Table of Contents

# AI Mental Health Assistant - Complete Repository Documentation

## Repository Overview

This document contains the complete source code and documentation for the AI Mental Health Assistant project.

**Repository URL:** https://github.com/LKGargProjects/ai-mental-health-assistant  
**Last Updated:** $(date)  
**Branch:** $(git branch –show-current)

## Table of Contents

1. [Project Structure](#project-structure)
2. [Backend (Flask)](#backend-flask)
3. [Frontend (Flutter)](#frontend-flutter)
4. [AI Providers](#ai-providers)
5. [Configuration Files](#configuration-files)
6. [Deployment Scripts](#deployment-scripts)
7. [Mobile Configuration](#mobile-configuration)

## Project Structure

## Backend (Flask)

## app.py

python  
  
from flask import Flask, request, jsonify, session, render\_template  
from flask\_sqlalchemy import SQLAlchemy  
from flask\_session import Session  
from dotenv import load\_dotenv  
import os  
import redis  
from datetime import datetime  
import uuid  
  
# Load environment variables  
load\_dotenv()  
  
from providers.gemini import get\_gemini\_response  
from providers.perplexity import get\_perplexity\_response  
from providers.openai import get\_openai\_response  
from models import db, UserSession, ConversationLog, CrisisEvent  
from crisis\_detection import detect\_crisis\_level  
from flask\_limiter import Limiter  
from flask\_limiter.util import get\_remote\_address  
from flask\_cors import CORS  
  
app = Flask(\_\_name\_\_)  
CORS(app, supports\_credentials=True)  
  
# Enhanced configuration  
app.config['SECRET\_KEY'] = os.environ.get('SECRET\_KEY', 'dev-key-change-in-prod')  
  
# Database configuration with fallback  
database\_url = os.environ.get('DATABASE\_URL')  
if database\_url and database\_url != 'port':  
 # Convert postgresql:// to postgresql+psycopg:// for psycopg3  
 if database\_url.startswith('postgresql://'):  
 database\_url = database\_url.replace('postgresql://', 'postgresql+psycopg://')  
 app.config['SQLALCHEMY\_DATABASE\_URI'] = database\_url  
else:  
 app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///mental\_health.db'  
  
app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False  
  
# Environment-based session configuration  
ENVIRONMENT = os.environ.get('ENVIRONMENT', 'local')  
  
# Try Redis first, fallback to filesystem  
redis\_url = os.environ.get('REDIS\_URL')  
if redis\_url and redis\_url != 'port':  
 try:  
 # Test Redis connection  
 redis\_client = redis.from\_url(redis\_url)  
 redis\_client.ping() # Test connection  
 app.config['SESSION\_TYPE'] = 'redis'  
 app.config['SESSION\_REDIS'] = redis\_client  
 app.logger.info("✅ Redis sessions enabled")  
 except Exception as e:  
 app.logger.warning(f"⚠️ Redis connection failed: {e}, using filesystem sessions")  
 app.config['SESSION\_TYPE'] = 'filesystem'  
 app.config['SESSION\_REDIS'] = None  
else:  
 app.logger.info("ℹ️ No REDIS\_URL found, using filesystem sessions")  
 app.config['SESSION\_TYPE'] = 'filesystem'  
 app.config['SESSION\_REDIS'] = None  
  
app.config['SESSION\_PERMANENT'] = False  
app.config['SESSION\_USE\_SIGNER'] = False # Disable signing for now  
  
# Initialize extensions  
db.init\_app(app)  
Session(app)  
  
# Rate limiting with Redis backend  
limiter = Limiter(  
 key\_func=get\_remote\_address,  
 app=app,  
 default\_limits=["100 per day", "20 per hour"],  
 storage\_uri=os.environ.get('REDIS\_URL', 'memory://')  
)  
  
# Get API keys from environment  
GEMINI\_API\_KEY = os.getenv("GEMINI\_API\_KEY")  
OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")  
PPLX\_API\_KEY = os.getenv("PPLX\_API\_KEY")  
PROVIDER = os.getenv('AI\_PROVIDER', 'gemini')  
  
def get\_or\_create\_session():  
 """Get or create anonymous user session"""  
 # Try to get existing session from Flask session  
 session\_id = session.get('session\_id')  
   
 if not session\_id:  
 # Create new session  
 session\_id = str(uuid.uuid4())  
 session['session\_id'] = session\_id  
   
 # Create new user session in database  
 try:  
 user\_session = UserSession(id=session\_id)  
 db.session.add(user\_session)  
 db.session.commit()  
 app.logger.info(f"✅ Created new session: {session\_id}")  
 except Exception as e:  
 db.session.rollback()  
 app.logger.warning(f"⚠️ Session {session\_id} might already exist: {e}")  
 else:  
 app.logger.info(f"ℹ️ Using existing session: {session\_id}")  
   
 return session\_id  
  
@app.before\_request  
def ensure\_session\_id\_is\_str():  
 """Ensure session\_id is always a string"""  
 session\_id = session.get('session\_id')  
 if isinstance(session\_id, bytes):  
 session['session\_id'] = session\_id.decode('utf-8')  
 app.logger.info("🔄 Converted bytes session\_id to string")  
  
@app.route("/chat", methods=["POST"])  
@limiter.limit("10 per minute")  
def chat():  
 try:  
 data = request.get\_json()  
 if not data or 'message' not in data:  
 return jsonify({"error": "No message provided"}), 400  
  
 message = data['message']  
 mode = data.get('mode', 'mental\_health')  
   
 # Get or create anonymous session  
 session\_id = get\_or\_create\_session()  
   
 # Analyze message for crisis indicators  
 risk\_level, resources = detect\_crisis\_level(message)  
   
 # Get AI response based on provider  
 if PROVIDER == 'openai' and OPENAI\_API\_KEY:  
 response = get\_openai\_response(message, mode)  
 elif PROVIDER == 'gemini' and GEMINI\_API\_KEY:  
 response = get\_gemini\_response(message, mode, session\_id)  
 elif PROVIDER == 'perplexity' and PPLX\_API\_KEY:  
 response = get\_perplexity\_response(message, mode)  
 else:  
 response = "I understand you're sharing something personal. I'm here to listen and support you. Would you like to tell me more about how you're feeling?"  
  
 # Log conversation  
 conversation\_log = ConversationLog(  
 session\_id=session\_id,  
 provider=PROVIDER,  
 risk\_score=risk\_level  
 )  
 db.session.add(conversation\_log)  
   
 # Handle crisis situations  
 response\_data = {  
 "response": response,  
 "risk\_level": risk\_level,  
 "resources": resources,  
 "timestamp": datetime.utcnow().isoformat(),  
 "provider": PROVIDER  
 }  
   
 if risk\_level in ['high', 'medium']:  
 # Log crisis event  
 crisis\_event = CrisisEvent(  
 session\_id=session\_id,  
 risk\_level=risk\_level,  
 intervention\_taken="AI response with resources",  
 escalated=risk\_level == 'high'  
 )  
 db.session.add(crisis\_event)  
   
 # Update session activity  
 user\_session = UserSession.query.get(session\_id)  
 if user\_session:  
 user\_session.last\_active = datetime.utcnow()  
 user\_session.conversation\_count += 1  
 user\_session.risk\_level = risk\_level  
   
 db.session.commit()  
   
 app.logger.info(f"Session: {session\_id}, Provider: {PROVIDER}, Risk: {risk\_level}")  
 return jsonify(response\_data)  
   
 except Exception as e:  
 app.logger.error(f"Error in /chat: {e}", exc\_info=True)  
 return jsonify({"error": f"Error: {str(e)}"}), 500  
  
@app.route("/", methods=["GET"])  
def index():  
 return jsonify({  
 "status": "ok",   
 "message": "AI Mental Health API is running",  
 "provider": PROVIDER,  
 "has\_gemini\_key": bool(GEMINI\_API\_KEY),  
 "has\_openai\_key": bool(OPENAI\_API\_KEY),  
 "has\_perplexity\_key": bool(PPLX\_API\_KEY)  
 })  
  
@app.route("/ping", methods=["GET"])  
def ping():  
 return "pong", 200  
  
@app.route("/health", methods=["GET"])  
def health():  
 return jsonify({"status": "healthy", "timestamp": datetime.utcnow()})  
  
@app.route("/stats", methods=["GET"])  
def stats():  
 return jsonify({  
 "total\_sessions": UserSession.query.count(),  
 "total\_conversations": ConversationLog.query.count(),  
 "crisis\_events": CrisisEvent.query.count()  
 })  
  
@app.route('/get\_or\_create\_session', methods=['GET'])  
def get\_or\_create\_session\_endpoint():  
 session\_id = get\_or\_create\_session()  
 return jsonify({"session\_id": session\_id})  
  
@app.route('/chat\_history', methods=['GET'])  
def get\_chat\_history():  
 session\_id = session.get('session\_id')  
 if not session\_id:  
 return jsonify([])  
   
 conversations = ConversationLog.query.filter\_by(session\_id=session\_id).all()  
 return jsonify([{  
 'id': conv.id,  
 'provider': conv.provider,  
 'risk\_score': conv.risk\_score,  
 'timestamp': conv.timestamp.isoformat() if conv.timestamp else None  
 } for conv in conversations])  
  
@app.route('/mood\_history', methods=['GET'])  
def get\_mood\_history():  
 # For now, return empty list as we haven't implemented mood persistence  
 return jsonify([])  
  
@app.route('/mood\_entry', methods=['POST'])  
def add\_mood\_entry():  
 try:  
 data = request.get\_json()  
 # For now, just echo back the entry as we haven't implemented persistence  
 return jsonify(data)  
 except Exception as e:  
 return jsonify({"error": str(e)}), 400  
  
with app.app\_context():  
 db.create\_all()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 port = int(os.environ.get("PORT", 5050))  
 app.run(host="0.0.0.0", port=port, debug=False)

## models.py

python  
  
  
from flask\_sqlalchemy import SQLAlchemy  
from datetime import datetime  
import uuid  
from cryptography.fernet import Fernet  
import os  
  
db = SQLAlchemy()  
  
class UserSession(db.Model):  
 \_\_tablename\_\_ = 'user\_sessions'  
   
 id = db.Column(db.String(36), primary\_key=True, default=lambda: str(uuid.uuid4()))  
 created\_at = db.Column(db.DateTime, default=datetime.utcnow)  
 last\_active = db.Column(db.DateTime, default=datetime.utcnow)  
 conversation\_count = db.Column(db.Integer, default=0)  
 risk\_level = db.Column(db.String(20), default='low')  
   
class ConversationLog(db.Model):  
 \_\_tablename\_\_ = 'conversation\_logs'  
   
 id = db.Column(db.Integer, primary\_key=True)  
 session\_id = db.Column(db.String(36), db.ForeignKey('user\_sessions.id'))  
 timestamp = db.Column(db.DateTime, default=datetime.utcnow)  
 provider = db.Column(db.String(20))  
 risk\_score = db.Column(db.Float, default=0.0)  
   
class CrisisEvent(db.Model):  
 \_\_tablename\_\_ = 'crisis\_events'  
   
 id = db.Column(db.Integer, primary\_key=True)  
 session\_id = db.Column(db.String(36), db.ForeignKey('user\_sessions.id'))  
 timestamp = db.Column(db.DateTime, default=datetime.utcnow)  
 risk\_level = db.Column(db.String(20))  
 intervention\_taken = db.Column(db.String(100))  
 escalated = db.Column(db.Boolean, default=False)

## crisis\_detection.py

python  
  
from textblob import TextBlob  
import re  
from datetime import datetime  
  
def detect\_crisis\_level(message):  
 """  
 Analyze message for crisis indicators and return risk level and resources.  
 """  
 message = message.lower()  
   
 # Crisis keywords  
 high\_risk\_keywords = ['suicide', 'kill myself', 'want to die', 'end my life']  
 medium\_risk\_keywords = ['hopeless', 'worthless', 'can\'t go on', 'give up']  
 low\_risk\_keywords = ['sad', 'depressed', 'anxious', 'stressed']  
   
 # Check for high risk  
 if any(keyword in message for keyword in high\_risk\_keywords):  
 return 'high', [  
 'National Suicide Prevention Lifeline: 988',  
 'Crisis Text Line: Text HOME to 741741',  
 'Emergency: Call 911'  
 ]  
   
 # Check for medium risk  
 if any(keyword in message for keyword in medium\_risk\_keywords):  
 return 'medium', [  
 'Crisis Text Line: Text HOME to 741741',  
 'Find a Therapist: https://www.psychologytoday.com/us/therapists',  
 'SAMHSA National Helpline: 1-800-662-4357'  
 ]  
   
 # Check for low risk  
 if any(keyword in message for keyword in low\_risk\_keywords):  
 return 'low', [  
 'Find a Therapist: https://www.psychologytoday.com/us/therapists',  
 'Mental Health Resources: https://www.nimh.nih.gov/health'  
 ]  
   
 return 'none', None

## requirements.txt

text  
  
Flask==3.0.0  
Flask-CORS==4.0.0  
Flask-Limiter==3.5.0  
Flask-SQLAlchemy==3.1.1  
Flask-Session==0.5.0  
python-dotenv==1.0.0  
google-generativeai==0.3.2  
openai==1.3.0  
gunicorn==21.2.0  
redis==5.0.1  
psycopg[binary]==3.2.9```  
  
---  
  
## Dockerfile

text

# Use official Python image

FROM python:3.11-slim

# Set working directory

WORKDIR /app

# Copy requirements and install dependencies

COPY requirements.txt . RUN pip install –no-cache-dir -r requirements.txt RUN pip install -U python-dotenv

# Copy project files

COPY . .

# Expose port (Flask default is 5000)

EXPOSE 5000

# Start the app with Gunicorn for production

CMD [“gunicorn”, “-b”, “0.0.0.0:5000”, “app:app”]

---  
  
## docker-compose.yml

yaml

services: web: build: . ports: - “5001:5000” environment: - ENVIRONMENT=docker - DATABASE\_URL=postgresql://postgres:password@db:5432/myapp - REDIS\_URL=redis://redis:6379/0 - SECRET\_KEY=your-secret-key - GEMINI\_API\_KEY=your-gemini-key - PPLX\_API\_KEY=your-pplx-key depends\_on: - db - redis volumes: - .:/app

db: image: postgres:15 restart: always environment: - POSTGRES\_PASSWORD=password - POSTGRES\_DB=myapp volumes: - postgres\_:/var/lib/postgresql/data

redis: image: redis:7 restart: always ports: - “6379:6379”

volumes: postgres\_:

---  
  
## startup.sh

bash

#!/bin/bash # startup.sh - Script to start the Flask application on Render

echo “Starting AI Mental Health API…”

# Install dependencies

pip install -r requirements.txt

# Start the application with gunicorn

exec gunicorn –bind 0.0.0.0:$PORT –workers 1 –threads 8 –timeout 0 app:app ```

## dev\_start.sh

bash  
  
#!/bin/bash  
# Development startup script  
  
echo "🚀 Starting AI Mental Health API in development mode..."  
  
# Activate virtual environment if it exists  
if [ -d "venv" ]; then  
 echo "📦 Activating virtual environment..."  
 source venv/bin/activate  
fi  
  
# Install dependencies if needed  
echo "📦 Installing dependencies..."  
pip install -r requirements.txt  
  
# Start the Flask development server  
echo "🌐 Starting Flask development server..."  
python app.py ```  
  
---  
  
## setup\_local.py

python

#!/usr/bin/env python3 ““” Local Development Setup Script Run this to set up the local development environment ““”

import os import subprocess import sys

def setup\_local\_env(): “““Set up local development environment”“” print(“🚀 Setting up local development environment…”)

# Create .env file for local development  
env\_content = """# Local Development Environment

ENVIRONMENT=local PORT=5050 SECRET\_KEY=dev-secret-key-change-in-prod

# AI Provider Keys (add your actual keys)

GEMINI\_API\_KEY=AIzaSyCsHmnv7YH-gnSbfaVxXrO-xYardOeEiCw OPENAI\_API\_KEY=your\_openai\_api\_key\_here PPLX\_API\_KEY=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 AI\_PROVIDER=gemini

# Local Development (no external services)

# DATABASE\_URL=sqlite:///mental\_health.db

# REDIS\_URL= (leave empty for filesystem sessions)

““”

with open('.env', 'w') as f:  
 f.write(env\_content)  
  
print("✅ Created .env file for local development")  
  
# Install dependencies  
print("📦 Installing Python dependencies...")  
subprocess.run([sys.executable, '-m', 'pip', 'install', '-r', 'requirements.txt'])  
  
print("✅ Local development environment ready!")  
print("\n🎯 To start local development:")  
print(" python app.py")  
print("\n🌐 To start Flutter web app:")  
print(" cd ai\_buddy\_web && flutter run -d chrome")

if **name** == “**main**”: setup\_local\_env() ```

## AI Providers

## providers/gemini.py

python  
  
import os  
import google.generativeai as genai  
from typing import Dict, List  
from datetime import datetime, timedelta  
  
# Store conversations with timestamp for cleanup  
conversations: Dict[str, List[dict]] = {}  
CONVERSATION\_TIMEOUT = timedelta(hours=1) # Clear conversations older than 1 hour  
  
def cleanup\_old\_conversations():  
 """Remove conversations that are older than the timeout"""  
 current\_time = datetime.now()  
 to\_remove = []  
 for session\_id in conversations:  
 if conversations[session\_id]:  
 last\_message\_time = conversations[session\_id][-1].get('timestamp')  
 if last\_message\_time and current\_time - last\_message\_time > CONVERSATION\_TIMEOUT:  
 to\_remove.append(session\_id)  
   
 for session\_id in to\_remove:  
 del conversations[session\_id]  
  
def get\_gemini\_response(message, mode='mental\_health', session\_id=None):  
 """Get response from Gemini API with conversation history"""  
 try:  
 api\_key = os.getenv('GEMINI\_API\_KEY')  
 if not api\_key:  
 print("Gemini API key not found")  
 return "Configuration error: Gemini API key not found"  
  
 # Configure the API  
 genai.configure(api\_key=api\_key)  
   
 # Create the model  
 try:  
# model = genai.GenerativeModel('models/gemini-1.5-flash-latest')  
 model = genai.GenerativeModel('models/gemini-2.5-flash-lite')  
 except Exception as e:  
 print(f"Error creating Gemini model: {str(e)}")  
 return f"Error initializing AI model: {str(e)}"  
   
 # Initialize or get conversation history  
 if session\_id not in conversations:  
 conversations[session\_id] = []  
   
 # Clean up old conversations periodically  
 cleanup\_old\_conversations()  
   
 # Prepare the conversation history  
 history = conversations[session\_id]  
   
 # Prepare the prompt with context  
 system\_message = """You are a supportive AI assistant for high school students.   
 Respond with empathy and understanding. If the user seems distressed,   
 provide emotional support and suggest healthy coping strategies.   
 Keep responses concise and focused."""  
  
 # Build the conversation context  
 conversation\_context = ""  
 if history:  
 conversation\_context = "\n".join([  
 f"{'User' if msg['is\_user'] else 'Assistant'}: {msg['content']}"  
 for msg in history[-5:] # Keep last 5 messages for context  
 ])  
 conversation\_context = f"\nPrevious conversation:\n{conversation\_context}\n"  
  
 prompt = f"{system\_message}\n{conversation\_context}\nUser: {message}"  
   
 # Generate response  
 try:  
 response = model.generate\_content(prompt)  
 if not response or not response.text:  
 print("Empty response from Gemini")  
 return "I received an empty response. Please try again."  
   
 # Store the conversation  
 history.append({  
 'content': message,  
 'is\_user': True,  
 'timestamp': datetime.now()  
 })  
 history.append({  
 'content': response.text,  
 'is\_user': False,  
 'timestamp': datetime.now()  
 })  
 conversations[session\_id] = history  
   
 return response.text  
 except Exception as e:  
 print(f"Error generating content: {str(e)}")  
 return f"Error generating response: {str(e)}"  
  
 except Exception as e:  
 print(f"Unexpected Gemini API error: {str(e)}")  
 return "I'm having trouble connecting to my AI services. Please try again in a moment."

## providers/openai.py

python  
  
import os  
from openai import OpenAI  
  
def get\_openai\_response(message, mode='mental\_health'):  
 """Get response from OpenAI API"""  
 try:  
 client = OpenAI(api\_key=os.getenv('OPENAI\_API\_KEY'))  
   
 system\_message = """You are a supportive AI assistant for high school students.   
 Respond with empathy and understanding. If the user seems distressed,   
 provide emotional support and suggest healthy coping strategies.   
 Keep responses concise and focused."""  
  
 response = client.chat.completions.create(  
 model="gpt-3.5-turbo",  
 messages=[  
 {"role": "system", "content": system\_message},  
 {"role": "user", "content": message}  
 ],  
 max\_tokens=150,  
 temperature=0.7,  
 )  
  
 return response.choices[0].message.content  
  
 except Exception as e:  
 print(f"OpenAI API error: {str(e)}")  
 return "I'm having trouble connecting to my AI services. Please try again in a moment."

## providers/perplexity.py

python  
  
import os  
import requests  
  
def get\_perplexity\_response(message, mode='mental\_health'):  
 """Get response from Perplexity API"""  
 try:  
 api\_key = os.getenv('PERPLEXITY\_API\_KEY')  
 headers = {  
 'Authorization': f'Bearer {api\_key}',  
 'Content-Type': 'application/json',  
 }  
   
 system\_message = """You are a supportive AI assistant for high school students.   
 Respond with empathy and understanding. If the user seems distressed,   
 provide emotional support and suggest healthy coping strategies.   
 Keep responses concise and focused."""  
  
 data = {  
 'model': 'mistral-7b-instruct',  
 'messages': [  
 {'role': 'system', 'content': system\_message},  
 {'role': 'user', 'content': message}  
 ]  
 }  
  
 response = requests.post(  
 'https://api.perplexity.ai/chat/completions',  
 headers=headers,  
 json=data  
 )  
   
 if response.status\_code == 200:  
 return response.json()['choices'][0]['message']['content']  
 else:  
 print(f"Perplexity API error: {response.status\_code} - {response.text}")  
 return "I'm having trouble connecting to my AI services. Please try again in a moment."  
  
 except Exception as e:  
 print(f"Perplexity API error: {str(e)}")  
 return "I'm having trouble connecting to my AI services. Please try again in a moment."

## Frontend (Flutter)

## ai\_buddy\_web/pubspec.yaml

yaml  
  
name: ai\_buddy\_web  
description: "AI-powered mental health and academic assistant"  
# The following line prevents the package from being accidentally published to  
# pub.dev using `flutter pub publish`. This is preferred for private packages.  
publish\_to: 'none' # Remove this line if you wish to publish to pub.dev  
  
# The following defines the version and build number for your application.  
# A version number is three numbers separated by dots, like 1.2.43  
# followed by an optional build number separated by a +.  
# Both the version and the builder number may be overridden in flutter  
# build by specifying --build-name and --build-number, respectively.  
# In Android, build-name is used as versionName while build-number used as versionCode.  
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning  
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion.  
# Read more about iOS versioning at  
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html  
# In Windows, build-name is used as the major, minor, and patch parts  
# of the product and file versions while build-number is used as the build suffix.  
version: 1.0.0+1  
  
environment:  
 sdk: ^3.8.1  
  
# Dependencies specify other packages that your package needs in order to work.  
# To automatically upgrade your package dependencies to the latest versions  
# consider running `flutter pub upgrade --major-versions`. Alternatively,  
# dependencies can be manually updated by changing the version numbers below to  
# the latest version available on pub.dev. To see which dependencies have newer  
# versions available, run `flutter pub outdated`.  
dependencies:  
 flutter:  
 sdk: flutter  
  
 # The following adds the Cupertino Icons font to your application.  
 # Use with the CupertinoIcons class for iOS style icons.  
 cupertino\_icons: ^1.0.8  
 dio: ^5.4.1 # For API calls  
 provider: ^6.1.2 # For state management  
 shared\_preferences: ^2.2.2 # For local storage  
 intl: ^0.19.0 # For date formatting  
 fl\_chart: ^0.66.2 # For mood tracking charts  
 url\_launcher: ^6.2.5 # For opening crisis resource links  
 flutter\_markdown: ^0.6.20 # For rendering markdown in messages  
 flutter\_secure\_storage: ^9.0.0 # For secure storage of session data  
 animated\_text\_kit: ^4.2.2 # For typing animations  
  
dev\_dependencies:  
 flutter\_test:  
 sdk: flutter  
  
 # The "flutter\_lints" package below contains a set of recommended lints to  
 # encourage good coding practices. The lint set provided by the package is  
 # activated in the `analysis\_options.yaml` file located at the root of your  
 # package. See that file for information about deactivating specific lint  
 # rules and activating additional ones.  
 flutter\_lints: ^5.0.0  
  
# For information on the generic Dart part of this file, see the  
# following page: https://dart.dev/tools/pub/pubspec  
  
# The following section is specific to Flutter packages.  
flutter:  
  
 # The following line ensures that the Material Icons font is  
 # included with your application, so that you can use the icons in  
 # the material Icons class.  
 uses-material-design: true  
  
 # To add assets to your application, add an assets section, like this:  
 assets:  
 - assets/images/  
 - assets/icons/  
  
 # An image asset can refer to one or more resolution-specific "variants", see  
 # https://flutter.dev/to/resolution-aware-images  
  
 # For details regarding adding assets from package dependencies, see  
 # https://flutter.dev/to/asset-from-package  
  
 # To add custom fonts to your application, add a fonts section here,  
 # in this "flutter" section. Each entry in this list should have a  
 # "family" key with the font family name, and a "fonts" key with a  
 # list giving the asset and other descriptors for the font. For  
 # example:  
 # fonts:  
 # - family: Schyler  
 # fonts:  
 # - asset: fonts/Schyler-Regular.ttf  
 # - asset: fonts/Schyler-Italic.ttf  
 # style: italic  
 # - family: Trajan Pro  
 # fonts:  
 # - asset: fonts/TrajanPro.ttf  
 # - asset: fonts/TrajanPro\_Bold.ttf  
 # weight: 700  
 #  
 # For details regarding fonts from package dependencies,  
 # see https://flutter.dev/to/font-from-package

## ai\_buddy\_web/lib/main.dart

dart  
  
import 'package:flutter/material.dart';  
import 'package:provider/provider.dart';  
import 'providers/chat\_provider.dart';  
import 'providers/mood\_provider.dart';  
import 'widgets/chat\_message\_widget.dart';  
import 'widgets/mood\_tracker.dart';  
import 'models/message.dart';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return MultiProvider(  
 providers: [  
 ChangeNotifierProvider(create: (\_) => ChatProvider()),  
 ChangeNotifierProvider(create: (\_) => MoodProvider()),  
 ],  
 child: MaterialApp(  
 title: 'AI Mental Health Buddy',  
 debugShowCheckedModeBanner: false,  
 theme: ThemeData(  
 colorScheme: ColorScheme.fromSeed(  
 seedColor: const Color(0xFF667EEA),  
 primary: const Color(0xFF667EEA),  
 secondary: const Color(0xFFFF6B6B),  
 ),  
 useMaterial3: true,  
 ),  
 home: const HomePage(),  
 ),  
 );  
 }  
}  
  
class HomePage extends StatefulWidget {  
 const HomePage({super.key});  
  
 @override  
 State<HomePage> createState() => \_HomePageState();  
}  
  
class \_HomePageState extends State<HomePage> {  
 final TextEditingController \_messageController = TextEditingController();  
 final ScrollController \_scrollController = ScrollController();  
 bool \_showMoodTracker = false;  
  
 @override  
 void dispose() {  
 \_messageController.dispose();  
 \_scrollController.dispose();  
 super.dispose();  
 }  
  
 void \_scrollToBottom() {  
 Future.delayed(const Duration(milliseconds: 100), () {  
 if (\_scrollController.hasClients) {  
 \_scrollController.animateTo(  
 \_scrollController.position.maxScrollExtent,  
 duration: const Duration(milliseconds: 300),  
 curve: Curves.easeOut,  
 );  
 }  
 });  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 backgroundColor: Theme.of(context).colorScheme.inversePrimary,  
 title: const Text('AI Mental Health Buddy'),  
 centerTitle: true,  
 actions: [  
 IconButton(  
 icon: Icon(\_showMoodTracker ? Icons.chat : Icons.mood),  
 onPressed: () {  
 setState(() {  
 \_showMoodTracker = !\_showMoodTracker;  
 });  
 },  
 tooltip: \_showMoodTracker ? 'Show Chat' : 'Show Mood Tracker',  
 ),  
 ],  
 ),  
 body: \_showMoodTracker  
 ? const SingleChildScrollView(  
 padding: EdgeInsets.all(16.0),  
 child: MoodTrackerWidget(),  
 )  
 : Column(  
 children: [  
 // Welcome message  
 Container(  
 width: double.infinity,  
 padding: const EdgeInsets.all(16.0),  
 color: Theme.of(context)  
 .colorScheme  
 .primaryContainer  
 .withOpacity(0.3),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Row(  
 children: [  
 Icon(  
 Icons.favorite,  
 color: Theme.of(context).colorScheme.primary,  
 ),  
 const SizedBox(width: 8),  
 Text(  
 'Welcome to Your Safe Space',  
 style: Theme.of(context).textTheme.titleLarge,  
 ),  
 ],  
 ),  
 const SizedBox(height: 8),  
 Text(  
 'Feel free to share your thoughts and feelings. I\'m here to listen and support you.',  
 style: Theme.of(context).textTheme.bodyLarge,  
 ),  
 ],  
 ),  
 ),  
 // Chat messages  
 Expanded(  
 child: Consumer<ChatProvider>(  
 builder: (context, chatProvider, child) {  
 if (chatProvider.isLoading && chatProvider.messages.isEmpty) {  
 return const Center(child: CircularProgressIndicator());  
 }  
  
 return ListView.builder(  
 controller: \_scrollController,  
 padding: const EdgeInsets.all(8.0),  
 itemCount: chatProvider.messages.length,  
 itemBuilder: (context, index) {  
 return ChatMessageWidget(  
 message: chatProvider.messages[index],  
 );  
 },  
 );  
 },  
 ),  
 ),  
 // Typing indicator  
 Consumer<ChatProvider>(  
 builder: (context, chatProvider, child) {  
 if (!chatProvider.isLoading) return const SizedBox.shrink();  
 return Container(  
 padding: const EdgeInsets.all(8),  
 child: Row(  
 children: [  
 Container(  
 padding: const EdgeInsets.all(12),  
 decoration: BoxDecoration(  
 color: Theme.of(context)  
 .colorScheme  
 .secondaryContainer,  
 borderRadius: BorderRadius.circular(20),  
 ),  
 child: const Text('AI is typing...'),  
 ),  
 ],  
 ),  
 );  
 },  
 ),  
 // Input area  
 Container(  
 decoration: BoxDecoration(  
 color: Theme.of(context).colorScheme.surface,  
 boxShadow: [  
 BoxShadow(  
 offset: const Offset(0, -2),  
 blurRadius: 4,  
 color: Colors.black.withOpacity(0.1),  
 ),  
 ],  
 ),  
 child: Padding(  
 padding: const EdgeInsets.all(8.0),  
 child: Row(  
 children: [  
 Expanded(  
 child: TextField(  
 controller: \_messageController,  
 decoration: InputDecoration(  
 hintText: 'Share your thoughts...',  
 border: OutlineInputBorder(  
 borderRadius: BorderRadius.circular(20),  
 ),  
 contentPadding: const EdgeInsets.symmetric(  
 horizontal: 16,  
 vertical: 12,  
 ),  
 ),  
 onSubmitted: \_handleSubmitted,  
 maxLines: null,  
 textInputAction: TextInputAction.send,  
 ),  
 ),  
 const SizedBox(width: 8),  
 IconButton(  
 onPressed: () =>  
 \_handleSubmitted(\_messageController.text),  
 icon: const Icon(Icons.send),  
 style: IconButton.styleFrom(  
 backgroundColor:  
 Theme.of(context).colorScheme.primary,  
 foregroundColor:  
 Theme.of(context).colorScheme.onPrimary,  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ],  
 ),  
 );  
 }  
  
 void \_handleSubmitted(String text) {  
 if (text.trim().isEmpty) return;  
  
 final chatProvider = Provider.of<ChatProvider>(context, listen: false);  
 chatProvider.sendMessage(text);  
 \_messageController.clear();  
 \_scrollToBottom();  
 }  
}

## ai\_buddy\_web/lib/config/api\_config.dart

dart  
  
import 'dart:io';  
import 'package:flutter/foundation.dart';  
  
class ApiConfig {  
 // Development  
 static const String localUrl = 'http://localhost:5058';  
   
 // Production (Render)  
 static const String productionUrl = 'https://ai-mental-health-assistant.onrender.com';  
   
 // Get the appropriate URL based on environment  
 static String get baseUrl {  
 // For mobile apps, always use production URL  
 if (!kIsWeb) {  
 return productionUrl;  
 }  
   
 // For web, check if we're in production  
 if (Uri.base.host != 'localhost' && Uri.base.host != '127.0.0.1') {  
 return productionUrl;  
 }  
 return localUrl;  
 }  
} ```  
  
---  
  
## ai\_buddy\_web/lib/services/api\_service.dart

dart

import ‘package:dio/dio.dart’; import ‘package:flutter\_secure\_storage/flutter\_secure\_storage.dart’; import ‘../models/message.dart’; import ‘../models/mood\_entry.dart’; import ‘../config/api\_config.dart’;

class ApiService { static String get baseUrl => ApiConfig.baseUrl; final Dio \_dio; final FlutterSecureStorage \_storage;

ApiService() : \_dio = Dio(BaseOptions( baseUrl: baseUrl, headers: { ‘Content-Type’: ‘application/json’, ‘Accept’: ‘application/json’, }, )), \_storage = const FlutterSecureStorage();

Future \_setupSession() async { String? sessionId = await \_storage.read(key: ‘session\_id’); if (sessionId == null) { // Get new session from backend final response = await \_dio.get(‘/get\_or\_create\_session’); sessionId = response.data[‘session\_id’]; await \_storage.write(key: ‘session\_id’, value: sessionId); } // Add session ID to all requests \_dio.options.headers[‘X-Session-ID’] = sessionId; }

Future sendMessage(String content) async { await \_setupSession(); try { final response = await \_dio.post(‘/chat’, data: { ‘message’: content, ‘mode’: ‘mental\_health’, // Always use mental health mode for now });

if (response.data['error'] != null) {  
 throw DioException(  
 requestOptions: RequestOptions(path: '/chat'),  
 error: response.data['error'],  
 );  
 }  
  
 // Extract risk level and resources if present  
 String riskLevel = 'none';  
 List<String>? resources;  
   
 if (response.data['risk\_level'] != null) {  
 riskLevel = response.data['risk\_level'].toString().toLowerCase();  
 }  
   
 if (response.data['resources'] != null) {  
 resources = List<String>.from(response.data['resources']);  
 }  
  
 final message = Message(  
 content: response.data['response'] ?? response.data['message'] ?? 'No response received',  
 isUser: false,  
 riskLevel: RiskLevel.values.firstWhere(  
 (e) => e.toString().split('.').last == riskLevel,  
 orElse: () => RiskLevel.none,  
 ),  
 resources: resources,  
 );  
  
 return message;  
} on DioException catch (e) {  
 print('Error sending message: ${e.message}');  
 print('Error response: ${e.response?.data}');  
 return Message(  
 content: e.response?.data?['error'] ?? 'An error occurred while communicating with the AI. Please try again.',  
 isUser: false,  
 type: MessageType.error,  
 );  
} catch (e) {  
 print('Unexpected error: $e');  
 return Message(  
 content: 'An unexpected error occurred. Please try again.',  
 isUser: false,  
 type: MessageType.error,  
 );  
}

}

Future<List> getMoodHistory() async { await \_setupSession(); try { final response = await \_dio.get(‘/mood\_history’); return (response.data as List) .map((json) => MoodEntry.fromJson(json)) .toList(); } catch (e) { print(‘Error getting mood history: $e’); return []; } }

Future addMoodEntry(MoodEntry entry) async { await \_setupSession(); try { final response = await \_dio.post(‘/mood\_entry’, data: entry.toJson()); return MoodEntry.fromJson(response.data); } on DioException catch (e) { throw Exception(e.response?.data?[‘error’] ?? ‘Failed to save mood entry’); } }

Future<List> getChatHistory() async { await \_setupSession(); try { final response = await \_dio.get(‘/chat\_history’); return (response.data as List) .map((json) => Message.fromJson(json)) .toList(); } catch (e) { print(‘Error getting chat history: $e’); return []; } }

Future clearSession() async { await \_storage.delete(key: ‘session\_id’); } } ```

## ai\_buddy\_web/lib/models/message.dart

dart  
  
import 'package:flutter/material.dart';  
  
enum MessageType { text, error, system }  
enum RiskLevel { none, low, medium, high }  
  
class Message {  
 final String id;  
 final String content;  
 final bool isUser;  
 final DateTime timestamp;  
 final MessageType type;  
 final RiskLevel riskLevel;  
 final List<String>? resources;  
  
 Message({  
 String? id,  
 required this.content,  
 required this.isUser,  
 DateTime? timestamp,  
 this.type = MessageType.text,  
 this.riskLevel = RiskLevel.none,  
 this.resources,  
 }) : id = id ?? DateTime.now().millisecondsSinceEpoch.toString(),  
 timestamp = timestamp ?? DateTime.now();  
  
 factory Message.fromJson(Map<String, dynamic> json) {  
 return Message(  
 id: json['id'] as String?,  
 content: json['content'] as String,  
 isUser: json['is\_user'] as bool,  
 timestamp: json['timestamp'] != null  
 ? DateTime.parse(json['timestamp'] as String)  
 : null,  
 type: MessageType.values.firstWhere(  
 (e) => e.toString() == 'MessageType.${json['type'] ?? 'text'}',  
 orElse: () => MessageType.text,  
 ),  
 riskLevel: RiskLevel.values.firstWhere(  
 (e) => e.toString() == 'RiskLevel.${json['risk\_level'] ?? 'none'}',  
 orElse: () => RiskLevel.none,  
 ),  
 resources: (json['resources'] as List<dynamic>?)?.cast<String>(),  
 );  
 }  
  
 Map<String, dynamic> toJson() {  
 return {  
 'id': id,  
 'content': content,  
 'is\_user': isUser,  
 'timestamp': timestamp.toIso8601String(),  
 'type': type.toString().split('.').last,  
 'risk\_level': riskLevel.toString().split('.').last,  
 'resources': resources,  
 };  
 }  
  
 Color getMessageColor(BuildContext context) {  
 if (type == MessageType.error) {  
 return Theme.of(context).colorScheme.error;  
 }  
 if (type == MessageType.system) {  
 return Theme.of(context).colorScheme.surfaceVariant;  
 }  
 return isUser  
 ? Theme.of(context).colorScheme.primary  
 : Theme.of(context).colorScheme.secondaryContainer;  
 }  
  
 Color getTextColor(BuildContext context) {  
 if (type == MessageType.error) {  
 return Theme.of(context).colorScheme.onError;  
 }  
 if (type == MessageType.system) {  
 return Theme.of(context).colorScheme.onSurfaceVariant;  
 }  
 return isUser  
 ? Theme.of(context).colorScheme.onPrimary  
 : Theme.of(context).colorScheme.onSecondaryContainer;  
 }  
} ```  
  
---  
  
## ai\_buddy\_web/lib/models/mood\_entry.dart

dart

class MoodEntry { final String id; final DateTime timestamp; final int moodLevel; // 1-5: 1=very bad, 5=very good final String? note;

MoodEntry({ String? id, required this.moodLevel, this.note, DateTime? timestamp, }) : id = id ?? DateTime.now().millisecondsSinceEpoch.toString(), timestamp = timestamp ?? DateTime.now(), assert(moodLevel >= 1 && moodLevel <= 5, ‘Mood level must be between 1 and 5’);

factory MoodEntry.fromJson(Map<String, dynamic> json) { return MoodEntry( id: json[‘id’] as String?, moodLevel: json[‘mood\_level’] as int, note: json[‘note’] as String?, timestamp: json[‘timestamp’] != null ? DateTime.parse(json[‘timestamp’] as String) : null, ); }

Map<String, dynamic> toJson() { return { ‘id’: id, ‘mood\_level’: moodLevel, ‘note’: note, ‘timestamp’: timestamp.toIso8601String(), }; }

String get moodEmoji { switch (moodLevel) { case 1: return ‘😢’; case 2: return ‘😕’; case 3: return ‘😐’; case 4: return ‘🙂’; case 5: return ‘😊’; default: return ‘❓’; } }

String get moodDescription { switch (moodLevel) { case 1: return ‘Very Bad’; case 2: return ‘Bad’; case 3: return ‘Okay’; case 4: return ‘Good’; case 5: return ‘Very Good’; default: return ‘Unknown’; } } } ```

## ai\_buddy\_web/lib/providers/chat\_provider.dart

dart  
  
import 'package:flutter/material.dart';  
import '../models/message.dart';  
import '../services/api\_service.dart';  
  
class ChatProvider extends ChangeNotifier {  
 final ApiService \_apiService;  
 final List<Message> \_messages = [];  
 bool \_isLoading = false;  
 String? \_error;  
  
 ChatProvider() : \_apiService = ApiService() {  
 \_loadChatHistory();  
 }  
  
 List<Message> get messages => List.unmodifiable(\_messages);  
 bool get isLoading => \_isLoading;  
 String? get error => \_error;  
  
 Future<void> \_loadChatHistory() async {  
 \_isLoading = true;  
 \_error = null;  
 notifyListeners();  
  
 try {  
 final history = await \_apiService.getChatHistory();  
 \_messages.clear();  
 \_messages.addAll(history);  
 } catch (e) {  
 \_error = 'Failed to load chat history';  
 } finally {  
 \_isLoading = false;  
 notifyListeners();  
 }  
 }  
  
 Future<void> sendMessage(String content) async {  
 if (content.trim().isEmpty) return;  
  
 final userMessage = Message(  
 content: content,  
 isUser: true,  
 );  
  
 \_messages.add(userMessage);  
 \_isLoading = true;  
 \_error = null;  
 notifyListeners();  
  
 try {  
 final aiMessage = await \_apiService.sendMessage(content);  
 \_messages.add(aiMessage);  
 \_error = null;  
 } catch (e) {  
 \_error = 'Failed to send message';  
 \_messages.add(Message(  
 content: 'Failed to get response. Please try again.',  
 isUser: false,  
 type: MessageType.error,  
 ));  
 } finally {  
 \_isLoading = false;  
 notifyListeners();  
 }  
 }  
  
 void clearChat() {  
 \_messages.clear();  
 \_apiService.clearSession();  
 notifyListeners();  
 }  
} ```  
  
---  
  
## ai\_buddy\_web/lib/providers/mood\_provider.dart

dart

import ‘package:flutter/material.dart’; import ‘../models/mood\_entry.dart’; import ‘../services/api\_service.dart’;

class MoodProvider extends ChangeNotifier { final ApiService \_apiService; List \_moodEntries = []; bool \_isLoading = false; String? \_error;

MoodProvider() : \_apiService = ApiService() { \_loadMoodHistory(); }

List get moodEntries => List.unmodifiable(\_moodEntries); bool get isLoading => \_isLoading; String? get error => \_error;

Future \_loadMoodHistory() async { \_isLoading = true; \_error = null; notifyListeners();

try {  
 \_moodEntries = await \_apiService.getMoodHistory();  
 \_error = null;  
} catch (e) {  
 \_error = 'Failed to load mood history';  
} finally {  
 \_isLoading = false;  
 notifyListeners();  
}

}

Future addMoodEntry(int moodLevel, {String? note}) async { \_isLoading = true; \_error = null; notifyListeners();

try {  
 final entry = MoodEntry(  
 moodLevel: moodLevel,  
 note: note,  
 );  
 final savedEntry = await \_apiService.addMoodEntry(entry);  
 \_moodEntries = [...\_moodEntries, savedEntry];  
 \_error = null;  
} catch (e) {  
 \_error = 'Failed to save mood entry';  
} finally {  
 \_isLoading = false;  
 notifyListeners();  
}

}

double get averageMood { if (\_moodEntries.isEmpty) return 0; final sum = \_moodEntries.fold( 0, (sum, entry) => sum + entry.moodLevel, ); return sum / \_moodEntries.length; }

List getMoodEntriesForDate(DateTime date) { return \_moodEntries.where((entry) { return entry.timestamp.year == date.year && entry.timestamp.month == date.month && entry.timestamp.day == date.day; }).toList(); }

Map<DateTime, List> get moodEntriesByDate { final map = <DateTime, List>{}; for (final entry in \_moodEntries) { final date = DateTime( entry.timestamp.year, entry.timestamp.month, entry.timestamp.day, ); map.putIfAbsent(date, () => []).add(entry); } return map; } } ```

## ai\_buddy\_web/lib/widgets/chat\_message\_widget.dart

dart  
  
import 'package:flutter/material.dart';  
import 'package:flutter\_markdown/flutter\_markdown.dart';  
import '../models/message.dart';  
import 'crisis\_resources.dart';  
  
class ChatMessageWidget extends StatelessWidget {  
 final Message message;  
  
 const ChatMessageWidget({  
 super.key,  
 required this.message,  
 });  
  
 @override  
 Widget build(BuildContext context) {  
 return Padding(  
 padding: const EdgeInsets.symmetric(vertical: 4.0, horizontal: 8.0),  
 child: Column(  
 crossAxisAlignment:  
 message.isUser ? CrossAxisAlignment.end : CrossAxisAlignment.start,  
 children: [  
 Row(  
 mainAxisAlignment:  
 message.isUser ? MainAxisAlignment.end : MainAxisAlignment.start,  
 children: [  
 if (!message.isUser) \_buildAvatar(context),  
 const SizedBox(width: 8),  
 Flexible(  
 child: Container(  
 padding:  
 const EdgeInsets.symmetric(horizontal: 16, vertical: 10),  
 decoration: BoxDecoration(  
 color: message.getMessageColor(context),  
 borderRadius: BorderRadius.circular(20),  
 ),  
 child: MarkdownBody(  
 data: message.content,  
 styleSheet: MarkdownStyleSheet(  
 p: TextStyle(color: message.getTextColor(context)),  
 a: TextStyle(  
 color: message.isUser  
 ? Theme.of(context).colorScheme.onPrimary  
 : Theme.of(context).colorScheme.primary,  
 ),  
 ),  
 ),  
 ),  
 ),  
 const SizedBox(width: 8),  
 if (message.isUser) \_buildAvatar(context),  
 ],  
 ),  
 if (message.riskLevel != RiskLevel.none && !message.isUser)  
 Padding(  
 padding: const EdgeInsets.only(top: 8.0),  
 child: CrisisResourcesWidget(riskLevel: message.riskLevel),  
 ),  
 ],  
 ),  
 );  
 }  
  
 Widget \_buildAvatar(BuildContext context) {  
 return CircleAvatar(  
 backgroundColor: message.isUser  
 ? Theme.of(context).colorScheme.primary  
 : Theme.of(context).colorScheme.secondary,  
 child: Text(  
 message.isUser ? '👤' : '🤖',  
 style: const TextStyle(fontSize: 16),  
 ),  
 );  
 }  
} ```  
  
---  
  
## Configuration Files  
## .gitignore

text

# Python

**pycache**/ *.py[cod]* $py.class \*.so .Python venv/ ENV/ .env

# Flutter/Dart

.dart\_tool/ .flutter-plugins .flutter-plugins-dependencies .packages .pub-cache/ .pub/ build/ .metadata \*.iml .idea/ .vscode/

# macOS

.DS\_Store .AppleDouble .LSOverride

# IDE

*.swp* .swo \*~

# Logs

\*.log logs/

# Local development

instance/ .webassets-cache .env.local .env.development.local .env.test.local .env.production.local

# Dependencies

node\_modules/ jspm\_packages/```

## README.md

markdown  
  
---  
  
# AI-MVP-Backend  
  
A modular Flask API backend that lets you access multiple AI providers (Gemini, Perplexity, and more) with a single `/chat` endpoint.  
  
---  
  
## \*\*Features\*\*  
  
- Supports Google Gemini and Perplexity AI providers (easy to extend for OpenAI, Hugging Face, etc.)  
- Simple `/chat` endpoint for unified prompt/response  
- Environment variable-based API key management  
- Modular provider code for easy swapping or extension  
- Logging and error handling included  
  
---  
  
## \*\*Project Structure\*\*

. ├── app.py ├── .env ├── requirements.txt ├── /providers │ ├── **init**.py │ ├── gemini.py │ ├── perplexity.py │ ├── openai.py │ └── huggingface.py └── README.md

---  
  
## \*\*Setup Instructions\*\*  
  
1. \*\*Clone the repository\*\*  
 ```bash  
 git clone <your-repo-url>  
 cd ai-mvp-backend

1. **Create and activate a virtual environment**

* python3 -m venv venv  
  source venv/bin/activate

1. **Install dependencies**

* pip install -r requirements.txt

1. **Configure your .env file**  
   Create a .env file in the project root with your API keys:

* GEMINI\_API\_KEY=your\_gemini\_api\_key\_here  
  PPLX\_API\_KEY=your\_perplexity\_api\_key\_here  
  # Add other keys as needed

1. **Run the Flask app**

* python app.py
* The server will run at <http://127.0.0.1:5000>.

## **Usage**

### **Send a Chat Request**

**Endpoint:**  
POST /chat

**Request Body Example:**

{  
 "prompt": "Hello, AI!",  
 "provider": "gemini"  
}

or

{  
 "prompt": "Hello, AI!",  
 "provider": "perplexity"  
}

**Curl Example:**

curl -X POST http://127.0.0.1:5000/chat \  
 -H "Content-Type: application/json" \  
 -d '{"prompt":"Hello from Gemini!","provider":"gemini"}'

## **Adding More Providers**

* Implement a new provider class in /providers/.
* Update the get\_provider function in app.py to support the new provider.
* Add the required API key to your .env.

## **Logging and Error Handling**

* All requests and responses are logged to the console for easy debugging.
* Errors are returned as JSON with an "error" field and logged for review.

## **License**

None

## **Root Cause**

Somewhere in your code, a bytes object is being stored in the session (most likely session['session\_id']). Flask/werkzeug expects cookie values to be strings, not bytes.

## **How to Fix**

### 1. **Force Session Values to be Strings**

In your get\_or\_create\_session() function in app.py, make sure you always store a string, not bytes:

**Find this code:**

if 'session\_id' not in session:  
 session['session\_id'] = str(uuid.uuid4())  
 # ... rest of code ...

**If you ever decode or encode session values, make sure you use .decode() or .encode() appropriately.**

### 2. **Patch: Always Store as String**

To be extra safe, you can update the assignment to:

session['session\_id'] = str(session['session\_id']) if isinstance(session.get('session\_id'), bytes) else session.get('session\_id', str(uuid.uuid4()))

But the original code should already store a string, so check if anywhere else you are putting a bytes value in the session.

## **Quick Diagnostic**

* Add a debug print right after setting the session:
* print("session\_id type:", type(session['session\_id']))
* Restart your app and try the /chat endpoint again. If you see <class 'bytes'>, something is storing bytes instead of a string.

## **Summary**

* The error is caused by storing a bytes object in the session.
* Make sure all session values (especially session['session\_id']) are always strings.

Would you like me to provide a code patch for your app.py to ensure this?```

## ai\_buddy\_web/README.md

markdown  
  
# ai\_buddy\_web  
  
A new Flutter project.  
  
## Getting Started  
  
This project is a starting point for a Flutter application.  
  
A few resources to get you started if this is your first Flutter project:  
  
- [Lab: Write your first Flutter app](https://docs.flutter.dev/get-started/codelab)  
- [Cookbook: Useful Flutter samples](https://docs.flutter.dev/cookbook)  
  
For help getting started with Flutter development, view the  
[online documentation](https://docs.flutter.dev/), which offers tutorials,  
samples, guidance on mobile development, and a full API reference.

## Deployment Scripts

## ai\_buddy\_web/build\_android.sh

bash  
  
#!/bin/bash  
# Android Build Script for AI Mental Health Buddy  
  
echo "🤖 Building Android APK..."  
  
# Clean previous builds  
flutter clean  
  
# Get dependencies  
flutter pub get  
  
# Build release APK  
flutter build apk --release  
  
echo "✅ Android APK built successfully!"  
echo "📱 APK location: build/app/outputs/flutter-apk/app-release.apk"  
echo ""  
echo "🚀 Next steps:"  
echo "1. Test the APK: flutter install"  
echo "2. Upload to Google Play Console"  
echo "3. Or distribute via direct download" ```  
  
---  
  
## ai\_buddy\_web/build\_ios.sh

bash

#!/bin/bash # iOS Build Script for AI Mental Health Buddy

echo “🍎 Building iOS App…”

# Clean previous builds

flutter clean

# Get dependencies

flutter pub get

# Build release iOS app

flutter build ios –release

echo “✅ iOS app built successfully!” echo “📱 App location: build/ios/iphoneos/Runner.app” echo “” echo “🚀 Next steps:” echo “1. Open ios/Runner.xcworkspace in Xcode” echo “2. Archive the app in Xcode” echo “3. Upload to App Store Connect” echo “4. Or distribute via TestFlight” ```

## setup\_mobile.sh

bash  
  
#!/bin/bash  
# Mobile Development Setup Script  
  
echo "📱 Setting up mobile development environment..."  
  
# Check if Android Studio is installed  
if [ -d "/Applications/Android Studio.app" ]; then  
 echo "✅ Android Studio is installed"  
else  
 echo "❌ Android Studio not found. Please install it from:"  
 echo " https://developer.android.com/studio"  
 echo " Or run: brew install --cask android-studio"  
fi  
  
# Check if Xcode is installed  
if xcode-select -p &> /dev/null; then  
 echo "✅ Xcode command line tools are installed"  
else  
 echo "❌ Xcode command line tools not found. Run:"  
 echo " xcode-select --install"  
fi  
  
# Check CocoaPods  
if command -v pod &> /dev/null; then  
 echo "✅ CocoaPods is installed"  
else  
 echo "❌ CocoaPods not found. Run:"  
 echo " sudo gem install cocoapods"  
fi  
  
echo ""  
echo "🚀 Next steps:"  
echo "1. Open Android Studio and complete the setup wizard"  
echo "2. Install Android SDK through Android Studio"  
echo "3. Run: flutter doctor"  
echo "4. Run: flutter config --android-sdk /path/to/android/sdk"  
echo ""  
echo "📱 To test mobile builds:"  
echo " cd ai\_buddy\_web"  
echo " flutter run -d android # For Android"  
echo " flutter run -d ios # For iOS" ```  
  
---  
  
## Mobile Configuration  
## ai\_buddy\_web/android/app/build.gradle.kts

text

plugins { id(“com.android.application”) id(“kotlin-android”) id(“dev.flutter.flutter-gradle-plugin”) }

val localProperties = Properties() val localPropertiesFile = rootProject.file(“local.properties”) if (localPropertiesFile.exists()) { localPropertiesFile.inputStream().use { localProperties.load(it) } }

val flutterVersionCode = localProperties.getProperty(“flutter.versionCode”) ?: “1” val flutterVersionName = localProperties.getProperty(“flutter.versionName”) ?: “1.0”

android { namespace = “com.example.ai\_buddy\_web” compileSdk = flutter.compileSdkVersion ndkVersion = flutter.ndkVersion

compileOptions {  
 sourceCompatibility = JavaVersion.VERSION\_1\_8  
 targetCompatibility = JavaVersion.VERSION\_1\_8  
}  
  
kotlinOptions {  
 jvmTarget = "1.8"  
}  
  
sourceSets {  
 getByName("main").java.srcDirs("src/main/kotlin")  
}  
  
defaultConfig {  
 applicationId = "com.example.ai\_buddy\_web"  
 minSdk = flutter.minSdkVersion  
 targetSdk = flutter.targetSdkVersion  
 versionCode = flutterVersionCode.toInt()  
 versionName = flutterVersionName  
}  
  
buildTypes {  
 release {  
 // TODO: Add your own signing config for the release build.  
 // Signing with the debug keys for now, so `flutter run --release` works.  
 signingConfig = signingConfigs.getByName("debug")  
 isMinifyEnabled = false  
 proguardFiles(  
 getDefaultProguardFile("proguard-android-optimize.txt"),  
 "proguard-rules.pro"  
 )  
 }  
}

}

flutter { source = “../..” }

dependencies { implementation(“org.jetbrains.kotlin:kotlin-stdlib-jdk7:${kotlin.version}”) }

---  
  
## ai\_buddy\_web/ios/Runner/Info.plist

xml

<!DOCTYPE plist PUBLIC “-//Apple//DTD PLIST 1.0//EN” “http://www.apple.com/DTDs/PropertyList-1.0.dtd”> CFBundleDevelopmentRegion (EXECUTABLE\_NAME) CFBundleIdentifier (FLUTTER\_BUILD\_NAME) CFBundleSignature ???? CFBundleVersion $(FLUTTER\_BUILD\_NUMBER) LSRequiresIPhoneOS UILaunchStoryboardName LaunchScreen UIMainStoryboardFile Main UISupportedInterfaceOrientations UIInterfaceOrientationPortrait UIInterfaceOrientationLandscapeLeft UIInterfaceOrientationLandscapeRight UISupportedInterfaceOrientations~ipad UIInterfaceOrientationPortrait UIInterfaceOrientationPortraitUpsideDown UIInterfaceOrientationLandscapeLeft UIInterfaceOrientationLandscapeRight UIViewControllerBasedStatusBarAppearance CADisableMinimumFrameDurationOnPhone UIApplicationSupportsIndirectInputEvents ```