }
    };

    socket.onclose = () => {
        console.log("Disconnected from the audio room WebSocket.");
    };

    socket.onerror = (error) => {
        console.error("WebSocket error:", error);
    };
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/meeting_room.css

```
.card {
    max-width: 600px;
    margin: 50px auto;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    background-color: #f9f9f9;
    font-family: Arial, sans-serif;
}


.card h2 {
    text-align: center;
    color: #333;
}


#nickname-section {
    text-align: center;
    margin-bottom: 20px;
}

#nickname-input {
    flex: 1;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button {
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    background-color: #007bff;
    color: white;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

x
#chat-section {
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.chat-box {
    height: 300px;
    overflow-y: auto;
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 10px;
    background-color: #fff;
}
```

```css
.chat-box .message {
    margin-bottom: 10px;
    padding: 8px;
    border-radius: 5px;
    max-width: 80%;
}

.chat-box .message.user {
    background-color: #d1e7dd;
    align-self: flex-start;
    text-align: left;
}

.chat-box .message.other {
    background-color: #f8d7da;
    align-self: flex-end;
    text-align: right;
}

.system-message {
    text-align: center;
    font-style: italic;
    color: gray;
}

/* Chat input styling */
.chat-input {
    display: flex;
    gap: 10px;
}

#message-input {
    flex: 1;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

.leave-btn {
    align-self: center;
    background-color: #dc3545;
}
```

```css
.leave-btn:hover {
    background-color: #a71d2a;
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/scripts.js

```javascript
/*
File: scripts.js
Purpose: Provides frontend functionality for sentiment analysis, audio
 ↪  transcription, and WebSocket chat.

Flow:
1. Sentiment Analysis:
   - Text input is sent to the /analyze endpoint in main.py.
   - Displays the sentiment result.

2. Audio Transcription:
   - Records or uploads audio.
   - Sends the audio to the /transcribe endpoint in main.py.
   - Displays the transcription and sentiment result.

3. WebSocket Chat:
   - Connects to the WebSocket endpoint (/ws/{room_id}) in main.py.
   - Sends and receives messages in real-time.
*/

async function analyzeText() {
    const text = document.getElementById("text-input").value;
    const spinner = document.getElementById("spinner");
    const resultEl = document.getElementById("result");

    spinner.style.display = "block";
    resultEl.innerText = "";

    try {
        const response = await fetch("http://127.0.0.1:8000/analyze", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ text })
        });
        if (!response.ok) {
            throw new Error("Failed to analyze text.");
        }

        const data = await response.json();
        resultEl.innerText = `Sentiment: ${data.sentiment.toUpperCase()}
 ↪  (Confidence: ${data.confidence})`;
    } catch (error) {
        resultEl.innerText = "Error: Unable to analyze text.";
        console.error(error);
    } finally {
        spinner.style.display = "none";
    }
}

let mediaRecorder;
let audioChunks = [];
let isRecording = false;

async function toggleRecording() {
    const recordBtn = document.getElementById("record-btn");
    const spinner = document.getElementById("spinner");
    const resultEl = document.getElementById("result");

    try {
        if (!isRecording) {
            const stream = await navigator.mediaDevices.getUserMedia({ audio: true
             ↪  });
            mediaRecorder = new MediaRecorder(stream);

            mediaRecorder.ondataavailable = (event) => {
                audioChunks.push(event.data);
            };

            mediaRecorder.onstop = async () => {
                spinner.style.display = "block";
                resultEl.innerText = "Transcribing your speech...";

                const audioBlob = new Blob(audioChunks, { type: 'audio/webm' });
                const formData = new FormData();
                formData.append("file", audioBlob, "voice_input.webm");
```

```
try {
    const response = await
     ↪  fetch("http://127.0.0.1:8000/transcribe/", {
        method: "POST",
        body: formData
    });

    if (!response.ok) {
        throw new Error("Failed to transcribe audio.");
    }

    const data = await response.json();
    console.log("Backend response:", data); // Debugging log

    // Ensure the transcription field exists
    const transcript = data.transcription || "No transcription
     ↪  available.";
    console.log("Transcript:", transcript); // Log the transcript
     ↪  for debugging

    // Debugging logs for DOM elements
    const textInputEl = document.getElementById("text-input");
    const resultEl = document.getElementById("result");
    const transcribe_text = document.getElementById("trans_text");
    console.log("Text Input Element:", textInputEl);
    console.log("Result Element:", resultEl);

    // Update the UI
    if (textInputEl) {
        textInputEl.value = transcript;
    } else {
        console.error("Text input element not found.");
    }

    if (transcribe_text) {
        transcribe_text.innerText = `You said: "${transcript}"`;
    } else {
        console.error("Result element not found.");
    }

    console.log("Transcribe Text:", transcribe_text);

    // Handle sentiment if available
```

```
                    if (data.sentiment) {
                        console.log("Sentiment:", data.sentiment);
                        resultEl.innerText += `\nSentiment:
↪   ${data.sentiment.sentiment.toUpperCase()} (Confidence:
↪   ${data.sentiment.confidence})`;
                    } else {
                        console.warn("Sentiment data not found in the response.");
                    }

                    analyzeText();
                } catch (error) {
                    resultEl.innerText = "Error: Unable to transcribe audio.";
                    console.error(error);
                } finally {
                    spinner.style.display = "none";
                }

                audioChunks = [];
            };

            audioChunks = [];
            mediaRecorder.start();
            recordBtn.innerText = "□ Stop Recording";
            isRecording = true;
        } else {
            mediaRecorder.stop();
            isRecording = false;
            recordBtn.innerText = "□ Start Recording";
        }
    } catch (error) {
        alert("Error accessing microphone. Please check permissions.");
        console.error(error);
    }
}

async function uploadAudio() {
    const fileInput = document.getElementById("audio-input");
    const spinner = document.getElementById("spinner");
    const resultEl = document.getElementById("result");

    if (fileInput.files.length === 0) {
        alert("Please select an audio file.");
        return;
    }

    const formData = new FormData();
    formData.append("file", fileInput.files[0]);

    spinner.style.display = "block";
    resultEl.innerText = "Transcribing audio...";

    try {
        const response = await fetch("http://127.0.0.1:8000/transcribe/", {
            method: "POST",
            body: formData
        });

        if (!response.ok) {
            throw new Error("Failed to transcribe audio.");
        }

        const data = await response.json();
        console.log("Backend response:", data); // Debugging log
        // Ensure the transcription field exists
        const transcript = data.transcription || "No transcription available.";

        document.getElementById("text-input").value = transcript;
        resultEl.innerText = `You said: "${transcript}"`;

        analyzeText();
    } catch (error) {
        resultEl.innerText = "Error: Unable to transcribe audio.";
        console.error(error);
    } finally {
        spinner.style.display = "none";
    }
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/styles.css

```
/*
File: styles.css
Purpose: Provides styling for the frontend pages.
```

```css
*/

body {
    margin: 0;
    padding: 0;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #0f0c29, #302b63, #24243e);
    color: white;
    display: flex;
    align-items: center;
    justify-content: center;
    min-height: 100vh;
}

.card {
    background: rgba(255, 255, 255, 0.05);
    border: 1px solid rgba(255, 255, 255, 0.15);
    border-radius: 20px;
    padding: 30px;
    max-width: 500px;
    width: 90%;
    box-shadow: 0 0 30px rgba(0, 255, 255, 0.15);
    backdrop-filter: blur(10px);
    text-align: center;
    animation: fadeIn 1.5s ease;
}

h1 {
    color: #00BFFF;
    margin-bottom: 20px;
    font-size: 26px;
}

textarea {
    width: 100%;
    height: 100px;
    padding: 12px;
    border-radius: 10px;
    border: none;
    resize: none;
    font-size: 16px;
    background: #1e1e2f;
    color: white;
    outline: none;
}

button {
    margin-top: 15px;
    background: #00BFFF;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 10px;
    font-size: 16px;
    cursor: pointer;
    transition: background 0.3s ease;
}

button:hover {
    background: #00aee0;
}

#result, #trans_text {
    margin-top: 20px;
    font-size: 18px;
    color: #aad2ff;
}

#spinner {
    display: none;
    margin: 15px auto;
    border: 4px solid rgba(255, 255, 255, 0.2);
    border-top: 4px solid #00BFFF;
    border-radius: 50%;
    width: 30px;
    height: 30px;
    animation: spin 1s linear infinite;
}

.waveform {
    margin-top: 10px;
    display: none;
}

.bar {
    display: inline-block;
```

```css
    width: 4px;
    height: 20px;
    margin: 0 1px;
    background-color: #00BFFF;
    animation: bounce 1s infinite ease-in-out;
}

.bar:nth-child(2) { animation-delay: 0.1s; }
.bar:nth-child(3) { animation-delay: 0.2s; }
.bar:nth-child(4) { animation-delay: 0.3s; }

@keyframes bounce {
    0%, 100% { height: 20px; }
    50% { height: 40px; }
}

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(30px); }
    to { opacity: 1; transform: translateY(0); }
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

.chat-box {
    max-height: 300px;
    overflow-y: auto;
    padding: 10px;
    border: 1px solid rgba(255, 255, 255, 0.2);
    border-radius: 10px;
    background: rgba(0, 0, 0, 0.5);
    margin-bottom: 10px;
    display: flex;
    flex-direction: column; /* Stack messages vertically */
    align-items: flex-start; /* Align all messages to the left by default */
}

#chatbot-prompt {
    margin-bottom: 10px;
    padding: 10px;
    background: rgba(0, 191, 255, 0.1);
    border-radius: 8px;
    text-align: center;
    font-style: italic;
}

.message {
    display: flex; /* Use flexbox for better alignment */
    flex-direction: row; /* Arrange spans in a row */
    align-items: flex-start; /* Align spans to the top-left of the container */
    text-align: left; /* Ensure text is left-aligned */
    background-color: #f1f1f1; /* Light background for general messages */
    color: #333; /* Dark text for contrast */
    padding: 10px; /* Add padding inside the message box */
    border-radius: 8px; /* Rounded corners for the message box */
    margin-bottom: 10px; /* Add spacing between messages */
    max-width: 70%; /* Limit the width of the message box */
    word-wrap: break-word; /* Ensure long words wrap properly */
    gap: 5px; /* Add spacing between spans */
}

.message span {
    text-align: left; /* Ensure text inside spans is left-aligned */
    margin: 0; /* Remove any default margin */
    padding: 0; /* Remove any default padding */
}

.message.user {
    background-color: #d1e7dd; /* Light green background for user messages */
    align-self: flex-end; /* Align user messages to the right */
    text-align: right; /* Right-align text for user messages */
    color: #155724; /* Dark green text for contrast */
}

.message.ai {
    background-color: #f8d7da; /* Light red background for AI messages */
    align-self: flex-start; /* Align AI messages to the left */
    text-align: left; /* Left-align text for AI messages */
    color: #721c24; /* Dark red text for contrast */
}

.chat-input {
    display: flex;
    gap: 10px;
```

```
}

#nickname-input{
    flex: 1;
    padding: 10px;
    border-radius: 10px;
    border: 1px solid rgba(255, 255, 255, 0.2);
    background: #1e1e2f;
    color: white;
    outline: none;
    font-size: 16px;
}
.chat-input input {
    flex: 1;
    padding: 10px;
    border-radius: 10px;
    border: 1px solid rgba(255, 255, 255, 0.2);
    background: #1e1e2f;
    color: white;
    outline: none;
    font-size: 16px;
}

chat-input button {
    padding: 10px 20px;
    border-radius: 10px;
    background: #00BFFF;
    color: white;
    border: none;
    cursor: pointer;
    transition: background 0.3s ease;
}

chat-input button:hover {
    background: #00aee0;
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/static/voice_chatbot.js

```
const startButton = document.getElementById("start-recording");
const stopButton = document.getElementById("stop-recording");
const chatLog = document.getElementById("chat-log");
const responseDiv = document.getElementById("response");

let recognition;
let isRecording = false;

// Initialize Speech Recognition
if ("SpeechRecognition" in window || "webkitSpeechRecognition" in window) {
    recognition = new (window.SpeechRecognition ||
 ↪  window.webkitSpeechRecognition)();
    recognition.continuous = true;
    recognition.interimResults = true;
    recognition.maxAlternatives = 1;
    recognition.lang = "en-US";

    // Increase timeout duration
    recognition.speechRecognitionTimeout = 10000; // 10 seconds

    recognition.onstart = () => {
        isRecording = true;
        responseDiv.textContent = "Listening...";
        startButton.disabled = true;
        stopButton.disabled = false;
        document.getElementById("waveform").style.display = "block";
    };

    recognition.onresult = async (event) => {
        const userMessage = event.results[event.results.length -
         ↪  1][0].transcript.trim();
        if (userMessage) {
            addMessageToLog(userMessage, "user");
            responseDiv.textContent = "Processing...";
            await sendToChatbot(userMessage);
        }
    };

    recognition.onerror = (event) => {
        console.error("Speech Recognition Error:", event.error);
        if (event.error === "no-speech") {
            responseDiv.textContent = "No speech detected. Please try again.";
            // Automatically restart recognition
            try {
                recognition.stop();
                setTimeout(() => {
                    if (isRecording) {
                        recognition.start();
                    }
                }, 100);
            } catch (error) {
                console.error("Error restarting recognition:", error);
            }
        } else {
            responseDiv.textContent = `Error: ${event.error}. Please try again.`;
            isRecording = false;
            startButton.disabled = false;
            stopButton.disabled = true;
        }
    };

    recognition.onend = () => {
        console.log("Recognition ended");
        // Only stop if the stop button was clicked
        if (isRecording) {
            try {
                recognition.start();
            } catch (error) {
                console.error("Error restarting recognition:", error);
                isRecording = false;
                startButton.disabled = false;
                stopButton.disabled = true;
                responseDiv.textContent = "Recognition stopped. Click Start to
↪   begin again.";
            }
        } else {
            responseDiv.textContent = "Stopped listening.";
            startButton.disabled = false;
            stopButton.disabled = true;
            document.getElementById("waveform").style.display = "none";
        }
    };
} else {
    responseDiv.textContent = "Speech Recognition is not supported in this
↪   browser.";
    startButton.disabled = true;
    stopButton.disabled = true;
}
```

```
// Start Recording
startButton.addEventListener("click", () => {
    if (!isRecording) {
        try {
            recognition.start();
        } catch (error) {
            responseDiv.textContent = "Error starting recognition.";
        }
    }
});

// Stop Recording
stopButton.addEventListener("click", () => {
    if (isRecording) {
        isRecording = false; // Set this first so onend doesn't restart
        try {
            recognition.stop();
        } catch (error) {
            console.error("Error stopping recognition:", error);
        }
    }
});

// Add message to chat log
function addMessageToLog(message, sender) {
    const messageDiv = document.createElement("div");
    messageDiv.className = `message ${sender}`;
    messageDiv.textContent = message;
    chatLog.appendChild(messageDiv);
    chatLog.scrollTop = chatLog.scrollHeight;
}

// Send message to chatbot API
async function sendToChatbot(message) {
    try {
        const response = await fetch("http://127.0.0.1:8000/api/voice-chat", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ text: message })
        });

        if (!response.ok) {
            throw new Error("Failed to connect to the chatbot API.");
        }

        const data = await response.json();
        addMessageToLog(data.response, "ai");
        responseDiv.textContent = "Ready for next input.";
    } catch (error) {
        console.error("Error:", error);
        addMessageToLog("Error: Unable to connect to the server.", "ai");
        responseDiv.textContent = "Error occurred.";
    }
}
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/ai_chat.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MindSentry Nova - AI Chat</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <link
    ↪   href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&family=Roboto+Mono:
    ↪   rel="stylesheet">
    <style>
        /* Custom Animations */
        @keyframes pulseGlow {
            0%, 100% { box-shadow: 0 0 10px rgba(96, 165, 250, 0.5); }
            50% { box-shadow: 0 0 20px rgba(96, 165, 250, 0.8); }
        }
        @keyframes float {
            0%, 100% { transform: translateY(0); }
            50% { transform: translateY(-10px); }
        }
        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(20px); }
            to { opacity: 1; transform: translateY(0); }
        }
        /* Global Styles */
        body {
           background: radial-gradient(circle at center, #1a1a2e 0%, #0f0f1f 100%);
            font-family: 'Roboto Mono', monospace;
            color: #e5e7eb;
            overflow-x: hidden;
        }
        .nova-text {
            font-family: 'Orbitron', sans-serif;
            background: linear-gradient(90deg, #60A5FA, #A78BFA);
            -webkit-background-clip: text;
            background-clip: text;
            color: transparent;
        }
        .chat-container {
            max-width: 800px;
            margin: 2rem auto;
            background: rgba(255, 255, 255, 0.05);
            border: 1px solid rgba(255, 255, 255, 0.1);
            border-radius: 20px;
            backdrop-filter: blur(15px);
            display: flex;
            flex-direction: column;
            height: calc(100vh - 4rem);
            animation: fadeIn 0.8s ease-out;
        }
        .chat-header {
            padding: 1.5rem;
            border-bottom: 1px solid rgba(255, 255, 255, 0.1);
            text-align: center;
        }
        .chat-box {
            flex: 1;
            overflow-y: auto;
            padding: 1.5rem;
            scroll-behavior: smooth;
        }
        .message {
            margin-bottom: 1rem;
            padding: 1rem;
            border-radius: 15px;
            max-width: 70%;
            word-wrap: break-word;
            animation: fadeIn 0.5s ease-out;
        }
        .message.user {
            background: linear-gradient(135deg, #60A5FA, #3B82F6);
            color: white;
            margin-left: auto;
            text-align: right;
        }
        .message.ai {
            background: linear-gradient(135deg, #A78BFA, #7C3AED);
            color: white;
            margin-right: auto;
            text-align: left;
        }
        .chat-input {
            display: flex;
            padding: 1rem;
            background: rgba(255, 255, 255, 0.05);
            border-top: 1px solid rgba(255, 255, 255, 0.1);
        }
        .chat-input input {
            flex: 1;
            padding: 0.75rem;
            background: rgba(255, 255, 255, 0.1);
            border: 1px solid rgba(255, 255, 255, 0.2);
            border-radius: 10px;
            color: white;
            font-size: 1rem;
            outline: none;
            transition: border-color 0.3s;
        }
        .chat-input input:focus {
            border-color: #60A5FA;
        }
        .chat-input button {
            margin-left: 0.75rem;
            padding: 0.75rem 1.5rem;
            background: linear-gradient(90deg, #60A5FA, #A78BFA);
            color: white;
            border: none;
            border-radius: 10px;
            font-size: 1rem;
            font-weight: 600;
            cursor: pointer;
            transition: transform 0.3s, background 0.3s;
        }
        .chat-input button:hover {
            background: linear-gradient(90deg, #3B82F6, #7C3AED);
            transform: scale(1.05);
        }
        .chat-input button:disabled {
            background: #6B7280;
            cursor: not-allowed;
            transform: none;
        }
        .back-btn {
            position: fixed;
            bottom: 1.5rem;
            right: 1.5rem;
            padding: 0.75rem 1.5rem;
            background: linear-gradient(90deg, #60A5FA, #A78BFA);
            color: white;
            border-radius: 50px;
            font-weight: 600;
            transition: all 0.3s;
            animation: pulseGlow 2s infinite;
        }
        .back-btn:hover {
            background: linear-gradient(90deg, #3B82F6, #7C3AED);
            transform: scale(1.1);
        }
    </style>
</head>
<body>
    <div class="chat-container">
        <div class="chat-header">
            <h2 class="text-2xl md:text-3xl font-bold nova-text">MindSentry Nova -
             ↪  Cosmic Chat</h2>
            <p class="text-sm md:text-base text-gray-400 mt-1">Connect with the
             ↪  universe through your thoughts</p>
        </div>
        <div id="chat-box" class="chat-box"></div>
        <div class="chat-input">
            <input id="chat-input" type="text" placeholder="Share your cosmic
             ↪  thoughts...">
            <button onclick="sendMessage()">Send</button>
        </div>
    </div>
```

```
<button onclick="window.location.href='/'" class="back-btn">� Back to
↪  Nova</button>
<script>
    const chatBox = document.getElementById("chat-box");
    const chatInput = document.getElementById("chat-input");
    let isSending = false;

    function appendMessage(content, isUser) {
        const messageElement = document.createElement("div");
        messageElement.className = `message ${isUser ? "user" : "ai"}`;
        messageElement.textContent = content;
        chatBox.appendChild(messageElement);
        chatBox.scrollTop = chatBox.scrollHeight;
    }

    async function sendMessage() {
        if (isSending) return;
        const message = chatInput.value.trim();
        if (!message) return;

        isSending = true;
        chatInput.disabled = true;
        chatInput.nextElementSibling.disabled = true;
        appendMessage(message, true);
        chatInput.value = "";

        try {
            const response = await fetch("/api/ai-chat", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ text: message })
            });

            if (response.ok) {
                const data = await response.json();
                appendMessage(data.response, false);
            } else {
                appendMessage("Error: Unable to get a response from the AI.",
                 ↪  false);
            }
        } catch (error) {
            appendMessage("Error: Unable to connect to the server.", false);
        } finally {
                isSending = false;
                chatInput.disabled = false;
                chatInput.nextElementSibling.disabled = false;
                chatInput.focus();
            }
        }

        // Handle Enter key
        chatInput.addEventListener("keypress", (e) => {
            if (e.key === "Enter" && !isSending) {
                sendMessage();
            }
        });

        // Initial welcome message
        appendMessage("Welcome to MindSentry Nova's Cosmic Chat. Share your
         ↪  thoughts, and let's explore the universe together.", false);
    </script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/audio_room.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Audio Room</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        .avatar {
            width: 80px;
            height: 80px;
            border-radius: 50%;
            background-color: #60A5FA;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 1.5rem;
            font-weight: bold;
            color: white;
            transition: box-shadow 0.3s ease;
        }
        .avatar.speaking {
            box-shadow: 0 0 20px rgba(96, 165, 250, 0.8);
        }
    </style>
</head>
<body class="bg-gray-900 text-white flex flex-col items-center justify-center
 ↪  min-h-screen">
    <h1 class="text-4xl font-bold mb-2">Audio Room</h1>
    <p id="room-info" class="text-lg mb-6">You are in room: general</p>
    <div id="chatbot-prompt" class="hidden text-center mb-6">
        <p class="mb-4">You're alone in the room. Would you like to chat with our
         ↪  AI chatbot?</p>
        <button id="chat-with-bot" class="px-4 py-2 bg-blue-500 rounded-lg
         ↪  hover:bg-blue-600 transition">Chat with Chatbot</button>
        <button id="stay-here" class="px-4 py-2 bg-gray-500 rounded-lg
         ↪  hover:bg-gray-600 transition">Stay Here</button>
    </div>
    <div id="user-grid" class="grid grid-cols-2 md:grid-cols-4 gap-6">
        <!-- User avatars will be dynamically added here -->
    </div>
    <button id="toggle-mic" class="mt-8 px-6 py-3 bg-blue-500 rounded-lg
     ↪  hover:bg-blue-600 transition">
        Toggle Microphone
    </button>
    <div id="debug-log" class="mt-6 text-sm text-gray-400"></div>
    <script src="/static/audio_room.js"></script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>MindSentry Nova – Echoes of Emotion</title>
    <script src="https://cdn.tailwindcss.com"></script>
```

```
<link
↪   href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&family=Roboto+Mono:
↪   rel="stylesheet">
<style>
    /* Custom Animations */
    @keyframes orbit {
        0% { transform: rotate(0deg) translateX(10px) rotate(0deg); }
        100% { transform: rotate(360deg) translateX(10px) rotate(-360deg); }
    }
    @keyframes pulseGlow {
        0%, 100% { box-shadow: 0 0 10px rgba(96, 165, 250, 0.5); }
        50% { box-shadow: 0 0 20px rgba(96, 165, 250, 0.8); }
    }
    @keyframes wavePulse {
        0%, 100% { transform: scaleY(1); }
        50% { transform: scaleY(2.5); }
    }
    @keyframes float {
        0%, 100% { transform: translateY(0); }
        50% { transform: translateY(-15px); }
    }
    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
    /* Global Styles */
    body {
       background: radial-gradient(circle at center, #1a1a2e 0%, #0f0f1f 100%);
        overflow-x: hidden;
        font-family: 'Roboto Mono', monospace;
    }
    .nova-text {
        font-family: 'Orbitron', sans-serif;
        background: linear-gradient(90deg, #60A5FA, #A78BFA);
        -webkit-background-clip: text;
        background-clip: text;
        color: transparent;
    }
    .card {
        background: rgba(255, 255, 255, 0.03);
        border: 1px solid rgba(255, 255, 255, 0.05);
        backdrop-filter: blur(15px);
        transition: all 0.5s cubic-bezier(0.4, 0, 0.2, 1);
        }
        .card:hover {
            transform: scale(1.05) rotate(1deg);
            border-color: rgba(96, 165, 250, 0.3);
            box-shadow: 0 0 30px rgba(96, 165, 250, 0.2);
        }
        .waveform .bar {
            display: inline-block;
            width: 5px;
            height: 25px;
            margin: 0 3px;
            background: linear-gradient(180deg, #60A5FA, #1D4ED8);
            border-radius: 5px;
            animation: wavePulse 1.2s infinite ease-in-out;
        }
        .waveform .bar:nth-child(2) { animation-delay: 0.2s; }
        .waveform .bar:nth-child(3) { animation-delay: 0.4s; }
        .waveform .bar:nth-child(4) { animation-delay: 0.6s; }
        .particle {
            position: absolute;
            width: 6px;
            height: 6px;
            background: #60A5FA;
            border-radius: 50%;
            animation: orbit 6s infinite linear;
        }
        #spinner {
            display: none;
            border: 4px solid rgba(255, 255, 255, 0.2);
            border-top: 4px solid #60A5FA;
            border-radius: 50%;
            width: 30px;
            height: 30px;
            animation: spin 1s linear infinite;
        }
    </style>
</head>
<body class="text-white relative">
    <!-- Cosmic Background -->
    <div class="fixed inset-0 pointer-events-none">
        <div class="particle top-20 left-10"></div>
       <div class="particle top-40 right-20" style="animation-duration: 8s;"></div>
```

```
        <div class="particle bottom-30 left-30" style="animation-duration:
         ↪  5s;"></div>
    </div>

    <!-- Hero Section -->
    <header class="relative h-screen flex items-center justify-center
     ↪  overflow-hidden">
        <div class="absolute inset-0 bg-gradient-to-t from-[#0f0f1f]
         ↪  via-transparent to-transparent z-10"></div>
        <div class="relative z-20 flex text-center px-6">
           <div class="flex flex-col items-center justify-center max-w-2xl mx-auto
             ↪  text-center">
                <h1 class="text-5xl md:text-7xl font-bold nova-text
                 ↪  animate-[pulseGlow_2s_infinite]">
                    MindSentry Nova
                </h1>
                <p class="mt-4 text-xl md:text-2xl text-gray-300 max-w-2xl mx-auto
                 ↪  animate-[float_3s_infinite]">
                    A cosmic journey into your emotions, powered by AI.
                </p>
                <button class="mt-8 px-10 py-4 bg-gradient-to-r from-blue-500
                 ↪  to-purple-600 rounded-full font-semibold text-white
                 ↪  hover:from-blue-600 hover:to-purple-700 transition-all
                 ↪  duration-500 animate-[pulseGlow_2s_infinite]"
                 ↪  onclick="scrollToSection('analyze')">
                    Explore Now
                </button>
            </div>
            <div width="300px" height="400px"><spline-viewer
             ↪  url="https://prod.spline.design/B7t4v-
             ↪  LpuyUgMf1e/scene.splinecode"></spline-viewer><div>
        </div>
        <div class="absolute bottom-0 left-0 w-full h-1 bg-gradient-to-r
         ↪  from-transparent via-blue-400 to-transparent"></div>
    </header>

    <!-- Navigation -->
    <nav class="sticky top-0 z-30 bg-[#0f0f1f]/90 backdrop-blur-lg py-4">
        <div class="max-w-7xl mx-auto flex justify-center gap-12 px-6">
            <button class="nav-tab text-gray-300 hover:nova-text text-lg
             ↪  font-semibold transition-all duration-300"
             ↪  data-section="analyze">Analyze</button>
            <button class="nav-tab text-gray-300 hover:nova-text text-lg
            ↪   font-semibold transition-all duration-300"
            ↪   data-section="connect">Connect</button>
            <button class="nav-tab text-gray-300 hover:nova-text text-lg
            ↪   font-semibold transition-all duration-300"
            ↪   data-section="therapy">Therapy</button>
        </div>
    </nav>

    <!-- Main Content -->
    <main class="max-w-7xl mx-auto px-6 py-20 relative">
        <!-- Analyze Section -->
        <section id="analyze" class="section min-h-screen flex flex-col
        ↪   items-center gap-16">
            <div class="text-center">
                <h2 class="text-4xl font-bold nova-text mb-4">Echoes of Your
                ↪   Mind</h2>
                <p class="text-lg text-gray-400 max-w-xl">Decode your emotions
                ↪   through text and voice in a cosmic interface.</p>
            </div>
            <div class="flex flex-col md:flex-row gap-10 w-full">
                <!-- Text Analysis -->
                <div class="card rounded-3xl p-8 flex-1 relative overflow-hidden">
                    <div class="absolute top-0 right-0 w-20 h-20 bg-blue-500/20
                    ↪   rounded-full blur-2xl animate-[float_4s_infinite]"></div>
                    <h3 class="text-2xl font-semibold text-white mb-4">Text
                    ↪   Nebula</h3>
                    <textarea
                        id="text-input"
                        placeholder="Pour your thoughts into the void..."
                        class="w-full h-48 p-4 rounded-xl bg-[#1a1a2e] text-white
↪   border-none focus:ring-2 focus:ring-blue-400/50 transition-all duration-300
↪   resize-none"
                    ></textarea>
                    <button
                        onclick="analyzeText()"
                        class="w-full mt-6 bg-gradient-to-r from-blue-500
↪   to-blue-700 hover:from-blue-600 hover:to-blue-800 text-white py-3 rounded-xl
↪   font-semibold transition-all duration-300 animate-[pulseGlow_3s_infinite]"
                    >
                        Decode Emotions
                    </button>
                    <div id="spinner" class="mx-auto mt-4"></div>
                    <p id="result" class="mt-4 text-gray-300 text-center"></p>
                </div>
                <!-- Voice Analysis -->
                <div class="card rounded-3xl p-8 flex-1 relative overflow-hidden">
                    <div class="absolute bottom-0 left-0 w-20 h-20 bg-purple-500/20
                     ↪  rounded-full blur-2xl animate-[float_5s_infinite]"></div>
                    <h3 class="text-2xl font-semibold text-white mb-4">Sonic
                     ↪  Waves</h3>
                    <div id="waveform" class="waveform hidden my-6 text-center">
                        <div class="bar"></div>
                        <div class="bar"></div>
                        <div class="bar"></div>
                        <div class="bar"></div>
                    </div>
                    <button
                        id="record-btn"
                        onclick="toggleRecording()"
                        class="w-full bg-gradient-to-r from-blue-500 to-purple-600
 ↪  hover:from-blue-600 hover:to-purple-700 text-white py-3 rounded-xl
 ↪  font-semibold transition-all duration-300 flex items-center justify-center
 ↪  gap-2 animate-[pulseGlow_3s_infinite]"
                    >
                        <span>□</span> Capture Voice
                    </button>
                    <div class="flex items-center gap-4 mt-4">
                        <input
                            type="file"
                            id="audio-input"
                            accept="audio/*"
                            class="flex-1 text-gray-400 bg-[#1a1a2e] p-2 rounded-xl
 ↪  file:mr-3 file:py-2 file:px-4 file:rounded-xl file:border-0 file:bg-gray-700
 ↪  file:text-white file:hover:bg-gray-800 transition-all duration-300"
                        >
                        <button
                            onclick="uploadAudio()"
                            class="bg-gradient-to-r from-gray-700 to-gray-600
 ↪  hover:from-gray-800 hover:to-gray-700 text-white py-3 px-6 rounded-xl
 ↪  font-semibold transition-all duration-300 flex items-center gap-2"
                        >
                            <span>□</span> Upload & Analyze
                        </button>
                    </div>
                    <p id="trans_text" class="mt-4 text-gray-300 text-center"></p>
```

```
            </div>
        </div>
    </section>

    <!-- Connect Section -->
    <section id="connect" class="section min-h-screen hidden flex flex-col
     ↪  items-center gap-16">
        <div class="text-center">
            <h2 class="text-4xl font-bold nova-text mb-4">Galactic
             ↪  Connections</h2>
            <p class="text-lg text-gray-400 max-w-xl">Link with a constellation
             ↪  of souls for support.</p>
        </div>
        <div class="card rounded-3xl p-8 w-full max-w-lg relative">
            <div class="absolute top-0 left-0 w-32 h-32 bg-blue-400/10
             ↪  rounded-full blur-3xl animate-[float_6s_infinite]"></div>
            <h3 class="text-2xl font-semibold text-white mb-4">Stellar Hub</h3>
            <input
                id="room-id"
                type="text"
                placeholder="Enter your cosmic ID..."
                class="w-full p-4 rounded-xl bg-[#1a1a2e] text-white
↪  border-none focus:ring-2 focus:ring-purple-400/50 transition-all duration-300"
            />
            <button
                onclick="window.location.href='/meeting-room'"
               class="w-full mt-6 bg-gradient-to-r from-purple-500 to-blue-500
↪  hover:from-purple-600 hover:to-blue-600 text-white py-3 rounded-xl
↪  font-semibold transition-all duration-300 animate-[pulseGlow_3s_infinite]"
            >
                Join the Orbit
            </button>
        </div>
    </section>

    <!-- Therapy Section -->
    <section id="therapy" class="section min-h-screen hidden flex flex-col
     ↪  items-center gap-16">
        <div class="text-center">
            <h2 class="text-4xl font-bold nova-text mb-4">Celestial Calm</h2>
            <p class="text-lg text-gray-400 max-w-xl">Drift into serenity with
             ↪  guided cosmic therapy.</p>
        </div>
```

```
           <div class="card rounded-3xl p-8 w-full max-w-lg relative">
               <div class="absolute bottom-0 right-0 w-32 h-32 bg-purple-400/10
                ↪  rounded-full blur-3xl animate-[float_7s_infinite]"></div>
               <h3 class="text-2xl font-semibold text-white mb-4">Tranquility
                ↪  Portal</h3>
             <p class="text-gray-400 mb-6">Embark on a journey to inner peace.</p>
               <button
                   onclick="window.location.href='/therapy'"
                   class="w-full bg-gradient-to-r from-blue-500 to-purple-600
↪  hover:from-blue-600 hover:to-purple-700 text-white py-3 rounded-xl
↪  font-semibold transition-all duration-300 flex items-center justify-center
↪  gap-2 animate-[pulseGlow_3s_infinite]"
               >
                   <span>□</span> Enter Therapy
               </button>
               <button
                   onclick="window.location.href='/voice-chatbot'"
                   class="w-full bg-gradient-to-r from-blue-500 to-purple-600
↪  hover:from-blue-600 hover:to-purple-700 text-white py-3 rounded-xl
↪  font-semibold transition-all duration-300 flex items-center justify-center
↪  gap-2 animate-[pulseGlow_3s_infinite]"
               >
                   □ Voice Chatbot
               </button>
           </div>
       </section>
   </main>

   <!-- Logout Button -->
   <button
       onclick="window.location.href='/logout'"
       class="fixed bottom-6 right-6 bg-gradient-to-r from-blue-500 to-purple-600
↪  hover:from-blue-600 hover:to-purple-700 text-white py-3 px-6 rounded-full
↪  font-semibold transition-all duration-300 animate-[pulseGlow_2s_infinite]
↪  z-30"
   >
       Logout
   </button>

   <!-- Footer -->
   <footer class="w-full py-10 text-center text-gray-500 bg-[#0f0f1f]/50 relative
    ↪  z-10">
       <p class="nova-text">MindSentry Nova © 2025 – A Cosmic Creation</p>
```

```
</footer>

<script>
    // Section Navigation
   const navTabs = document.querySelectorAll('.nav-tab');
    const sections = document.querySelectorAll('.section');

    navTabs.forEach(tab => {
        tab.addEventListener('click', () => {
            const sectionId = tab.getAttribute('data-section');
            sections.forEach(section => section.classList.add('hidden'));
            document.getElementById(sectionId).classList.remove('hidden');
            navTabs.forEach(t => t.classList.remove('nova-text'));
            tab.classList.add('nova-text');
        });
    });

    // Trigger first section by default
    navTabs[0].click();

    // Parallax Effect
    window.addEventListener('scroll', () => {
        const scrollY = window.scrollY;
        document.querySelectorAll('.card').forEach(card => {
            card.style.transform = `translateY(${scrollY * 0.05}px)`;
        });
    });

    // Scroll to Section from Hero
    function scrollToSection(sectionId) {
        document.getElementById(sectionId).scrollIntoView({ behavior: 'smooth'
         ↪  });
        sections.forEach(section => section.classList.add('hidden'));
        document.getElementById(sectionId).classList.remove('hidden');
        navTabs.forEach(t => t.classList.remove('nova-text'));
        document.querySelector(`[data-
         ↪  section="${sectionId}"]`).classList.add('nova-text');
    }
</script>
<script src="/static/scripts.js"></script>
<script src="/static/meeting-room.js"></script>
```

```
    <script type="module"
    ↪   src="https://unpkg.com/@splinetool/viewer@1.9.82/build/spline-
    ↪   viewer.js"></script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login - MindSentry Nova</title>
    <!-- Tailwind CSS CDN -->
    <script src="https://cdn.tailwindcss.com"></script>
    <!-- Custom styles for enhanced theme and creativity -->
    <style>
        body {
            background: linear-gradient(135deg, #0f0c29, #302b63, #24243e);
            min-height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            margin: 0;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            overflow: hidden;
            position: relative;
        }
        /* Particle animation for background */
        @keyframes float {
            0% { transform: translateY(0) translateX(0); }
            50% { transform: translateY(-20px) translateX(15px); }
            100% { transform: translateY(0) translateX(0); }
        }
        .particle {
            position: absolute;
            background: rgba(0, 191, 255, 0.3);
            border-radius: 50%;
            animation: float 6s ease-in-out infinite;
        }
        .particle-1 { width: 10px; height: 10px; top: 15%; left: 25%;
        ↪   animation-delay: 0s; }
```

```
.particle-2 { width: 15px; height: 15px; top: 40%; left: 75%;
 ↪  animation-delay: 1s; }
.particle-3 { width: 8px; height: 8px; top: 70%; left: 35%;
 ↪  animation-delay: 2s; }
.particle-4 { width: 12px; height: 12px; top: 85%; left: 85%;
 ↪  animation-delay: 3s; }

.card {
    background: rgba(255, 255, 255, 0.05);
    border: 1px solid rgba(0, 191, 255, 0.2);
    border-radius: 20px;
    padding: 2.5rem;
    max-width: 450px;
    width: 90%;
   box-shadow: 0 0 40px rgba(0, 191, 255, 0.3), inset 0 0 10px rgba(0, 191,
    ↪  255, 0.1);
    backdrop-filter: blur(15px);
    text-align: center;
    position: relative;
    overflow: hidden;
    animation: fadeInUp 1.5s ease-out;
    z-index: 1;
}
.card::before {
    content: '';
    position: absolute;
    top: -50%;
    left: -50%;
    width: 200%;
    height: 200%;
    background: radial-gradient(circle, rgba(0, 191, 255, 0.1) 0%,
    ↪  transparent 70%);
    animation: rotate 10s linear infinite;
    z-index: -1;
}
@keyframes rotate {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
@keyframes fadeInUp {
    from { opacity: 0; transform: translateY(50px); }
    to { opacity: 1; transform: translateY(0); }
}
```

```
h1 {
    color: #00BFFF;
    font-size: 2.5rem;
    margin-bottom: 1.5rem;
    text-shadow: 0 0 10px rgba(0, 191, 255, 0.7);
}
p {
    color: #aad2ff;
    margin-bottom: 1.5rem;
}
input[type="text"],
input[type="password"] {
    width: 100%;
    padding: 0.9rem;
    margin: 0.5rem 0;
    border: none;
    border-radius: 12px;
    background: linear-gradient(90deg, #1e1e2f, #2a2a44);
    color: #fff;
    font-size: 1rem;
    outline: none;
    transition: all 0.4s ease;
    box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.5);
}
input[type="text"]:focus,
input[type="password"]:focus {
    background: linear-gradient(90deg, #25253a, #343456);
    box-shadow: 0 0 15px rgba(0, 191, 255, 0.6), inset 0 0 5px rgba(0, 191,
     ↪  255, 0.3);
    transform: scale(1.02);
}
button {
    width: 100%;
    padding: 0.9rem;
    background: linear-gradient(90deg, #00BFFF, #00aaff);
    color: #fff;
    border: none;
    border-radius: 12px;
    font-size: 1.1rem;
    cursor: pointer;
    transition: all 0.4s ease;
    text-transform: uppercase;
    letter-spacing: 1px;
```

```
            box-shadow: 0 5px 15px rgba(0, 191, 255, 0.4);
        }
        button:hover {
            background: linear-gradient(90deg, #00aaff, #0088cc);
            transform: translateY(-2px);
            box-shadow: 0 7px 20px rgba(0, 191, 255, 0.6);
        }
        a {
            color: #00BFFF;
            text-decoration: none;
            font-weight: 600;
            transition: all 0.3s ease;
            position: relative;
        }
        a::after {
            content: '';
            position: absolute;
            width: 0;
            height: 2px;
            bottom: -4px;
            left: 0;
            background: #00BFFF;
            transition: width 0.3s ease;
        }
        a:hover::after {
            width: 100%;
        }
        a:hover {
            color: #00aaff;
        }
        #message {
            margin-top: 1rem;
            color: #aad2ff;
            min-height: 1rem;
            font-size: 0.9rem;
            text-shadow: 0 0 5px rgba(170, 210, 255, 0.5);
        }
    </style>
</head>
<body>
    <!-- Background particles for creative effect -->
    <div class="particle particle-1"></div>
    <div class="particle particle-2"></div>
    <div class="particle particle-3"></div>
    <div class="particle particle-4"></div>

    <div class="card">
        <h1>� MindSentry Nova</h1>
        <p class="text-lg">Welcome back—log in to continue your journey!</p>
        <form action="/login" method="post" class="space-y-6">
            <input type="text" name="username" placeholder="Enter your username"
             ↪  required class="w-full p-4 rounded-xl focus:ring-2
             ↪  focus:ring-cyan-400">
            <input type="password" name="password" placeholder="Enter your
             ↪  password" required class="w-full p-4 rounded-xl focus:ring-2
             ↪  focus:ring-cyan-400">
           <button type="submit" class="w-full py-3 bg-gradient-to-r from-cyan-500
             ↪  to-blue-600 text-white rounded-xl hover:from-cyan-600
             ↪  hover:to-blue-700 transition-all duration-300 transform
             ↪  hover:scale-105">Log In</button>
        </form>
        <p class="mt-4">Don't have an account? <a href="/signup" class="relative
         ↪  text-cyan-300 hover:text-cyan-200">Sign up here</a></p>
        <p id="message"></p>
    </div>

    <script>
        document.querySelector('form').addEventListener('submit', async (e) => {
            e.preventDefault();
            const formData = new FormData(e.target);
            const response = await fetch('/login', {
                method: 'POST',
                body: new URLSearchParams(formData)
            });
            if (response.ok) window.location.href = '/';
            else {
                const data = await response.json();
                document.getElementById('message').innerText = data.detail ||
                 ↪  'Login failed';
            }
        });
    </script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/meeting_room.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Meeting Room</title>
    <link rel="stylesheet" href="/static/styles.css">
    <style>
        .ai-status {
            padding: 8px;
            margin: 8px 0;
            background: rgba(0, 191, 255, 0.1);
            border-radius: 8px;
            display: block;
        }

        .ai-message {
            background: linear-gradient(135deg, #00bfff20, #00bfff40);
            border-left: 3px solid #00bfff;
        }

        .system-message {
            color: #00bfff;
            text-align: center;
            padding: 8px;
            font-style: italic;
        }
    </style>
</head>
<body>
    <!--
    File: meeting_room.html
    Purpose: Provides the UI for the meeting room.

    Flow:
    - Accessed via the /meeting-room endpoint in main.py.
    - Uses meeting-room.js for WebSocket communication.
    -->
    <div class="card">
        <h2>� Meeting Room</h2>
        <p id="room-info"></p>
        <div id="nickname-section">
            <input id="nickname-input" type="text" placeholder="Enter your
             ↪  nickname..." />
            <button onclick="setNickname()">Join Room</button>
        </div>
        <div id="chat-section" style="display: none;">
            <div id="chatbot-prompt" class="system-message" style="display: none;
             ↪  text-align: center;">
                <p>You're alone in the room. Would you like to chat with our AI
                 ↪  chatbot?</p>
                <button onclick="redirectToAiChat()">Chat with Chatbot</button>
                <button onclick="stayInRoom()">Stay Here</button>
            </div>
            <div id="messages" class="chat-box"></div>
            <div class="chat-input">
                <input id="message-input" type="text" placeholder="Type a
                 ↪  message..." onkeypress="handleEnter(event)" />
                <button onclick="sendMessage()">Send</button>
            </div>
            <button class="leave-btn" onclick="leaveRoom()">Leave Room</button>
        </div>
    </div>
    <script src="/static/meeting-room.js"></script>
    <script>
        function handleEnter(event) {
            if (event.key === "Enter") {
                sendMessage();
            }
        }

        // Add AI chat hint to messages div when user is alone
        const originalOnMessage = socket.onmessage;
        socket.onmessage = (event) => {
            const data = JSON.parse(event.data);
            const messages = document.getElementById("messages");

            if (data.type === "system" && data.showAiOption) {
                const aiHint = document.createElement("div");
                aiHint.className = "system-message";
                aiHint.innerHTML = `
                    <div class="ai-hint">
                        ${data.message}<br>
                        <small>Example: @ai How are you today?</small>
                    </div>
                `;
                messages.appendChild(aiHint);
                messages.scrollTop = messages.scrollHeight;

                // Show chatbot prompt
                document.getElementById("chatbot-prompt").style.display = "block";
            } else {
                originalOnMessage(event);
            }
        };

        function redirectToAiChat() {
            window.location.href = "/ai-chat";
        }

        function stayInRoom() {
            document.getElementById("chatbot-prompt").style.display = "none";
        }
    </script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Sign Up - MindSentry Nova</title>
    <!-- Tailwind CSS CDN -->
    <script src="https://cdn.tailwindcss.com"></script>
    <!-- Custom styles for enhanced theme and creativity -->
    <style>
        body {
            background: linear-gradient(135deg, #0f0c29, #302b63, #24243e);
            min-height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            margin: 0;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            overflow: hidden;
            position: relative;
        }
        /* Particle animation for background */
        @keyframes float {
            0% { transform: translateY(0) translateX(0); }
            50% { transform: translateY(-20px) translateX(15px); }
            100% { transform: translateY(0) translateX(0); }
        }
        .particle {
            position: absolute;
            background: rgba(0, 191, 255, 0.3);
            border-radius: 50%;
            animation: float 6s ease-in-out infinite;
        }
        .particle-1 { width: 10px; height: 10px; top: 10%; left: 20%;
         ↪  animation-delay: 0s; }
        .particle-2 { width: 15px; height: 15px; top: 30%; left: 70%;
         ↪  animation-delay: 1s; }
        .particle-3 { width: 8px; height: 8px; top: 60%; left: 30%;
         ↪  animation-delay: 2s; }
        .particle-4 { width: 12px; height: 12px; top: 80%; left: 80%;
         ↪  animation-delay: 3s; }

        .card {
            background: rgba(255, 255, 255, 0.05);
            border: 1px solid rgba(0, 191, 255, 0.2);
            border-radius: 20px;
            padding: 2.5rem;
            max-width: 450px;
            width: 90%;
           box-shadow: 0 0 40px rgba(0, 191, 255, 0.3), inset 0 0 10px rgba(0, 191,
             ↪  255, 0.1);
            backdrop-filter: blur(15px);
            text-align: center;
            position: relative;
            overflow: hidden;
            animation: fadeInUp 1.5s ease-out;
            z-index: 1;
        }
        .card::before {
            content: '';
```

```
        position: absolute;
        top: -50%;
        left: -50%;
        width: 200%;
        height: 200%;
        background: radial-gradient(circle, rgba(0, 191, 255, 0.1) 0%,
         ↪  transparent 70%);
        animation: rotate 10s linear infinite;
        z-index: -1;
    }
    @keyframes rotate {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
    @keyframes fadeInUp {
        from { opacity: 0; transform: translateY(50px); }
        to { opacity: 1; transform: translateY(0); }
    }
    h1 {
        color: #00BFFF;
        font-size: 2.5rem;
        margin-bottom: 1.5rem;
        text-shadow: 0 0 10px rgba(0, 191, 255, 0.7);
    }
    p {
        color: #aad2ff;
        margin-bottom: 1.5rem;
    }
    input[type="text"],
    input[type="password"] {
        width: 100%;
        padding: 0.9rem;
        margin: 0.5rem 0;
        border: none;
        border-radius: 12px;
        background: linear-gradient(90deg, #1e1e2f, #2a2a44);
        color: #fff;
        font-size: 1rem;
        outline: none;
        transition: all 0.4s ease;
        box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.5);
    }
    input[type="text"]:focus,
    input[type="password"]:focus {
        background: linear-gradient(90deg, #25253a, #343456);
        box-shadow: 0 0 15px rgba(0, 191, 255, 0.6), inset 0 0 5px rgba(0, 191,
         ↪  255, 0.3);
        transform: scale(1.02);
    }
    button {
        width: 100%;
        padding: 0.9rem;
        background: linear-gradient(90deg, #00BFFF, #00aaff);
        color: #fff;
        border: none;
        border-radius: 12px;
        font-size: 1.1rem;
        cursor: pointer;
        transition: all 0.4s ease;
        text-transform: uppercase;
        letter-spacing: 1px;
        box-shadow: 0 5px 15px rgba(0, 191, 255, 0.4);
    }
    button:hover {
        background: linear-gradient(90deg, #00aaff, #0088cc);
        transform: translateY(-2px);
        box-shadow: 0 7px 20px rgba(0, 191, 255, 0.6);
    }
    a {
        color: #00BFFF;
        text-decoration: none;
        font-weight: 600;
        transition: all 0.3s ease;
        position: relative;
    }
    a::after {
        content: '';
        position: absolute;
        width: 0;
        height: 2px;
        bottom: -4px;
        left: 0;
        background: #00BFFF;
        transition: width 0.3s ease;
    }
    a:hover::after {
```

```
            width: 100%;
        }
        a:hover {
            color: #00aaff;
        }
        #message {
            margin-top: 1rem;
            color: #aad2ff;
            min-height: 1rem;
            font-size: 0.9rem;
            text-shadow: 0 0 5px rgba(170, 210, 255, 0.5);
        }
    </style>
</head>
<body>
    <!-- Background particles for creative effect -->
    <div class="particle particle-1"></div>
    <div class="particle particle-2"></div>
    <div class="particle particle-3"></div>
    <div class="particle particle-4"></div>

    <div class="card">
        <h1>? MindSentry Nova</h1>
        <p class="text-lg">Embark on your mental wellness journey—sign up now!</p>
        <form action="/signup" method="post" class="space-y-6">
            <input type="text" name="username" placeholder="Enter your username"
             ↪ required class="w-full p-4 rounded-xl focus:ring-2
             ↪ focus:ring-cyan-400">
            <input type="password" name="password" placeholder="Create a password"
             ↪ required class="w-full p-4 rounded-xl focus:ring-2
             ↪ focus:ring-cyan-400">
            <button type="submit" class="w-full py-3 bg-gradient-to-r from-cyan-500
             ↪ to-blue-600 text-white rounded-xl hover:from-cyan-600
             ↪ hover:to-blue-700 transition-all duration-300 transform
             ↪ hover:scale-105">Sign Up Now</button>
        </form>
        <p class="mt-4">Already a member? <a href="/login" class="relative
         ↪ text-cyan-300 hover:text-cyan-200">Log in here</a></p>
        <p id="message"></p>
    </div>

    <script>
        document.querySelector('form').addEventListener('submit', async (e) => {
            e.preventDefault();
            const formData = new FormData(e.target);
            const response = await fetch('/signup', {
                method: 'POST',
                body: new URLSearchParams(formData)
            });
            if (response.ok) window.location.href = '/login';
            else {
                const data = await response.json();
                document.getElementById('message').innerText = data.detail ||
                 ↪  'Signup failed';
            }
        });
    </script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/therapy.html

```
<!--<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Therapy Page</title>
    <link rel="stylesheet" href="/static/styles.css">
</head>
<body>
    <!--
    File: therapy.html
    Purpose: Provides a relaxation page with soothing sounds, calming videos, and a
 ↪  chatbot (coming soon).

    Flow:
    - Accessed via the /therapy endpoint in main.py.
    -->
    <!--<div class="card">
        <h1>□ Therapy Zone</h1>
        <p>Relax with soothing sounds, calming videos, and a chatbot (coming
 ↪  soon).</p>

        <h2>□ Soothing Sounds</h2>
        <ul>
            <li><a href="https://example.com/sound1.mp3" target="_blank">Sound
↪   1</a></li>
            <li><a href="https://example.com/sound2.mp3" target="_blank">Sound
↪   2</a></li>
            <li><a href="https://example.com/sound3.mp3" target="_blank">Sound
↪   3</a></li>
            <li><a href="https://example.com/sound4.mp3" target="_blank">Sound
↪   4</a></li>
            <li><a href="https://example.com/sound5.mp3" target="_blank">Sound
↪   5</a></li>
        </ul>

        <h2>▯ Calming Videos</h2>
        <ul>
            <li><a href="https://example.com/video1.mp4" target="_blank">Video
↪   1</a></li>
            <li><a href="https://example.com/video2.mp4" target="_blank">Video
↪   2</a></li>
            <li><a href="https://example.com/video3.mp4" target="_blank">Video
↪   3</a></li>
            <li><a href="https://example.com/video4.mp4" target="_blank">Video
↪   4</a></li>
            <li><a href="https://example.com/video5.mp4" target="_blank">Video
↪   5</a></li>
        </ul>

        <h2>▯ Chatbot</h2>
        <p>Chatbot functionality coming soon!</p>

        <button onclick="window.location.href='/'">▯ Back to Home</button>
    </div>
</body>
</html>-->



<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Therapy Zone - MindSentry Nova</title>
    <script src="https://cdn.tailwindcss.com"></script>
```

```
<link
↪   href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&family=Poppins:wght
↪   rel="stylesheet">
<link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-
↪   beta3/css/all.min.css" rel="stylesheet">
<style>
    /* Refined Animations */
    @keyframes slideIn {
        from { transform: translateY(30px); opacity: 0; }
        to { transform: translateY(0); opacity: 1; }
    }
    @keyframes subtleLift {
        0%, 100% { transform: translateY(0); }
        50% { transform: translateY(-5px); }
    }
    @keyframes softGlow {
        0%, 100% { box-shadow: 0 0 8px rgba(96, 165, 250, 0.2); }
        50% { box-shadow: 0 0 15px rgba(96, 165, 250, 0.4); }
    }
    @keyframes gentlePulse {
        0%, 100% { transform: scale(1); }
        50% { transform: scale(1.02); }
    }
    /* Global Styles */
    body {
        background: radial-gradient(circle at 20% 20%, #0f0f1f 0%, #1a1a2e 30%,
        ↪   #2a2a4e 100%);
        font-family: 'Poppins', sans-serif;
        overflow-x: hidden;
        position: relative;
    }
    .nova-text {
        font-family: 'Orbitron', sans-serif;
        background: linear-gradient(90deg, #60A5FA, #A78BFA);
        -webkit-background-clip: text;
        background-clip: text;
        color: transparent;
        text-shadow: 0 0 5px rgba(96, 165, 250, 0.1);
    }
    .therapy-card {
        background: linear-gradient(135deg, rgba(255, 255, 255, 0.05), rgba(26,
        ↪   26, 46, 0.2));
        border: 1px solid rgba(96, 165, 250, 0.1);
        backdrop-filter: blur(15px);
        border-radius: 15px;
        padding: 1.5rem;
        position: relative;
        overflow: hidden;
        transition: transform 0.4s ease, box-shadow 0.4s ease;
        animation: slideIn 0.6s ease-out forwards;
        display: flex;
        flex-direction: column;
        justify-content: space-between;
        height: 100%;
    }
    .therapy-card:hover {
        transform: translateY(-8px) scale(1.03);
        box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
        border-color: rgba(96, 165, 250, 0.3);
        animation: gentlePulse 2s infinite ease-in-out;
    }
    .therapy-card::before {
        content: '';
        position: absolute;
        top: -40%;
        left: -40%;
        width: 180%;
        height: 180%;
        background: radial-gradient(circle, rgba(96, 165, 250, 0.1),
         ↪  transparent 70%);
        opacity: 0.3;
        animation: subtleLift 4s infinite ease-in-out;
        pointer-events: none;
        z-index: -1;
    }
    .glow-orb {
        position: absolute;
        width: 80px;
        height: 80px;
        background: radial-gradient(circle, rgba(167, 139, 250, 0.2),
         ↪  transparent 70%);
        border-radius: 50%;
        animation: subtleLift 6s infinite ease-in-out;
        filter: blur(8px);
        opacity: 0.6;
    }
    .section-header {
        position: relative;
        z-index: 1;
    }
    .section-header::after {
        content: '';
        position: absolute;
        bottom: -6px;
        left: 50%;
        transform: translateX(-50%);
        width: 80px;
        height: 2px;
        background: linear-gradient(90deg, #60A5FA, #A78BFA);
        border-radius: 1px;
        animation: softGlow 3s infinite;
    }
    .icon {
        transition: transform 0.3s ease, color 0.3s ease;
    }
    .therapy-card:hover .icon {
        transform: scale(1.1);
        color: #60A5FA;
    }
    /* Media Specific Styles */
    .media-image {
        width: 100%;
        height: 200px;
        background-size: cover;
        background-position: center;
        border-radius: 10px 10px 0 0;
        position: relative;
        overflow: hidden;
        display: flex;
        align-items: center;
        justify-content: center;
    }
    .media-image::before {
        content: '';
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background: radial-gradient(circle, rgba(96, 165, 250, 0.05),
        ↪  transparent);
        opacity: 0.5;
        z-index: 1;
    }
    .media-controls {
        padding: 0.5rem;
        text-align: center;
        flex-grow: 1;
        display: flex;
        flex-direction: column-reverse;
        justify-content: space-between;
    }
    .media-player {
        width: 100%;
        margin-top: 1rem;
        border-radius: 8px;
        background: rgba(0, 0, 0, 0.1);
        padding: 0.5rem;
        outline: none;
    }
    .waveform {
        position: absolute;
        bottom: 10px;
        left: 50%;
        transform: translateX(-50%);
        display: flex;
        gap: 2px;
        z-index: 2;
        opacity: 0;
        transition: opacity 0.3s ease;
    }
    .therapy-card:hover .waveform {
        opacity: 1;
    }
    .wave-bar {
        width: 4px;
        height: 20px;
        background: #60A5FA;
        border-radius: 2px;
        animation: wavePulse 1.5s infinite ease-in-out;
    }
    .wave-bar:nth-child(2) { animation-delay: 0.2s; }
```

```
.wave-bar:nth-child(3) { animation-delay: 0.4s; }
.wave-bar:nth-child(4) { animation-delay: 0.6s; }
.galaxy-swirl {
    position: absolute;
    width: 100%;
    height: 100%;
    background: radial-gradient(circle, rgba(167, 139, 250, 0.05),
     ↪  transparent 70%);
    opacity: 0.3;
    z-index: 1;
    transition: opacity 0.3s ease;
}
.therapy-card:hover .galaxy-swirl {
    opacity: 0.6;
}
.particle {
    position: absolute;
    width: 8px;
    height: 8px;
    background: radial-gradient(circle, #60A5FA, #A78BFA);
    border-radius: 50%;
    animation: subtleLift 8s infinite ease-in-out;
    opacity: 0.4;
}
#video-card {
    width: 600px;
    height: 400px;
    display: flex;
    flex-direction: column-reverse;
}
.video-player {
    width: 100%;
    height: 100%;
    object-fit: cover;
    border-radius: 10px 10px 0 0;
    display: block;
}
.video-controls {
    display: none;
}
.therapy-card:hover .video-controls {
    display: none; /* Keep controls hidden on hover as per requirement */
}
```

```
    </style>
</head>
<body class="text-white">
    <!-- Cosmic Overlay -->
    <div class="fixed inset-0 pointer-events-none z-0">
        <div class="glow-orb top-10 left-10" style="animation-delay: 0s;"></div>
       <div class="glow-orb bottom-20 right-20" style="animation-delay: 2s;"></div>
        <div class="particle top-5 left-5" style="animation-delay: 1s;"></div>
       <div class="particle bottom-10 right-10" style="animation-delay: 3s;"></div>
    </div>

    <!-- Header -->
    <header class="relative py-20 text-center z-10">
        <h1 class="text-5xl md:text-6xl font-bold nova-text
         ↪  animate-[softGlow_2s_infinite]">
            Therapy Zone
        </h1>
        <p class="mt-4 text-xl text-gray-300 max-w-3xl mx-auto">
            A serene retreat powered by cosmic innovation.
        </p>
    </header>

    <!-- Main Content -->
    <main class="max-w-7xl mx-auto px-6 py-16 relative z-10">
        <!-- Soothing Sounds Section -->
        <section class="mb-24">
            <h2 class="section-header text-3xl font-semibold nova-text mb-12
             ↪  text-center">Harmonic Frequencies</h2>
            <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-2 gap-8
             ↪  relative">
                <div class="therapy-card transform translate-x-12 lg:translate-x-24
                 ↪  animate-[subtleLift_5s_infinite]">
                    <div class="media-image" style="background-image:
                     ↪  url('static/images/img1.jpg');">
                        <i class="fas fa-music text-3xl text-blue-400 icon"></i>
                        <div class="waveform">
                            <div class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div>
                        </div>
                    </div>
                    <div class="media-controls">
                        <h3 class="text-xl font-semibold text-white">Echo of
                         ↪  Stillness</h3>
                       <p class="text-gray-400 text-sm mt-2">A gentle resonance to
                         ↪  quiet the mind.</p>
                        <audio id="audio1" class="media-player"
                         ↪  src="static\audio\aud1.mp3" controls></audio>
                    </div>
                </div>
               <div class="therapy-card transform -translate-y-8 lg:-translate-y-16
                 ↪  animate-[subtleLift_5s_infinite] delay-200">
                    <div class="media-image" style="background-image:
                     ↪  url('static/images/img2.jpg');">
                        <i class="fas fa-music text-3xl text-blue-400 icon"></i>
                        <div class="waveform">
                            <div class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div>
                        </div>
                    </div>
                    <div class="media-controls">
                        <h3 class="text-xl font-semibold text-white">Pulse of
                         ↪  Serenity</h3>
                        <p class="text-gray-400 text-sm mt-2">Rhythmic waves for
                         ↪  deep relaxation.</p>
                        <audio id="audio2" class="media-player"
                         ↪  src="static\audio\aud2.mp3" controls></audio>
                    </div>
                </div>
                <div class="therapy-card transform translate-x-16 lg:translate-x-32
                 ↪  animate-[subtleLift_5s_infinite] delay-400">
                    <div class="media-image" style="background-image:
                     ↪  url('static/images/img3.jpg');">
                        <i class="fas fa-music text-3xl text-blue-400 icon"></i>
                        <div class="waveform">
                            <div class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div><div
                             ↪  class="wave-bar"></div>
                        </div>
                    </div>
                    <div class="media-controls">
```

```
                    <h3 class="text-xl font-semibold text-white">Whisper of
                     ↪  Cosmos</h3>
                    <p class="text-gray-400 text-sm mt-2">A celestial hum to
                     ↪  soothe the soul.</p>
                    <audio id="audio3" class="media-player"
                     ↪  src="static\audio\aud3.mp3" controls></audio>
                </div>
            </div>
            <div class="therapy-card transform -translate-y-12
             ↪  lg:-translate-y-24 animate-[subtleLift_5s_infinite]
             ↪  delay-600">
                <div class="media-image" style="background-image:
                 ↪  url('static/images/imgg4.jpg');">
                    <i class="fas fa-music text-3xl text-blue-400 icon"></i>
                    <div class="waveform">
                        <div class="wave-bar"></div><div
                         ↪  class="wave-bar"></div><div
                         ↪  class="wave-bar"></div><div
                         ↪  class="wave-bar"></div>
                    </div>
                </div>
                <div class="media-controls">
                    <h3 class="text-xl font-semibold text-white">Drift of
                     ↪  Silence</h3>
                    <p class="text-gray-400 text-sm mt-2">Soft tones to float
                     ↪  away stress.</p>
                    <audio id="audio4" class="media-player enhanced-audio"
                     ↪  src="static\audio\aud4.mp3" controls
                     ↪  controlslist="nodownload noplaybackrate"></audio>
                </div>
            </div>
    </section>

    <!-- Calming Videos Section -->
    <section class="mb-24">
        <h2 class="section-header text-3xl font-semibold nova-text mb-12
         ↪  text-center">Visual Serenity</h2>
        <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-2 gap-8
         ↪  relative">
           <div class="therapy-card transform -translate-y-8 lg:-translate-y-16
             ↪  animate-[subtleLift_5s_infinite] delay-200" id="video-card">
                <div class="media-image" style="background-image:
                 ↪  url('https://via.placeholder.com/400x200/1a1a2e/60A5FA?text=Flow');">
                    <i class="fas fa-video text-3xl text-purple-400 icon"></i>
                    <div class="galaxy-swirl"></div>
                </div>
                <div class="media-controls">
                    <h3 class="text-xl font-semibold text-white">Stellar
                     ↪  Flow</h3>
                    <p class="text-gray-400 text-sm mt-2">Gentle currents of
                     ↪  peace.</p>
                    <video id="video2" class="media-player"
                        ↪  src="static\video\vid1.mp4" autoplay muted
                        ↪  loop></video>
                </div>
            </div>
            <div class="therapy-card transform translate-x-16 lg:translate-x-32
             ↪  animate-[subtleLift_5s_infinite] delay-400" id="video-card">
                <div class="media-image" style="background-image:
                 ↪  url('https://via.placeholder.com/400x200/1a1a2e/60A5FA?text=Drift');">
                    <i class="fas fa-video text-3xl text-purple-400 icon"></i>
                    <div class="galaxy-swirl"></div>
                </div>
                <div class="media-controls">
                    <h3 class="text-xl font-semibold text-white">Nebula
                     ↪  Drift</h3>
                    <p class="text-gray-400 text-sm mt-2">A dance of serene
                     ↪  lights.</p>
                    <video id="video3" class="media-player"
                        ↪  src="static\video\vid2.mp4" autoplay muted
                        ↪  loop></video>
                </div>
            </div>
            <div class="therapy-card transform -translate-y-12
                ↪  lg:-translate-y-24 animate-[subtleLift_5s_infinite] delay-600"
                ↪  id="video-card">
                <div class="media-image" style="background-image:
                 ↪  url('https://via.placeholder.com/400x200/1a1a2e/60A5FA?text=Serenity');'
                    <i class="fas fa-video text-3xl text-purple-400 icon"></i>
                    <div class="galaxy-swirl"></div>
                </div>
                <div class="media-controls">
                    <h3 class="text-xl font-semibold text-white">Void
                     ↪  Serenity</h3>
                    <p class="text-gray-400 text-sm mt-2">Embrace the quiet
                     ↪  expanse.</p>
```

```
                        <video id="video4" class="media-player"
                         ↪  src="static\video\vid3.mp4" autoplay muted
                         ↪  loop></video>
                    </div>
                </div>
                <div class="therapy-card transform -translate-y-12
                 ↪  lg:-translate-y-24 animate-[subtleLift_5s_infinite] delay-600"
                 ↪  id="video-card">
                    <div class="media-image" style="background-image:
                     ↪  url('https://via.placeholder.com/400x200/1a1a2e/60A5FA?text=Serenity');'
                        <i class="fas fa-video text-3xl text-purple-400 icon"></i>
                        <div class="galaxy-swirl"></div>
                    </div>
                    <div class="media-controls">
                        <h3 class="text-xl font-semibold text-white">Void
                         ↪  Serenity</h3>
                        <p class="text-gray-400 text-sm mt-2">Embrace the quiet
                         ↪  expanse.</p>
                        <video id="video5" class="media-player"
                         ↪  src="static\video\vid4.mp4" autoplay muted
                         ↪  loop></video>
                    </div>
                </div>
            </div>
        </section>

        <!-- Chatbot Section -->
        <section class="mb-24 text-center">
            <h2 class="section-header text-3xl font-semibold nova-text mb-12">AI
             ↪  Companion</h2>
            <div class="therapy-card max-w-lg mx-auto">
                <div class="flex items-center gap-3 mb-2 justify-center">
                    <i class="fas fa-robot text-2xl text-blue-400 icon"></i>
                    <h3 class="text-xl font-semibold text-white">Neural Guide</h3>
                </div>
                <p class="text-gray-300 animate-[softGlow_3s_infinite]">
                    An advanced companion is being crafted. Launching soon.
                </p>
                <br>
                <button class="text-xl font-semibold text-white"
                 ↪  onclick="window.location.href='/ai-chat'">Chatbot</button>
            </div>
        </section>
```

```
            <!-- Back to Home -->
            <div class="text-center">
                <button
                    onclick="window.location.href='/'"
                    class="px-12 py-4 bg-gradient-to-r from-blue-500 to-purple-600
↪   hover:from-blue-600 hover:to-purple-700 text-white rounded-full font-semibold
↪   transition-all duration-300 animate-[softGlow_2s_infinite]"
                >
                    Return to Home
                </button>
            </div>
        </main>

        <!-- Footer -->
        <footer class="w-full py-12 text-center text-gray-500 bg-[#0f0f1f]/70 relative
        ↪   z-10">
            <p class="nova-text text-lg">MindSentry Nova © 2025 – Elevating Emotional
            ↪   Wellness</p>
        </footer>
        <script>
            // Video hover playback
            const videos = document.querySelectorAll('.video-player');
            videos.forEach(video => {
                const card = video.closest('.therapy-card');
                card.addEventListener('mouseenter', () => {
                    video.play();
                });
                card.addEventListener('mouseleave', () => {
                    video.pause();
                    video.currentTime = 0; // Reset to start on next hover
                });
            });
        </script>
</body>
</html>
```

D:/Programming/Simplified_Agentic_AI-Mental-Heatlh-checker/templates/voice_chatbot.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Voice Chatbot</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
        }
        .chatbot-container {
            background: white;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            width: 100%;
            max-width: 500px;
            text-align: center;
        }
        h1 {
            color: #333;
            margin-bottom: 10px;
        }
        .chat-log {
            border: 1px solid #ddd;
            height: 300px;
            overflow-y: auto;
            margin: 10px 0;
            padding: 10px;
            background: #fafafa;
            border-radius: 5px;
        }
        .message {
            margin: 5px 0;
            padding: 8px;
            border-radius: 5px;
            text-align: left;
        }
        .message.user {
            background: #007bff;
            color: white;
            margin-left: 20%;
            margin-right: 5px;
        }
        .message.ai {
            background: #28a745;
            color: white;
            margin-right: 20%;
            margin-left: 5px;
        }
        .controls {
            margin: 10px 0;
        }
        .btn {
            padding: 10px 20px;
            margin: 5px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 16px;
            transition: background-color 0.3s;
        }
        .btn:disabled {
            background: #ccc;
            cursor: not-allowed;
        }
        #start-recording {
            background: #007bff;
            color: white;
        }
        #start-recording:hover:not(:disabled) {
            background: #0056b3;
        }
        #stop-recording {
            background: #dc3545;
            color: white;
        }
        #stop-recording:hover:not(:disabled) {
            background: #b02a37;
        }
        .back-btn {
            background: #6c757d;
            color: white;
        }
        .back-btn:hover {
            background: #5a6268;
        }
        .response {
            margin-top: 10px;
            color: #333;
            font-style: italic;
        }
        #waveform {
            display: none;
            text-align: center;
            margin: 10px 0;
        }
        .wave {
            display: inline-block;
            width: 4px;
            height: 20px;
            margin: 0 2px;
            background: #007bff;
            animation: wave 1s infinite ease-in-out;
        }
        @keyframes wave {
            0%, 100% { transform: scaleY(1); }
            50% { transform: scaleY(2); }
        }
    </style>
</head>
<body>
    <div class="chatbot-container">
        <h1>⍰ Voice Chatbot</h1>
        <p>Interact with our AI assistant using your voice.</p>
        <div id="chat-log" class="chat-log"></div>
        <div id="waveform">
            <div class="wave" style="animation-delay: 0s"></div>
            <div class="wave" style="animation-delay: 0.1s"></div>
            <div class="wave" style="animation-delay: 0.2s"></div>
            <div class="wave" style="animation-delay: 0.3s"></div>
        </div>
        <div class="controls">
            <button id="start-recording" class="btn">⍰ Start Talking</button>
            <button id="stop-recording" class="btn" disabled>□ Stop</button>
        </div>
        <div id="response" class="response"></div>
        <button onclick="window.location.href='/'" class="btn back-btn">□ Back to
         ↪  Home</button>
    </div>
    <script>
        const startButton = document.getElementById("start-recording");
        const stopButton = document.getElementById("stop-recording");
        const chatLog = document.getElementById("chat-log");
        const responseDiv = document.getElementById("response");
        const waveform = document.getElementById("waveform");

        let recognition;
        let isRecording = false;

        // Initialize Speech Recognition
        if ("SpeechRecognition" in window || "webkitSpeechRecognition" in window) {
            recognition = new (window.SpeechRecognition ||
↪   window.webkitSpeechRecognition)();
            recognition.continuous = true;
            recognition.interimResults = true;
            recognition.lang = "en-US";

            recognition.onstart = () => {
                isRecording = true;
                responseDiv.textContent = "Listening...";
                waveform.style.display = "block";
                startButton.disabled = true;
                stopButton.disabled = false;
            };

            recognition.onresult = async (event) => {
                const userMessage = event.results[event.results.length -
                 ↪  1][0].transcript.trim();
                if (userMessage) {
                    addMessageToLog(userMessage, "user");
                    responseDiv.textContent = "Processing...";
                    await sendToChatbot(userMessage);
                }
            };

            recognition.onerror = (event) => {
                console.error(`Recognition error: ${event.error}`);
                responseDiv.textContent = `Error: ${event.error}`;
                if (event.error === "no-speech") {
                    recognition.stop();
                    recognition.start();
                }
            };

            recognition.onend = () => {
                if (isRecording) {
                    recognition.start();
                } else {
                    responseDiv.textContent = "Stopped listening.";
                    waveform.style.display = "none";
                    startButton.disabled = false;
                    stopButton.disabled = true;
                }
            };
        } else {
            responseDiv.textContent = "Speech Recognition is not supported in this
↪   browser.";
            startButton.disabled = true;
            stopButton.disabled = true;
        }

        // Start Recording
        startButton.addEventListener("click", () => {
            if (!isRecording) {
                try {
                    recognition.start();
                } catch (error) {
                    console.error("Error starting recognition:", error);
                    responseDiv.textContent = "Error starting recognition.";
                }
            }
        });

        // Stop Recording
        stopButton.addEventListener("click", () => {
            if (isRecording) {
                isRecording = false;
                recognition.stop();
            }
        });

        // Add message to chat log
        function addMessageToLog(message, sender) {
            const messageDiv = document.createElement("div");
            messageDiv.className = `message ${sender}`;
            messageDiv.textContent = message;
            chatLog.appendChild(messageDiv);
            chatLog.scrollTop = chatLog.scrollHeight;
        }

        // Send message to chatbot API (mock response for demo)
        async function sendToChatbot(message) {
            try {
                // Mock API response for testing
                const mockResponse = {
                  response: `You said: "${message}". How can I assist you further?`
                };

                // Simulate network delay
                await new Promise(resolve => setTimeout(resolve, 1000));

                addMessageToLog(mockResponse.response, "ai");
                responseDiv.textContent = "Ready for next input.";
            } catch (error) {
                console.error("Error sending message to chatbot:", error);
                addMessageToLog("Error: Unable to connect to the server.", "ai");
                responseDiv.textContent = "Error occurred.";
            }
        }
    </script>
</body>
</html>
```