-e — FILE: ./setup\_local.py — #!/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() -e — FILE: ./models.py —

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)

-e — FILE: ./requirements.txt — 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-e — FILE: ./archive/test\_apis\_v4\_gemini\_refined.py — import requests

def ask\_gemini(prompt, api\_key, model=‘models/gemini-1.5-flash-latest’): url = f”https://generativelanguage.googleapis.com/v1beta/{model}:generateContent?key={api\_key}” headers = {“Content-Type”: “application/json”} payload = { “contents”: [{“parts”: [{“text”: prompt}]}] } response = requests.post(url, headers=headers, json=payload) if response.status\_code == 429: return { “error”: “Quota exceeded. Please wait and try again later.”, “details”: response.text } try: json\_response = response.json() # Extract the main text answer if available if ( ‘candidates’ in json\_response and isinstance(json\_response[‘candidates’], list) and len(json\_response[‘candidates’]) > 0 and ‘content’ in json\_response[‘candidates’][0] and ‘parts’ in json\_response[‘candidates’][0][‘content’] and len(json\_response[‘candidates’][0][‘content’][‘parts’]) > 0 and ‘text’ in json\_response[‘candidates’][0][‘content’][‘parts’][0] ): answer = json\_response[‘candidates’][0][‘content’][‘parts’][0][‘text’] return {“answer”: answer} return json\_response except Exception as e: return { “error”: f”JSON decode error: {e}“,”status\_code”: response.status\_code, “response\_text”: response.text }

if **name** == “**main**”: GEMINI\_API\_KEY = “AIzaSyCsHmnv7YH-gnSbfaVxXrO-xYardOeEiCw” # <— Replace with your valid key from AI Studio prompt = “Summarize the latest AI trends.”

gemini\_result = ask\_gemini(  
 prompt,  
 GEMINI\_API\_KEY,  
 model='models/gemini-1.5-flash-latest'  
)  
  
if "answer" in gemini\_result:  
 print("Gemini AI Answer:\n", gemini\_result["answer"])  
else:  
 print("Gemini API response:", gemini\_result)

-e — FILE: ./archive/test\_apis.py — import requests

# Perplexity API

# Available models: ‘gemini-2’, ‘sonar-pro’, ‘o1’, ‘gpt-4’, etc.

def ask\_perplexity(prompt, api\_key, model=‘gemini-2’): url = “https://api.perplexity.ai/chat/completions” headers = {“Authorization”: f”Bearer {api\_key}“,”Content-Type”: “application/json”} payload = { “model”: model, # Choose model here: ‘gemini-2’, ‘sonar-pro’, ‘o1’, ‘gpt-4’, etc. “messages”: [ {“role”: “user”, “content”: prompt} ] } response = requests.post(url, headers=headers, json=payload) return response.json()

# Gemini API (Google)

# Available models: ‘gemini-pro’, ‘gemini-2.5-pro’, ‘gemini-flash’, etc.

def ask\_gemini(prompt, api\_key, model=‘gemini-pro’): url = f”https://generativelanguage.googleapis.com/v1beta/models/{model}:generateContent” headers = {“Authorization”: f”Bearer {api\_key}“,”Content-Type”: “application/json”} payload = { “contents”: [{“parts”: [{“text”: prompt}]}] } response = requests.post(url, headers=headers, json=payload) return response.json()

# Usage examples:

# To use a different model, just change the ‘model’ argument below.

perplexity\_result = ask\_perplexity( “Summarize the latest AI trends.”, “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”, model=‘sonar-pro’ # Try ‘gemini-2’, ‘o1’, ‘gpt-4’, etc. )

gemini\_result = ask\_gemini( “Summarize the latest AI trends.”, “AIzaSyCsHmnv7YH-gnSbfaVxXrO-xYardOeEiCw”, model=‘gemini-flash’ # Try ‘gemini-pro’, ‘gemini-flash’, ‘gemini-2.5-pro’ etc. )

print(“Perplexity API response:”, perplexity\_result) print(“Gemini API response:”, gemini\_result) -e — FILE: ./archive/test\_apis\_v2\_ppx\_working.py — import requests

# ——— Perplexity API Function ———

def ask\_perplexity(prompt, api\_key, model=‘sonar-pro’): url = “https://api.perplexity.ai/chat/completions” headers = {“Authorization”: f”Bearer {api\_key}“,”Content-Type”: “application/json”} payload = { “model”: model, # Try ‘sonar-pro’, ‘gemini-2’, ‘o1’, ‘gpt-4’, etc. “messages”: [ {“role”: “user”, “content”: prompt} ] } try: response = requests.post(url, headers=headers, json=payload) response.raise\_for\_status() return response.json() except Exception as e: return {“error”: str(e), “response\_text”: getattr(response, “text”, ““)}

# ——— Gemini API Function (with error handling) ———

def ask\_gemini(prompt, api\_key, model=‘models/gemini-1.5-pro-latest’): url = f”https://generativelanguage.googleapis.com/v1beta/{model}:generateContent?key={api\_key}” headers = {“Content-Type”: “application/json”} payload = { “contents”: [{“parts”: [{“text”: prompt}]}] } try: response = requests.post(url, headers=headers, json=payload) response.raise\_for\_status() try: return response.json() except Exception as e: return { “error”: f”JSON decode error: {e}“,”status\_code”: response.status\_code, “response\_text”: response.text } except Exception as e: return { “error”: f”HTTP error: {e}“,”response\_text”: getattr(response, “text”, ““) }

# ——— MAIN TEST ———

if **name** == “**main**”: # Replace with your actual keys! PPLX\_API\_KEY = “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” GEMINI\_API\_KEY = “AIzaSyCsHmnv7YH-gnSbfaVxXrO-xYardOeEiCw”

prompt = "Summarize the latest AI trends."  
  
# Test Perplexity  
perplexity\_result = ask\_perplexity(  
 prompt,  
 PPLX\_API\_KEY,  
 model='sonar-pro' # or 'gemini-2', 'o1', 'gpt-4', etc.  
)  
print("Perplexity API response:")  
print(perplexity\_result)  
print("-" \* 60)  
  
# Test Gemini  
gemini\_result = ask\_gemini(  
 prompt,  
 GEMINI\_API\_KEY,  
 model='models/gemini-1.5-pro-latest' # or 'models/gemini-1.5-flash-latest'  
)  
print("Gemini API response:")  
print(gemini\_result)

-e — FILE: ./archive/test\_apis\_v3\_gemini\_working.py — import requests

# Perplexity API function commented out

’’’ def ask\_perplexity(prompt, api\_key, model=‘sonar-pro’): … ’’’

def ask\_gemini(prompt, api\_key, model=‘models/gemini-1.5-pro-latest’): url = f”https://generativelanguage.googleapis.com/v1beta/{model}:generateContent?key={api\_key}” headers = {“Content-Type”: “application/json”} payload = { “contents”: [{“parts”: [{“text”: prompt}]}] } response = requests.post(url, headers=headers, json=payload) if response.status\_code == 429: return { “error”: “Quota exceeded. Please wait and try again later.”, “details”: response.text } try: return response.json() except Exception as e: return { “error”: f”JSON decode error: {e}“,”status\_code”: response.status\_code, “response\_text”: response.text }

if **name** == “**main**”: GEMINI\_API\_KEY = “AIzaSyCsHmnv7YH-gnSbfaVxXrO-xYardOeEiCw” # <— Replace with your valid key from AI Studio prompt = “Summarize the latest AI trends.”

gemini\_result = ask\_gemini(  
 prompt,  
 GEMINI\_API\_KEY,  
 model='models/gemini-1.5-flash-latest'  
)  
  
print("Gemini API response:", gemini\_result)

-e — FILE: ./archive/checking\_env\_api.py — from dotenv import load\_dotenv import os

load\_dotenv() print(“Gemini Key:”, os.getenv(“GEMINI\_API\_KEY”)) -e — FILE: ./test/widget\_test.dart — // This is a basic Flutter widget test. // // To perform an interaction with a widget in your test, use the WidgetTester // utility in the flutter\_test package. For example, you can send tap and scroll // gestures. You can also use WidgetTester to find child widgets in the widget // tree, read text, and verify that the values of widget properties are correct.

import ‘package:flutter/material.dart’; import ‘package:flutter\_test/flutter\_test.dart’;

import ‘package:ai\_wellness\_buddy/main.dart’;

void main() { testWidgets(‘Counter increments smoke test’, (WidgetTester tester) async { // Build our app and trigger a frame. await tester.pumpWidget(const MyApp());

// Verify that our counter starts at 0.  
expect(find.text('0'), findsOneWidget);  
expect(find.text('1'), findsNothing);  
  
// Tap the '+' icon and trigger a frame.  
await tester.tap(find.byIcon(Icons.add));  
await tester.pump();  
  
// Verify that our counter has incremented.  
expect(find.text('0'), findsNothing);  
expect(find.text('1'), findsOneWidget);

}); } -e — FILE: ./providers/perplexity.py — 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."

-e — FILE: ./providers/gemini.py — 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."

-e — FILE: ./providers/openai.py — 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."

-e — FILE: ./checkpoints/20250727\_121814/perplexity.py — 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."

-e — FILE: ./checkpoints/20250727\_121814/crisis\_detection.py — 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

-e — FILE: ./checkpoints/20250727\_121814/gemini.py — 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."

-e — FILE: ./checkpoints/20250727\_121814/main.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 createState() => \_HomePageState(); }

class \_HomePageState extends State { 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( 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();

} } -e — FILE: ./checkpoints/20250727\_121814/openai.py — 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."

-e — FILE: ./checkpoints/20250727\_121814/api\_service.dart — import ‘package:dio/dio.dart’; import ‘package:flutter\_secure\_storage/flutter\_secure\_storage.dart’; import ‘../models/message.dart’; import ‘../models/mood\_entry.dart’;

class ApiService { static const String baseUrl = ‘http://localhost:5058’; // Updated port to match Flask 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’); } } -e — FILE: ./checkpoints/20250727\_121814/app.py — from flask import Flask, request, jsonify, session, render\_template from flask\_cors import CORS from flask\_limiter import Limiter from flask\_limiter.util import get\_remote\_address from dotenv import load\_dotenv import os import uuid from datetime import datetime from crisis\_detection import detect\_crisis\_level from providers.openai import get\_openai\_response from providers.gemini import get\_gemini\_response from providers.perplexity import get\_perplexity\_response

load\_dotenv()

app = Flask(**name**) CORS(app, supports\_credentials=True) # Enable CORS for all routes

# Configure session

app.secret\_key = os.getenv(‘SECRET\_KEY’, ‘your-secret-key’)

# Configure rate limiting

limiter = Limiter( get\_remote\_address, app=app, default\_limits=[“100 per minute”], storage\_uri=“memory://”, )

# Providers configuration

PROVIDER = os.getenv(‘AI\_PROVIDER’, ‘gemini’) # Default to Gemini OPENAI\_API\_KEY = os.getenv(‘OPENAI\_API\_KEY’) GEMINI\_API\_KEY = os.getenv(‘GEMINI\_API\_KEY’) PERPLEXITY\_API\_KEY = os.getenv(‘PPLX\_API\_KEY’) # Updated to match your env variable name

def get\_mock\_response(message): “““Provide a mock response for testing”“” return “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?”

@app.route(‘/’) def home(): 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(PERPLEXITY\_API\_KEY) })

@app.route(‘/get\_or\_create\_session’, methods=[‘GET’]) def get\_or\_create\_session(): if ‘session\_id’ not in session: session[‘session\_id’] = str(uuid.uuid4()) return jsonify({“session\_id”: session[‘session\_id’]})

@app.route(‘/chat’, methods=[‘POST’]) @limiter.limit(“20 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') # Default to mental health mode  
 session\_id = request.headers.get('X-Session-ID') # Get session ID from header  
  
 print(f"Using provider: {PROVIDER}") # Debug log  
 print(f"Message: {message}") # Debug log  
 print(f"Session ID: {session\_id}") # Debug log  
  
 # Get 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) # Pass session\_id  
 elif PROVIDER == 'perplexity' and PERPLEXITY\_API\_KEY:  
 response = get\_perplexity\_response(message, mode)  
 else:  
 # Use mock response if no valid provider is configured  
 response = get\_mock\_response(message)  
  
 # Detect crisis level  
 risk\_level, resources = detect\_crisis\_level(message)  
  
 return jsonify({  
 "response": response,  
 "risk\_level": risk\_level,  
 "resources": resources,  
 "timestamp": datetime.utcnow().isoformat(),  
 "provider": PROVIDER # Include provider in response for debugging  
 })  
  
except Exception as e:  
 print(f"Error in chat endpoint: {str(e)}") # Debug log  
 return jsonify({  
 "error": "An error occurred while processing your request. Please try again."  
 }), 500

@app.route(‘/chat\_history’, methods=[‘GET’]) def get\_chat\_history(): # For now, return empty list as we haven’t implemented persistence return jsonify([])

@app.route(‘/mood\_history’, methods=[‘GET’]) def get\_mood\_history(): # For now, return empty list as we haven’t implemented 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

if **name** == ‘**main**’: port = int(os.getenv(‘PORT’, 5050)) debug = os.getenv(‘FLASK\_ENV’) == ‘development’ print(f”Starting server with provider: {PROVIDER}“) # Debug log print(f”Gemini API key present: {bool(GEMINI\_API\_KEY)}“) # Debug log app.run(host=‘0.0.0.0’, port=port, debug=debug) -e — FILE: ./crisis\_detection.py — 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

-e — FILE: ./ios/Flutter/ephemeral/flutter\_lldb\_helper.py — # # Generated file, do not edit. #

import lldb

def handle\_new\_rx\_page(frame: lldb.SBFrame, bp\_loc, extra\_args, intern\_dict): “““Intercept NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES and touch the pages.”“” base = frame.register[“x0”].GetValueAsAddress() page\_len = frame.register[“x1”].GetValueAsUnsigned()

# Note: NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES will check contents of the  
# first page to see if handled it correctly. This makes diagnosing  
# misconfiguration (e.g. missing breakpoint) easier.  
data = bytearray(page\_len)  
data[0:8] = b'IHELPED!'  
  
error = lldb.SBError()  
frame.GetThread().GetProcess().WriteMemory(base, data, error)  
if not error.Success():  
 print(f'Failed to write into {base}[+{page\_len}]', error)  
 return

def \_\_lldb\_init\_module(debugger: lldb.SBDebugger, \_): target = debugger.GetDummyTarget() # Caveat: must use BreakpointCreateByRegEx here and not # BreakpointCreateByName. For some reasons callback function does not # get carried over from dummy target for the later. bp = target.BreakpointCreateByRegex(“^NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES$”) bp.SetScriptCallbackFunction(‘{}.handle\_new\_rx\_page’.format(**name**)) bp.SetAutoContinue(True) print(“– LLDB integration loaded –”) -e — FILE: ./pubspec.yaml — name: ai\_wellness\_buddy description: “AI-powered mental health and academic assistant for students” publish\_to: ‘none’ version: 1.0.0+1

environment: sdk: ‘>=3.2.3 <4.0.0’

dependencies: flutter: sdk: flutter cupertino\_icons: ^1.0.2 # State Management provider: ^6.1.1 # HTTP Client dio: ^5.4.0 # Local Storage shared\_preferences: ^2.2.2 # UI Components google\_fonts: ^6.1.0 flutter\_markdown: ^0.6.18 animated\_text\_kit: ^4.2.2 # Animations lottie: ^2.7.0 # Utils url\_launcher: ^6.2.2 share\_plus: ^7.2.1 # Firebase (for future use) firebase\_core: ^2.24.2 firebase\_analytics: ^10.7.4 # PWA Support pwa: ^0.2.12

dev\_dependencies: flutter\_test: sdk: flutter flutter\_lints: ^2.0.0 # Testing mockito: ^5.4.4 # Build flutter\_launcher\_icons: ^0.13.1 flutter\_native\_splash: ^2.3.8

flutter: uses-material-design: true

assets: - assets/images/ - assets/animations/ - assets/icons/

fonts: - family: Inter fonts: - asset: assets/fonts/Inter-Regular.ttf - asset: assets/fonts/Inter-Medium.ttf weight: 500 - asset: assets/fonts/Inter-SemiBold.ttf weight: 600 - asset: assets/fonts/Inter-Bold.ttf weight: 700

flutter\_icons: android: true ios: true image\_path: “assets/icons/app\_icon.png” web: generate: true image\_path: “assets/icons/app\_icon.png” background\_color: “#FFFFFF” theme\_color: “#667eea”

flutter\_native\_splash: color: “#FFFFFF” image: assets/icons/splash\_icon.png branding: assets/icons/branding.png color\_dark: “#1a1a1a” image\_dark: assets/icons/splash\_icon\_dark.png branding\_dark: assets/icons/branding\_dark.png

android\_12: image: assets/icons/splash\_icon.png icon\_background\_color: “#FFFFFF” image\_dark: assets/icons/splash\_icon\_dark.png icon\_background\_color\_dark: “#1a1a1a” -e — FILE: ./wellness\_buddy\_web/test/widget\_test.dart — // This is a basic Flutter widget test. // // To perform an interaction with a widget in your test, use the WidgetTester // utility in the flutter\_test package. For example, you can send tap and scroll // gestures. You can also use WidgetTester to find child widgets in the widget // tree, read text, and verify that the values of widget properties are correct.

import ‘package:flutter/material.dart’; import ‘package:flutter\_test/flutter\_test.dart’;

import ‘package:wellness\_buddy\_web/main.dart’;

void main() { testWidgets(‘Counter increments smoke test’, (WidgetTester tester) async { // Build our app and trigger a frame. await tester.pumpWidget(const MyApp());

// Verify that our counter starts at 0.  
expect(find.text('0'), findsOneWidget);  
expect(find.text('1'), findsNothing);  
  
// Tap the '+' icon and trigger a frame.  
await tester.tap(find.byIcon(Icons.add));  
await tester.pump();  
  
// Verify that our counter has incremented.  
expect(find.text('0'), findsNothing);  
expect(find.text('1'), findsOneWidget);

}); } -e — FILE: ./wellness\_buddy\_web/pubspec.yaml — name: wellness\_buddy\_web description: “A new Flutter project.” # 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

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: # - images/a\_dot\_burr.jpeg # - images/a\_dot\_ham.jpeg

# 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 -e — FILE: ./wellness\_buddy\_web/lib/main.dart — import ‘package:flutter/material.dart’; import ‘package:google\_fonts/google\_fonts.dart’; import ‘package:provider/provider.dart’; import ‘src/providers/chat\_provider.dart’; import ‘src/providers/theme\_provider.dart’; import ‘src/app.dart’;

void main() { WidgetsFlutterBinding.ensureInitialized();

runApp( MultiProvider( providers: [ ChangeNotifierProvider(create: (*) => ThemeProvider()), ChangeNotifierProvider(create: (*) => ChatProvider()), ], child: const WellnessBuddyApp(), ), ); }

class WellnessBuddyApp extends StatelessWidget { const WellnessBuddyApp({super.key});

@override Widget build(BuildContext context) { final themeProvider = Provider.of(context);

return MaterialApp(  
 title: 'AI Wellness & Study Buddy',  
 debugShowCheckedModeBanner: false,  
 themeMode: themeProvider.themeMode,  
 theme: ThemeData(  
 useMaterial3: true,  
 colorScheme: ColorScheme.fromSeed(  
 seedColor: const Color(0xFF667EEA),  
 brightness: Brightness.light,  
 ),  
 textTheme: GoogleFonts.interTextTheme(),  
 ),  
 darkTheme: ThemeData(  
 useMaterial3: true,  
 colorScheme: ColorScheme.fromSeed(  
 seedColor: const Color(0xFF667EEA),  
 brightness: Brightness.dark,  
 ),  
 textTheme: GoogleFonts.interTextTheme(ThemeData.dark().textTheme),  
 ),  
 home: const AppScaffold(),  
);

} } -e — FILE: ./wellness\_buddy\_web/analysis\_options.yaml — # This file configures the analyzer, which statically analyzes Dart code to # check for errors, warnings, and lints. # # The issues identified by the analyzer are surfaced in the UI of Dart-enabled # IDEs (https://dart.dev/tools#ides-and-editors). The analyzer can also be # invoked from the command line by running flutter analyze.

# The following line activates a set of recommended lints for Flutter apps,

# packages, and plugins designed to encourage good coding practices.

include: package:flutter\_lints/flutter.yaml

linter: # The lint rules applied to this project can be customized in the # section below to disable rules from the package:flutter\_lints/flutter.yaml # included above or to enable additional rules. A list of all available lints # and their documentation is published at https://dart.dev/lints. # # Instead of disabling a lint rule for the entire project in the # section below, it can also be suppressed for a single line of code # or a specific dart file by using the // ignore: name\_of\_lint and # // ignore\_for\_file: name\_of\_lint syntax on the line or in the file # producing the lint. rules: # avoid\_print: false # Uncomment to disable the avoid\_print rule # prefer\_single\_quotes: true # Uncomment to enable the prefer\_single\_quotes rule

# Additional information about this file can be found at

# https://dart.dev/guides/language/analysis-options

-e — FILE: ./wellness\_buddy\_web/.dart\_tool/dartpad/web\_plugin\_registrant.dart — // Flutter web plugin registrant file. // // Generated file. Do not edit. //

// ignore\_for\_file: type=lint

void registerPlugins() {} -e — FILE: ./ai\_buddy\_web/test/widget\_test.dart — // This is a basic Flutter widget test. // // To perform an interaction with a widget in your test, use the WidgetTester // utility in the flutter\_test package. For example, you can send tap and scroll // gestures. You can also use WidgetTester to find child widgets in the widget // tree, read text, and verify that the values of widget properties are correct.

import ‘package:flutter/material.dart’; import ‘package:flutter\_test/flutter\_test.dart’;

import ‘package:ai\_buddy\_web/main.dart’;

void main() { testWidgets(‘Counter increments smoke test’, (WidgetTester tester) async { // Build our app and trigger a frame. await tester.pumpWidget(const MyApp());

// Verify that our counter starts at 0.  
expect(find.text('0'), findsOneWidget);  
expect(find.text('1'), findsNothing);  
  
// Tap the '+' icon and trigger a frame.  
await tester.tap(find.byIcon(Icons.add));  
await tester.pump();  
  
// Verify that our counter has incremented.  
expect(find.text('0'), findsNothing);  
expect(find.text('1'), findsOneWidget);

}); } -e — FILE: ./ai\_buddy\_web/ios/Flutter/ephemeral/flutter\_lldb\_helper.py — # # Generated file, do not edit. #

import lldb

def handle\_new\_rx\_page(frame: lldb.SBFrame, bp\_loc, extra\_args, intern\_dict): “““Intercept NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES and touch the pages.”“” base = frame.register[“x0”].GetValueAsAddress() page\_len = frame.register[“x1”].GetValueAsUnsigned()

# Note: NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES will check contents of the  
# first page to see if handled it correctly. This makes diagnosing  
# misconfiguration (e.g. missing breakpoint) easier.  
data = bytearray(page\_len)  
data[0:8] = b'IHELPED!'  
  
error = lldb.SBError()  
frame.GetThread().GetProcess().WriteMemory(base, data, error)  
if not error.Success():  
 print(f'Failed to write into {base}[+{page\_len}]', error)  
 return

def \_\_lldb\_init\_module(debugger: lldb.SBDebugger, \_): target = debugger.GetDummyTarget() # Caveat: must use BreakpointCreateByRegEx here and not # BreakpointCreateByName. For some reasons callback function does not # get carried over from dummy target for the later. bp = target.BreakpointCreateByRegex(“^NOTIFY\_DEBUGGER\_ABOUT\_RX\_PAGES$”) bp.SetScriptCallbackFunction(‘{}.handle\_new\_rx\_page’.format(**name**)) bp.SetAutoContinue(True) print(“– LLDB integration loaded –”) -e — FILE: ./ai\_buddy\_web/pubspec.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 -e — FILE: ./ai\_buddy\_web/lib/config/api\_config.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;

} } -e — FILE: ./ai\_buddy\_web/lib/providers/chat\_provider.dart — import ‘package:flutter/material.dart’; import ‘../models/message.dart’; import ‘../services/api\_service.dart’;

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

ChatProvider() : \_apiService = ApiService() { \_loadChatHistory(); }

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

Future \_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 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(); } } -e — FILE: ./ai\_buddy\_web/lib/providers/mood\_provider.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; } } -e — FILE: ./ai\_buddy\_web/lib/models/message.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? 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[’risk\_level’] ?? ’none’}’, orElse: () => RiskLevel.none, ), resources: (json[‘resources’] as List?)?.cast(), ); }

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; } } -e — FILE: ./ai\_buddy\_web/lib/models/mood\_entry.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’; } } } -e — FILE: ./ai\_buddy\_web/lib/main.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 createState() => \_HomePageState(); }

class \_HomePageState extends State { 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( 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();

} } -e — FILE: ./ai\_buddy\_web/lib/services/api\_service.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’); } } -e — FILE: ./ai\_buddy\_web/lib/widgets/crisis\_resources.dart — import ‘package:flutter/material.dart’; import ‘package:url\_launcher/url\_launcher.dart’; import ‘../models/message.dart’;

class CrisisResourcesWidget extends StatelessWidget { final RiskLevel riskLevel;

const CrisisResourcesWidget({ super.key, required this.riskLevel, });

@override Widget build(BuildContext context) { if (riskLevel == RiskLevel.none) return const SizedBox.shrink();

return Card(  
 color: \_getBackgroundColor(context),  
 child: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Row(  
 children: [  
 Icon(  
 Icons.warning\_rounded,  
 color: \_getIconColor(context),  
 ),  
 const SizedBox(width: 8),  
 Text(  
 \_getTitle(),  
 style: Theme.of(context).textTheme.titleMedium?.copyWith(  
 color: \_getIconColor(context),  
 fontWeight: FontWeight.bold,  
 ),  
 ),  
 ],  
 ),  
 const SizedBox(height: 8),  
 Text(  
 \_getMessage(),  
 style: Theme.of(context).textTheme.bodyMedium,  
 ),  
 const SizedBox(height: 12),  
 Wrap(  
 spacing: 8,  
 runSpacing: 8,  
 children: \_getResources().map((resource) {  
 return ElevatedButton.icon(  
 onPressed: () => \_launchUrl(resource.url),  
 icon: Icon(resource.icon),  
 label: Text(resource.label),  
 style: ElevatedButton.styleFrom(  
 backgroundColor: \_getButtonColor(context),  
 foregroundColor: \_getButtonTextColor(context),  
 ),  
 );  
 }).toList(),  
 ),  
 ],  
 ),  
 ),  
);

}

Color \_getBackgroundColor(BuildContext context) { switch (riskLevel) { case RiskLevel.high: return Theme.of(context).colorScheme.errorContainer; case RiskLevel.medium: return Theme.of(context).colorScheme.secondaryContainer; case RiskLevel.low: return Theme.of(context).colorScheme.surfaceVariant; default: return Theme.of(context).colorScheme.surface; } }

Color \_getIconColor(BuildContext context) { switch (riskLevel) { case RiskLevel.high: return Theme.of(context).colorScheme.error; case RiskLevel.medium: return Theme.of(context).colorScheme.secondary; case RiskLevel.low: return Theme.of(context).colorScheme.onSurfaceVariant; default: return Theme.of(context).colorScheme.onSurface; } }

Color \_getButtonColor(BuildContext context) { switch (riskLevel) { case RiskLevel.high: return Theme.of(context).colorScheme.error; case RiskLevel.medium: return Theme.of(context).colorScheme.secondary; default: return Theme.of(context).colorScheme.primary; } }

Color \_getButtonTextColor(BuildContext context) { switch (riskLevel) { case RiskLevel.high: return Theme.of(context).colorScheme.onError; case RiskLevel.medium: return Theme.of(context).colorScheme.onSecondary; default: return Theme.of(context).colorScheme.onPrimary; } }

String \_getTitle() { switch (riskLevel) { case RiskLevel.high: return ‘Immediate Help Available’; case RiskLevel.medium: return ‘Support Resources’; case RiskLevel.low: return ‘Helpful Resources’; default: return ’’; } }

String \_getMessage() { switch (riskLevel) { case RiskLevel.high: return ‘If you're in crisis, please reach out. Help is available 24/7.’; case RiskLevel.medium: return ‘It sounds like you're going through a difficult time. These resources might help.’; case RiskLevel.low: return ‘Here are some resources that might be helpful.’; default: return ’’; } }

List \_getResources() { final resources = [];

// Add emergency resources for high risk  
if (riskLevel == RiskLevel.high) {  
 resources.addAll([  
 CrisisResource(  
 label: 'Call 988',  
 url: 'tel:988',  
 icon: Icons.phone,  
 ),  
 CrisisResource(  
 label: '988 Lifeline Chat',  
 url: 'https://988lifeline.org/chat/',  
 icon: Icons.chat,  
 ),  
 ]);  
}  
  
// Add general resources  
resources.addAll([  
 CrisisResource(  
 label: 'Crisis Text Line',  
 url: 'sms:741741',  
 icon: Icons.message,  
 ),  
 CrisisResource(  
 label: 'Find a Therapist',  
 url: 'https://www.psychologytoday.com/us/therapists',  
 icon: Icons.person,  
 ),  
]);  
  
return resources;

}

Future \_launchUrl(String url) async { final uri = Uri.parse(url); if (await canLaunchUrl(uri)) { await launchUrl(uri); } } }

class CrisisResource { final String label; final String url; final IconData icon;

const CrisisResource({ required this.label, required this.url, required this.icon, }); } -e — FILE: ./ai\_buddy\_web/lib/widgets/mood\_tracker.dart — import ‘package:flutter/material.dart’; import ‘package:fl\_chart/fl\_chart.dart’; import ‘package:intl/intl.dart’; import ‘package:provider/provider.dart’; import ‘../models/mood\_entry.dart’; import ‘../providers/mood\_provider.dart’;

class MoodTrackerWidget extends StatelessWidget { const MoodTrackerWidget({super.key});

@override Widget build(BuildContext context) { return Consumer( builder: (context, moodProvider, child) { if (moodProvider.isLoading) { return const Center(child: CircularProgressIndicator()); }

if (moodProvider.error != null) {  
 return Center(  
 child: Text(  
 moodProvider.error!,  
 style: TextStyle(color: Theme.of(context).colorScheme.error),  
 ),  
 );  
 }  
  
 return Column(  
 children: [  
 \_buildMoodInput(context, moodProvider),  
 const SizedBox(height: 16),  
 if (moodProvider.moodEntries.isNotEmpty) ...[  
 \_buildMoodChart(context, moodProvider),  
 const SizedBox(height: 16),  
 \_buildMoodStats(context, moodProvider),  
 ],  
 ],  
 );  
 },  
);

}

Widget \_buildMoodInput(BuildContext context, MoodProvider moodProvider) { return Card( child: Padding( padding: const EdgeInsets.all(16.0), child: Column( crossAxisAlignment: CrossAxisAlignment.start, children: [ Text( ‘How are you feeling?’, style: Theme.of(context).textTheme.titleMedium, ), const SizedBox(height: 12), Row( mainAxisAlignment: MainAxisAlignment.spaceEvenly, children: List.generate(5, (index) { final moodLevel = index + 1; final entry = MoodEntry(moodLevel: moodLevel); return IconButton( onPressed: () => \_showMoodDialog(context, moodProvider, moodLevel), icon: Text( entry.moodEmoji, style: const TextStyle(fontSize: 24), ), tooltip: entry.moodDescription, ); }), ), ], ), ), ); }

Widget \_buildMoodChart(BuildContext context, MoodProvider moodProvider) { final entries = moodProvider.moodEntries; if (entries.isEmpty) return const SizedBox.shrink();

return SizedBox(  
 height: 200,  
 child: LineChart(  
 LineChartData(  
 gridData: const FlGridData(show: false),  
 titlesData: FlTitlesData(  
 leftTitles: AxisTitles(  
 sideTitles: SideTitles(  
 showTitles: true,  
 interval: 1,  
 reservedSize: 40,  
 getTitlesWidget: (value, meta) {  
 if (value < 1 || value > 5) return const Text('');  
 return Text(MoodEntry(moodLevel: value.toInt()).moodEmoji);  
 },  
 ),  
 ),  
 bottomTitles: AxisTitles(  
 sideTitles: SideTitles(  
 showTitles: true,  
 interval: 1,  
 getTitlesWidget: (value, meta) {  
 if (value >= entries.length) return const Text('');  
 final date = entries[value.toInt()].timestamp;  
 return Text(DateFormat('MM/dd').format(date));  
 },  
 ),  
 ),  
 rightTitles: const AxisTitles(  
 sideTitles: SideTitles(showTitles: false),  
 ),  
 topTitles: const AxisTitles(  
 sideTitles: SideTitles(showTitles: false),  
 ),  
 ),  
 borderData: FlBorderData(show: false),  
 minX: 0,  
 maxX: entries.length.toDouble() - 1,  
 minY: 1,  
 maxY: 5,  
 lineBarsData: [  
 LineChartBarData(  
 spots: entries.asMap().entries.map((entry) {  
 return FlSpot(  
 entry.key.toDouble(),  
 entry.value.moodLevel.toDouble(),  
 );  
 }).toList(),  
 isCurved: true,  
 color: Theme.of(context).colorScheme.primary,  
 barWidth: 3,  
 isStrokeCapRound: true,  
 dotData: const FlDotData(show: true),  
 belowBarData: BarAreaData(  
 show: true,  
 color: Theme.of(context).colorScheme.primary.withOpacity(0.1),  
 ),  
 ),  
 ],  
 ),  
 ),  
);

}

Widget \_buildMoodStats(BuildContext context, MoodProvider moodProvider) { final entries = moodProvider.moodEntries; if (entries.isEmpty) return const SizedBox.shrink();

final averageMood = moodProvider.averageMood;  
final latestMood = entries.last;  
  
return Card(  
 child: Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Text(  
 'Mood Stats',  
 style: Theme.of(context).textTheme.titleMedium,  
 ),  
 const SizedBox(height: 8),  
 Row(  
 mainAxisAlignment: MainAxisAlignment.spaceAround,  
 children: [  
 \_buildStatItem(  
 context,  
 'Current Mood',  
 latestMood.moodEmoji,  
 latestMood.moodDescription,  
 ),  
 \_buildStatItem(  
 context,  
 'Average Mood',  
 MoodEntry(moodLevel: averageMood.round()).moodEmoji,  
 averageMood.toStringAsFixed(1),  
 ),  
 \_buildStatItem(  
 context,  
 'Total Entries',  
 '📊',  
 entries.length.toString(),  
 ),  
 ],  
 ),  
 ],  
 ),  
 ),  
);

}

Widget \_buildStatItem( BuildContext context, String label, String emoji, String value, ) { return Column( children: [ Text( label, style: Theme.of(context).textTheme.bodySmall, ), const SizedBox(height: 4), Text( emoji, style: const TextStyle(fontSize: 24), ), const SizedBox(height: 4), Text( value, style: Theme.of(context).textTheme.bodyMedium, ), ], ); }

Future \_showMoodDialog( BuildContext context, MoodProvider moodProvider, int moodLevel, ) async { final noteController = TextEditingController();

return showDialog(  
 context: context,  
 builder: (context) => AlertDialog(  
 title: Row(  
 children: [  
 Text(  
 MoodEntry(moodLevel: moodLevel).moodEmoji,  
 style: const TextStyle(fontSize: 24),  
 ),  
 const SizedBox(width: 8),  
 Text('Feeling ${MoodEntry(moodLevel: moodLevel).moodDescription}'),  
 ],  
 ),  
 content: TextField(  
 controller: noteController,  
 decoration: const InputDecoration(  
 labelText: 'Add a note (optional)',  
 hintText: 'What made you feel this way?',  
 ),  
 maxLines: 3,  
 ),  
 actions: [  
 TextButton(  
 onPressed: () => Navigator.of(context).pop(),  
 child: const Text('Cancel'),  
 ),  
 FilledButton(  
 onPressed: () {  
 moodProvider.addMoodEntry(  
 moodLevel,  
 note: noteController.text.trim(),  
 );  
 Navigator.of(context).pop();  
 },  
 child: const Text('Save'),  
 ),  
 ],  
 ),  
);

} } -e — FILE: ./ai\_buddy\_web/lib/widgets/chat\_message\_widget.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), ), ); } } -e — FILE: ./ai\_buddy\_web/analysis\_options.yaml — # This file configures the analyzer, which statically analyzes Dart code to # check for errors, warnings, and lints. # # The issues identified by the analyzer are surfaced in the UI of Dart-enabled # IDEs (https://dart.dev/tools#ides-and-editors). The analyzer can also be # invoked from the command line by running flutter analyze.

# The following line activates a set of recommended lints for Flutter apps,

# packages, and plugins designed to encourage good coding practices.

include: package:flutter\_lints/flutter.yaml

linter: # The lint rules applied to this project can be customized in the # section below to disable rules from the package:flutter\_lints/flutter.yaml # included above or to enable additional rules. A list of all available lints # and their documentation is published at https://dart.dev/lints. # # Instead of disabling a lint rule for the entire project in the # section below, it can also be suppressed for a single line of code # or a specific dart file by using the // ignore: name\_of\_lint and # // ignore\_for\_file: name\_of\_lint syntax on the line or in the file # producing the lint. rules: # avoid\_print: false # Uncomment to disable the avoid\_print rule # prefer\_single\_quotes: true # Uncomment to enable the prefer\_single\_quotes rule

# Additional information about this file can be found at

# https://dart.dev/guides/language/analysis-options

-e — FILE: ./ai\_buddy\_web/.dart\_tool/dartpad/web\_plugin\_registrant.dart — // Flutter web plugin registrant file. // // Generated file. Do not edit. //

// @dart = 2.13 // ignore\_for\_file: type=lint

import ‘package:flutter\_secure\_storage\_web/flutter\_secure\_storage\_web.dart’; import ‘package:shared\_preferences\_web/shared\_preferences\_web.dart’; import ‘package:url\_launcher\_web/url\_launcher\_web.dart’; import ‘package:flutter\_web\_plugins/flutter\_web\_plugins.dart’;

void registerPlugins([final Registrar? pluginRegistrar]) { final Registrar registrar = pluginRegistrar ?? webPluginRegistrar; FlutterSecureStorageWeb.registerWith(registrar); SharedPreferencesPlugin.registerWith(registrar); UrlLauncherPlugin.registerWith(registrar); registrar.registerMessageHandler(); } -e — FILE: ./linux/CMakeLists.txt — # Project-level configuration. cmake\_minimum\_required(VERSION 3.13) project(runner LANGUAGES CXX)

# The name of the executable created for the application. Change this to change

# the on-disk name of your application.

set(BINARY\_NAME “ai\_wellness\_buddy”) # The unique GTK application identifier for this application. See: # https://wiki.gnome.org/HowDoI/ChooseApplicationID set(APPLICATION\_ID “com.example.ai\_wellness\_buddy”)

# Explicitly opt in to modern CMake behaviors to avoid warnings with recent

# versions of CMake.

cmake\_policy(SET CMP0063 NEW)

# Load bundled libraries from the lib/ directory relative to the binary.

set(CMAKE\_INSTALL\_RPATH “$ORIGIN/lib”)

# Root filesystem for cross-building.

if(FLUTTER\_TARGET\_PLATFORM\_SYSROOT) set(CMAKE\_SYSROOT ${FLUTTER\_TARGET\_PLATFORM\_SYSROOT}) set(CMAKE\_FIND\_ROOT\_PATH ${CMAKE\_SYSROOT}) set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM NEVER) set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PACKAGE ONLY) set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY) set(CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY) endif()

# Define build configuration options.

if(NOT CMAKE\_BUILD\_TYPE AND NOT CMAKE\_CONFIGURATION\_TYPES) set(CMAKE\_BUILD\_TYPE “Debug” CACHE STRING “Flutter build mode” FORCE) set\_property(CACHE CMAKE\_BUILD\_TYPE PROPERTY STRINGS “Debug” “Profile” “Release”) endif()

# Compilation settings that should be applied to most targets.

# Be cautious about adding new options here, as plugins use this function by

# default. In most cases, you should add new options to specific targets instead

# of modifying this function.

function(APPLY\_STANDARD\_SETTINGS TARGET) target\_compile\_features({TARGET} PRIVATE -Wall -Werror) target\_compile\_options(${TARGET} PRIVATE "$<>:-O3>“) target\_compile\_definitions(${TARGET} PRIVATE "$<>:NDEBUG>”) endfunction()

# Flutter library and tool build rules.

set(FLUTTER\_MANAGED\_DIR “${CMAKE\_CURRENT\_SOURCE\_DIR}/flutter")
add\_subdirectory(${FLUTTER\_MANAGED\_DIR})

# System-level dependencies.

find\_package(PkgConfig REQUIRED) pkg\_check\_modules(GTK REQUIRED IMPORTED\_TARGET gtk+-3.0)

# Application build; see runner/CMakeLists.txt.

add\_subdirectory(“runner”)

# Run the Flutter tool portions of the build. This must not be removed.

add\_dependencies(${BINARY\_NAME} flutter\_assemble)

# Only the install-generated bundle’s copy of the executable will launch

# correctly, since the resources must in the right relative locations. To avoid

# people trying to run the unbundled copy, put it in a subdirectory instead of

# the default top-level location.

set\_target\_properties(${BINARY\_NAME}
PROPERTIES
RUNTIME\_OUTPUT\_DIRECTORY "${CMAKE\_BINARY\_DIR}/intermediates\_do\_not\_run” )

# Generated plugin build rules, which manage building the plugins and adding

# them to the application.

include(flutter/generated\_plugins.cmake)

# === Installation ===

# By default, “installing” just makes a relocatable bundle in the build

# directory.

set(BUILD\_BUNDLE\_DIR “${PROJECT\_BINARY\_DIR}/bundle")
if(CMAKE\_INSTALL\_PREFIX\_INITIALIZED\_TO\_DEFAULT)
set(CMAKE\_INSTALL\_PREFIX "${BUILD\_BUNDLE\_DIR}” CACHE PATH “…” FORCE) endif()

# Start with a clean build bundle directory every time.

install(CODE ” file(REMOVE\_RECURSE "${BUILD\_BUNDLE\_DIR}/") ” COMPONENT Runtime)

set(INSTALL\_BUNDLE\_DATA\_DIR “${CMAKE\_INSTALL\_PREFIX}/data")
set(INSTALL\_BUNDLE\_LIB\_DIR "${CMAKE\_INSTALL\_PREFIX}/lib”)

install(TARGETS ${BINARY\_NAME} RUNTIME DESTINATION "${CMAKE\_INSTALL\_PREFIX}” COMPONENT Runtime)

install(FILES “${FLUTTER\_ICU\_DATA\_FILE}" DESTINATION "${INSTALL\_BUNDLE\_DATA\_DIR}” COMPONENT Runtime)

install(FILES “${FLUTTER\_LIBRARY}" DESTINATION "${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime)

foreach(bundled\_library ${PLUGIN\_BUNDLED\_LIBRARIES})
install(FILES "${bundled\_library}” DESTINATION “${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime) endforeach(bundled\_library)

# Copy the native assets provided by the build.dart from all packages.

set(NATIVE\_ASSETS\_DIR “${PROJECT\_BUILD\_DIR}native\_assets/linux/")
install(DIRECTORY "${NATIVE\_ASSETS\_DIR}” DESTINATION “${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime)

# Fully re-copy the assets directory on each build to avoid having stale files

# from a previous install.

set(FLUTTER\_ASSET\_DIR\_NAME “flutter\_assets”) install(CODE ” file(REMOVE\_RECURSE "{FLUTTER\_ASSET\_DIR\_NAME}") ” COMPONENT Runtime) install(DIRECTORY “{FLUTTER\_ASSET\_DIR\_NAME}” DESTINATION “${INSTALL\_BUNDLE\_DATA\_DIR}” COMPONENT Runtime)

# Install the AOT library on non-Debug builds only.

if(NOT CMAKE\_BUILD\_TYPE MATCHES “Debug”) install(FILES “${AOT\_LIBRARY}" DESTINATION "${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime) endif() -e — FILE: ./linux/runner/CMakeLists.txt — cmake\_minimum\_required(VERSION 3.13) project(runner LANGUAGES CXX)

# Define the application target. To change its name, change BINARY\_NAME in the

# top-level CMakeLists.txt, not the value here, or flutter run will no longer

# work.

# Any new source files that you add to the application should be added here.

add\_executable(${BINARY\_NAME}
"main.cc"
"my\_application.cc"
"${FLUTTER\_MANAGED\_DIR}/generated\_plugin\_registrant.cc” )

# Apply the standard set of build settings. This can be removed for applications

# that need different build settings.

apply\_standard\_settings(${BINARY\_NAME})

# Add preprocessor definitions for the application ID.

add\_definitions(-DAPPLICATION\_ID=“${APPLICATION\_ID}”)

# Add dependency libraries. Add any application-specific dependencies here.

target\_link\_libraries({BINARY\_NAME} PRIVATE PkgConfig::GTK)

target\_include\_directories(${BINARY\_NAME} PRIVATE "${CMAKE\_SOURCE\_DIR}“) -e — FILE: ./linux/flutter/CMakeLists.txt — # This file controls Flutter-level build steps. It should not be edited. cmake\_minimum\_required(VERSION 3.10)

set(EPHEMERAL\_DIR “${CMAKE\_CURRENT\_SOURCE\_DIR}/ephemeral”)

# Configuration provided via flutter tool.

include(${EPHEMERAL\_DIR}/generated\_config.cmake)

# TODO: Move the rest of this into files in ephemeral. See

# https://github.com/flutter/flutter/issues/57146.

# Serves the same purpose as list(TRANSFORM … PREPEND …),

# which isn’t available in 3.10.

function(list\_prepend LIST\_NAME PREFIX) set(NEW\_LIST ““) foreach(element ${${LIST\_NAME}}) list(APPEND NEW\_LIST”{element}“) endforeach(element) set(${LIST\_NAME} "${NEW\_LIST}” PARENT\_SCOPE) endfunction()

# === Flutter Library ===

# System-level dependencies.

find\_package(PkgConfig REQUIRED) pkg\_check\_modules(GTK REQUIRED IMPORTED\_TARGET gtk+-3.0) pkg\_check\_modules(GLIB REQUIRED IMPORTED\_TARGET glib-2.0) pkg\_check\_modules(GIO REQUIRED IMPORTED\_TARGET gio-2.0)

set(FLUTTER\_LIBRARY “${EPHEMERAL\_DIR}/libflutter\_linux\_gtk.so”)

# Published to parent scope for install step.

set(FLUTTER\_LIBRARY ${FLUTTER\_LIBRARY} PARENT\_SCOPE)
set(FLUTTER\_ICU\_DATA\_FILE "${EPHEMERAL\_DIR}/icudtl.dat” PARENT\_SCOPE) set(PROJECT\_BUILD\_DIR “${PROJECT\_DIR}/build/" PARENT\_SCOPE)
set(AOT\_LIBRARY "${PROJECT\_DIR}/build/lib/libapp.so” PARENT\_SCOPE)

list(APPEND FLUTTER\_LIBRARY\_HEADERS “fl\_basic\_message\_channel.h” “fl\_binary\_codec.h” “fl\_binary\_messenger.h” “fl\_dart\_project.h” “fl\_engine.h” “fl\_json\_message\_codec.h” “fl\_json\_method\_codec.h” “fl\_message\_codec.h” “fl\_method\_call.h” “fl\_method\_channel.h” “fl\_method\_codec.h” “fl\_method\_response.h” “fl\_plugin\_registrar.h” “fl\_plugin\_registry.h” “fl\_standard\_message\_codec.h” “fl\_standard\_method\_codec.h” “fl\_string\_codec.h” “fl\_value.h” “fl\_view.h” “flutter\_linux.h” ) list\_prepend(FLUTTER\_LIBRARY\_HEADERS “${EPHEMERAL\_DIR}/flutter\_linux/")
add\_library(flutter INTERFACE)
target\_include\_directories(flutter INTERFACE
"${EPHEMERAL\_DIR}” ) target\_link\_libraries(flutter INTERFACE “${FLUTTER\_LIBRARY}”) target\_link\_libraries(flutter INTERFACE PkgConfig::GTK PkgConfig::GLIB PkgConfig::GIO ) add\_dependencies(flutter flutter\_assemble)

# === Flutter tool backend ===

# *phony* is a non-existent file to force this command to run every time,

# since currently there’s no way to get a full input/output list from the

# flutter tool.

add\_custom\_command( OUTPUT ${FLUTTER\_LIBRARY} ${FLUTTER\_LIBRARY\_HEADERS} ${CMAKE\_CURRENT\_BINARY\_DIR}/*phony* COMMAND ${CMAKE\_COMMAND} -E env ${FLUTTER\_TOOL\_ENVIRONMENT}
"${FLUTTER\_ROOT}/packages/flutter\_tools/bin/tool\_backend.sh” ${FLUTTER\_TARGET\_PLATFORM} ${CMAKE\_BUILD\_TYPE}
VERBATIM
)
add\_custom\_target(flutter\_assemble DEPENDS
"${FLUTTER\_LIBRARY}” ${FLUTTER\_LIBRARY\_HEADERS} ) -e — FILE: ./app.py — 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) -e — FILE: ./lib/main.dart — import ‘package:flutter/material.dart’;

void main() { runApp(const MyApp()); }

class MyApp extends StatelessWidget { const MyApp({super.key});

@override Widget build(BuildContext context) { return MaterialApp( title: ‘AI Wellness & Study Buddy’, debugShowCheckedModeBanner: false, theme: ThemeData( colorScheme: ColorScheme.fromSeed(seedColor: const Color(0xFF667EEA)), useMaterial3: true, ), home: const MyHomePage(), ); } }

class MyHomePage extends StatelessWidget { const MyHomePage({super.key});

@override Widget build(BuildContext context) { return Scaffold( appBar: AppBar( backgroundColor: Theme.of(context).colorScheme.inversePrimary, title: const Text(‘AI Wellness & Study Buddy’), ), body: Center( child: Column( mainAxisAlignment: MainAxisAlignment.center, children: [ const Text( ‘Welcome to AI Wellness & Study Buddy!’, style: TextStyle(fontSize: 24), textAlign: TextAlign.center, ), const SizedBox(height: 20), const Text( ‘Choose your mode:’, style: TextStyle(fontSize: 18), ), const SizedBox(height: 20), Row( mainAxisAlignment: MainAxisAlignment.center, children: [ ElevatedButton.icon( onPressed: () {}, icon: const Text(‘❤️’, style: TextStyle(fontSize: 24)), label: const Text(‘Mental Health’), ), const SizedBox(width: 20), ElevatedButton.icon( onPressed: () {}, icon: const Text(‘📚’, style: TextStyle(fontSize: 24)), label: const Text(‘Academic Help’), ), ], ), ], ), ), ); } } -e — FILE: ./lib/src/providers/chat\_provider.dart — import ‘package:flutter/material.dart’; import ‘package:dio/dio.dart’; import ‘../models/message.dart’; import ‘../models/chat\_mode.dart’;

class ChatProvider with ChangeNotifier { final List \_messages = []; final Dio \_dio = Dio(); bool \_isLoading = false; ChatMode \_currentMode = ChatMode.mentalHealth; String? \_error;

List get messages => List.unmodifiable(\_messages); bool get isLoading => \_isLoading; ChatMode get currentMode => \_currentMode; String? get error => \_error;

void setMode(ChatMode mode) { if (\_currentMode != mode) { \_currentMode = mode; notifyListeners(); } }

Future sendMessage(String content) async { if (content.trim().isEmpty) return;

final userMessage = Message(  
 content: content,  
 isUser: true,  
 timestamp: DateTime.now(),  
);  
  
\_messages.add(userMessage);  
\_isLoading = true;  
\_error = null;  
notifyListeners();  
  
try {  
 final response = await \_dio.post(  
 'https://ai-mental-health-assistant-tddc.onrender.com/chat',  
 data: {  
 'prompt': content,  
 'provider': 'gemini',  
 'mode': \_currentMode.name,  
 },  
 );  
  
 if (response.statusCode == 200) {  
 final aiMessage = Message(  
 content: response.data['answer'],  
 isUser: false,  
 timestamp: DateTime.now(),  
 riskLevel: response.data['risk\_level'],  
 resources: response.data['crisis\_resources'],  
 );  
  
 \_messages.add(aiMessage);  
 } else {  
 \_error = 'Failed to get response from AI';  
 }  
} catch (e) {  
 \_error = 'Error: ${e.toString()}';  
} finally {  
 \_isLoading = false;  
 notifyListeners();  
}

}

void clearChat() { \_messages.clear(); \_error = null; notifyListeners(); }

void clearError() { \_error = null; notifyListeners(); } } -e — FILE: ./lib/src/providers/theme\_provider.dart — import ‘package:flutter/material.dart’; import ‘package:shared\_preferences/shared\_preferences.dart’;

class ThemeProvider with ChangeNotifier { static const String \_themeKey = ‘theme\_mode’; late SharedPreferences \_prefs; ThemeMode \_themeMode = ThemeMode.system;

ThemeProvider() { \_loadThemeMode(); }

ThemeMode get themeMode => \_themeMode;

Future \_loadThemeMode() async { \_prefs = await SharedPreferences.getInstance(); final savedTheme = \_prefs.getString(\_themeKey); if (savedTheme != null) { \_themeMode = ThemeMode.values.firstWhere( (mode) => mode.toString() == savedTheme, orElse: () => ThemeMode.system, ); notifyListeners(); } }

Future setThemeMode(ThemeMode mode) async { if (\_themeMode == mode) return; \_themeMode = mode; await \_prefs.setString(\_themeKey, mode.toString()); notifyListeners(); }

bool get isDarkMode => \_themeMode == ThemeMode.dark; } -e — FILE: ./lib/src/models/message.dart — class Message { final String content; final bool isUser; final DateTime timestamp; final String? riskLevel; final List<Map<String, String>>? resources;

const Message({ required this.content, required this.isUser, required this.timestamp, this.riskLevel, this.resources, });

bool get isCritical => riskLevel == ‘critical’; bool get isHigh => riskLevel == ‘high’; bool get hasResources => resources != null && resources!.isNotEmpty;

@override String toString() { return ‘Message{content: $content, isUser: $isUser, timestamp: $timestamp, riskLevel: $riskLevel}’; } } -e — FILE: ./lib/src/models/chat\_mode.dart — enum ChatMode { mentalHealth, academic;

String get displayName { switch (this) { case ChatMode.mentalHealth: return ‘Mental Health Support’; case ChatMode.academic: return ‘Academic Help’; } }

String get description { switch (this) { case ChatMode.mentalHealth: return ‘Talk about your feelings, get emotional support, and learn coping strategies’; case ChatMode.academic: return ‘Get help with homework, understand concepts, and improve your study skills’; } }

String get icon { switch (this) { case ChatMode.mentalHealth: return ‘❤️’; case ChatMode.academic: return ‘📚’; } } } -e — FILE: ./lib/src/screens/chat\_screen.dart — import ‘package:flutter/material.dart’; import ‘package:provider/provider.dart’; import ‘../providers/chat\_provider.dart’; import ‘../widgets/chat\_message.dart’; import ‘../widgets/chat\_input.dart’; import ‘../widgets/typing\_indicator.dart’; import ‘../widgets/crisis\_resources.dart’;

class ChatScreen extends StatelessWidget { const ChatScreen({super.key});

@override Widget build(BuildContext context) { return Consumer( builder: (context, chatProvider, child) { return Column( children: [ if (chatProvider.error != null) Container( color: Theme.of(context).colorScheme.errorContainer, padding: const EdgeInsets.all(8), child: Row( children: [ Icon( Icons.error\_outline, color: Theme.of(context).colorScheme.error, ), const SizedBox(width: 8), Expanded( child: Text( chatProvider.error!, style: TextStyle( color: Theme.of(context).colorScheme.error, ), ), ), IconButton( icon: const Icon(Icons.close), onPressed: chatProvider.clearError, color: Theme.of(context).colorScheme.error, ), ], ), ), Expanded( child: chatProvider.messages.isEmpty ? Center( child: Column( mainAxisSize: MainAxisSize.min, children: [ Text( chatProvider.currentMode.icon, style: const TextStyle(fontSize: 48), ), const SizedBox(height: 16), Text( chatProvider.currentMode.description, textAlign: TextAlign.center, style: Theme.of(context).textTheme.bodyLarge, ), ], ), ) : ListView.builder( padding: const EdgeInsets.all(16), itemCount: chatProvider.messages.length, itemBuilder: (context, index) { final message = chatProvider.messages[index]; return Column( children: [ ChatMessage(message: message), if (message.hasResources) CrisisResources(resources: message.resources!), ], ); }, ), ), if (chatProvider.isLoading) const TypingIndicator(), const ChatInput(), ], ); }, ); } } -e — FILE: ./lib/src/app.dart — import ‘package:flutter/material.dart’; import ‘package:provider/provider.dart’; import ‘screens/chat\_screen.dart’; import ‘providers/theme\_provider.dart’; import ‘providers/chat\_provider.dart’; import ‘models/chat\_mode.dart’;

class AppScaffold extends StatelessWidget { const AppScaffold({super.key});

@override Widget build(BuildContext context) { final themeProvider = Provider.of(context); final chatProvider = Provider.of(context);

return Scaffold(  
 appBar: AppBar(  
 title: Row(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 Text(chatProvider.currentMode.icon),  
 const SizedBox(width: 8),  
 Text(chatProvider.currentMode.displayName),  
 ],  
 ),  
 actions: [  
 IconButton(  
 icon: Icon(  
 themeProvider.isDarkMode ? Icons.light\_mode : Icons.dark\_mode,  
 ),  
 onPressed: () {  
 themeProvider.setThemeMode(  
 themeProvider.isDarkMode ? ThemeMode.light : ThemeMode.dark,  
 );  
 },  
 ),  
 ],  
 ),  
 body: const ChatScreen(),  
 bottomNavigationBar: NavigationBar(  
 selectedIndex: chatProvider.currentMode == ChatMode.mentalHealth ? 0 : 1,  
 onDestinationSelected: (index) {  
 chatProvider.setMode(  
 index == 0 ? ChatMode.mentalHealth : ChatMode.academic,  
 );  
 },  
 destinations: const [  
 NavigationDestination(  
 icon: Text('❤️'),  
 label: 'Mental Health',  
 ),  
 NavigationDestination(  
 icon: Text('📚'),  
 label: 'Academic',  
 ),  
 ],  
 ),  
);

} } -e — FILE: ./lib/src/widgets/chat\_message.dart — import ‘package:flutter/material.dart’; import ‘package:flutter\_markdown/flutter\_markdown.dart’; import ‘../models/message.dart’;

class ChatMessage extends StatelessWidget { final Message message;

const ChatMessage({ super.key, required this.message, });

@override Widget build(BuildContext context) { final theme = Theme.of(context); final colorScheme = theme.colorScheme;

return Align(  
 alignment: message.isUser  
 ? Alignment.centerRight  
 : Alignment.centerLeft,  
 child: ConstrainedBox(  
 constraints: BoxConstraints(  
 maxWidth: MediaQuery.of(context).size.width \* 0.75,  
 ),  
 child: Card(  
 color: \_getMessageColor(colorScheme),  
 child: Padding(  
 padding: const EdgeInsets.all(12),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 MarkdownBody(  
 data: message.content,  
 styleSheet: MarkdownStyleSheet(  
 p: TextStyle(  
 color: message.isUser  
 ? Colors.white  
 : theme.textTheme.bodyLarge?.color,  
 ),  
 ),  
 ),  
 if (!message.isUser) ...[  
 const SizedBox(height: 4),  
 Text(  
 '${DateTime.now().difference(message.timestamp).inSeconds} seconds ago',  
 style: theme.textTheme.bodySmall?.copyWith(  
 color: message.isUser  
 ? Colors.white70  
 : theme.textTheme.bodySmall?.color?.withOpacity(0.7),  
 ),  
 ),  
 ],  
 ],  
 ),  
 ),  
 ),  
 ),  
);

}

Color \_getMessageColor(ColorScheme colorScheme) { if (message.isUser) { return colorScheme.primary; } if (message.isCritical) { return colorScheme.errorContainer; } if (message.isHigh) { return colorScheme.surfaceContainerHighest; } return colorScheme.surface; } } -e — FILE: ./lib/src/widgets/crisis\_resources.dart — import ‘package:flutter/material.dart’; import ‘package:url\_launcher/url\_launcher.dart’;

class CrisisResources extends StatelessWidget { final List<Map<String, String>> resources;

const CrisisResources({ super.key, required this.resources, });

Future \_launchUrl(String url) async { if (await canLaunchUrl(Uri.parse(url))) { await launchUrl(Uri.parse(url)); } }

@override Widget build(BuildContext context) { return Card( color: Theme.of(context).colorScheme.errorContainer, child: Padding( padding: const EdgeInsets.all(16), child: Column( crossAxisAlignment: CrossAxisAlignment.start, children: [ Row( children: [ Icon( Icons.emergency, color: Theme.of(context).colorScheme.error, ), const SizedBox(width: 8), Text( ‘Crisis Resources’, style: Theme.of(context).textTheme.titleMedium?.copyWith( color: Theme.of(context).colorScheme.error, fontWeight: FontWeight.bold, ), ), ], ), const SizedBox(height: 8), …resources.map((resource) => Padding( padding: const EdgeInsets.symmetric(vertical: 4), child: ListTile( contentPadding: EdgeInsets.zero, title: Text( resource[‘name’] ?? ’‘, style: Theme.of(context).textTheme.bodyLarge?.copyWith( color: Theme.of(context).colorScheme.onErrorContainer, ), ), subtitle: Text( resource[’description’] ??’’, style: Theme.of(context).textTheme.bodyMedium?.copyWith( color: Theme.of(context).colorScheme.onErrorContainer .withOpacity(0.7), ), ), trailing: resource[‘url’] != null ? IconButton( icon: const Icon(Icons.open\_in\_new), onPressed: () => \_launchUrl(resource[‘url’]!), color: Theme.of(context).colorScheme.onErrorContainer, ) : null, ), )), ], ), ), ); } } -e — FILE: ./lib/src/widgets/typing\_indicator.dart — import ‘package:flutter/material.dart’;

class TypingIndicator extends StatelessWidget { const TypingIndicator({super.key});

@override Widget build(BuildContext context) { return Container( padding: const EdgeInsets.symmetric(vertical: 8, horizontal: 16), child: Row( mainAxisSize: MainAxisSize.min, children: [ \_buildDot(context, 0), \_buildDot(context, 1), \_buildDot(context, 2), ], ), ); }

Widget \_buildDot(BuildContext context, int index) { return Container( margin: const EdgeInsets.symmetric(horizontal: 2), height: 8, width: 8, decoration: BoxDecoration( color: Theme.of(context).colorScheme.primary.withOpacity(0.6), shape: BoxShape.circle, ), child: Center( child: AnimatedContainer( duration: const Duration(milliseconds: 600), curve: Curves.easeInOut, height: 8, width: 8, decoration: BoxDecoration( color: Theme.of(context).colorScheme.primary, shape: BoxShape.circle, ), ), ), ); } } -e — FILE: ./lib/src/widgets/chat\_input.dart — import ‘package:flutter/material.dart’; import ‘package:provider/provider.dart’; import ‘../providers/chat\_provider.dart’; import ‘../models/chat\_mode.dart’;

class ChatInput extends StatefulWidget { const ChatInput({super.key});

@override State createState() => \_ChatInputState(); }

class \_ChatInputState extends State { final \_controller = TextEditingController(); bool \_isComposing = false;

@override void dispose() { \_controller.dispose(); super.dispose(); }

void \_handleSubmitted(String text) { if (text.trim().isEmpty) return;

final chatProvider = Provider.of<ChatProvider>(context, listen: false);  
chatProvider.sendMessage(text);  
\_controller.clear();  
setState(() {  
 \_isComposing = false;  
});

}

@override Widget build(BuildContext context) { final theme = Theme.of(context); final chatProvider = Provider.of(context);

return Container(  
 decoration: BoxDecoration(  
 color: theme.colorScheme.surface,  
 border: Border(  
 top: BorderSide(  
 color: theme.colorScheme.outlineVariant,  
 ),  
 ),  
 ),  
 child: SafeArea(  
 child: Padding(  
 padding: const EdgeInsets.all(8.0),  
 child: Row(  
 children: [  
 Expanded(  
 child: TextField(  
 controller: \_controller,  
 onChanged: (text) {  
 setState(() {  
 \_isComposing = text.trim().isNotEmpty;  
 });  
 },  
 onSubmitted: \_handleSubmitted,  
 decoration: InputDecoration(  
 hintText: chatProvider.currentMode == ChatMode.mentalHealth  
 ? 'Share what\'s on your mind...'  
 : 'Ask your academic question...',  
 border: OutlineInputBorder(  
 borderRadius: BorderRadius.circular(20),  
 ),  
 contentPadding: const EdgeInsets.symmetric(  
 horizontal: 16,  
 vertical: 8,  
 ),  
 ),  
 maxLines: null,  
 textCapitalization: TextCapitalization.sentences,  
 ),  
 ),  
 const SizedBox(width: 8),  
 IconButton.filled(  
 onPressed: !\_isComposing  
 ? null  
 : () => \_handleSubmitted(\_controller.text),  
 icon: const Icon(Icons.send),  
 ),  
 ],  
 ),  
 ),  
 ),  
);

} } -e — FILE: ./docker-compose.yml — 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\_: -e — FILE: ./analysis\_options.yaml — # This file configures the analyzer, which statically analyzes Dart code to # check for errors, warnings, and lints. # # The issues identified by the analyzer are surfaced in the UI of Dart-enabled # IDEs (https://dart.dev/tools#ides-and-editors). The analyzer can also be # invoked from the command line by running flutter analyze.

# The following line activates a set of recommended lints for Flutter apps,

# packages, and plugins designed to encourage good coding practices.

include: package:flutter\_lints/flutter.yaml

linter: # The lint rules applied to this project can be customized in the # section below to disable rules from the package:flutter\_lints/flutter.yaml # included above or to enable additional rules. A list of all available lints # and their documentation is published at https://dart.dev/lints. # # Instead of disabling a lint rule for the entire project in the # section below, it can also be suppressed for a single line of code # or a specific dart file by using the // ignore: name\_of\_lint and # // ignore\_for\_file: name\_of\_lint syntax on the line or in the file # producing the lint. rules: # avoid\_print: false # Uncomment to disable the avoid\_print rule # prefer\_single\_quotes: true # Uncomment to enable the prefer\_single\_quotes rule

# Additional information about this file can be found at

# https://dart.dev/guides/language/analysis-options

-e — FILE: ./.dart\_tool/dartpad/web\_plugin\_registrant.dart — // Flutter web plugin registrant file. // // Generated file. Do not edit. //

// @dart = 2.13 // ignore\_for\_file: type=lint

import ‘package:share\_plus/src/share\_plus\_web.dart’; import ‘package:shared\_preferences\_web/shared\_preferences\_web.dart’; import ‘package:url\_launcher\_web/url\_launcher\_web.dart’; import ‘package:flutter\_web\_plugins/flutter\_web\_plugins.dart’;

void registerPlugins([final Registrar? pluginRegistrar]) { final Registrar registrar = pluginRegistrar ?? webPluginRegistrar; SharePlusWebPlugin.registerWith(registrar); SharedPreferencesPlugin.registerWith(registrar); UrlLauncherPlugin.registerWith(registrar); registrar.registerMessageHandler(); } -e — FILE: ./windows/CMakeLists.txt — # Project-level configuration. cmake\_minimum\_required(VERSION 3.14) project(ai\_wellness\_buddy LANGUAGES CXX)

# The name of the executable created for the application. Change this to change

# the on-disk name of your application.

set(BINARY\_NAME “ai\_wellness\_buddy”)

# Explicitly opt in to modern CMake behaviors to avoid warnings with recent

# versions of CMake.

cmake\_policy(VERSION 3.14…3.25)

# Define build configuration option.

get\_property(IS\_MULTICONFIG GLOBAL PROPERTY GENERATOR\_IS\_MULTI\_CONFIG) if(IS\_MULTICONFIG) set(CMAKE\_CONFIGURATION\_TYPES “Debug;Profile;Release” CACHE STRING “” FORCE) else() if(NOT CMAKE\_BUILD\_TYPE AND NOT CMAKE\_CONFIGURATION\_TYPES) set(CMAKE\_BUILD\_TYPE “Debug” CACHE STRING “Flutter build mode” FORCE) set\_property(CACHE CMAKE\_BUILD\_TYPE PROPERTY STRINGS “Debug” “Profile” “Release”) endif() endif() # Define settings for the Profile build mode. set(CMAKE\_EXE\_LINKER\_FLAGS\_PROFILE “${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE}")
set(CMAKE\_SHARED\_LINKER\_FLAGS\_PROFILE "${CMAKE\_SHARED\_LINKER\_FLAGS\_RELEASE}”) set(CMAKE\_C\_FLAGS\_PROFILE “${CMAKE\_C\_FLAGS\_RELEASE}")
set(CMAKE\_CXX\_FLAGS\_PROFILE "${CMAKE\_CXX\_FLAGS\_RELEASE}”)

# Use Unicode for all projects.

add\_definitions(-DUNICODE -D\_UNICODE)

# Compilation settings that should be applied to most targets.

# Be cautious about adding new options here, as plugins use this function by

# default. In most cases, you should add new options to specific targets instead

# of modifying this function.

function(APPLY\_STANDARD\_SETTINGS TARGET) target\_compile\_features({TARGET} PRIVATE /W4 /WX /wd”4100”) target\_compile\_options({TARGET} PRIVATE “\_HAS\_EXCEPTIONS=0”) target\_compile\_definitions(${TARGET} PRIVATE "$<$:\_DEBUG>“) endfunction()

# Flutter library and tool build rules.

set(FLUTTER\_MANAGED\_DIR “${CMAKE\_CURRENT\_SOURCE\_DIR}/flutter")
add\_subdirectory(${FLUTTER\_MANAGED\_DIR})

# Application build; see runner/CMakeLists.txt.

add\_subdirectory(“runner”)

# Generated plugin build rules, which manage building the plugins and adding

# them to the application.

include(flutter/generated\_plugins.cmake)

# === Installation ===

# Support files are copied into place next to the executable, so that it can

# run in place. This is done instead of making a separate bundle (as on Linux)

# so that building and running from within Visual Studio will work.

set(BUILD\_BUNDLE\_DIR “{BINARY\_NAME}>”) # Make the “install” step default, as it’s required to run. set(CMAKE\_VS\_INCLUDE\_INSTALL\_TO\_DEFAULT\_BUILD 1) if(CMAKE\_INSTALL\_PREFIX\_INITIALIZED\_TO\_DEFAULT) set(CMAKE\_INSTALL\_PREFIX “${BUILD\_BUNDLE\_DIR}” CACHE PATH “…” FORCE) endif()

set(INSTALL\_BUNDLE\_DATA\_DIR “${CMAKE\_INSTALL\_PREFIX}/data")
set(INSTALL\_BUNDLE\_LIB\_DIR "${CMAKE\_INSTALL\_PREFIX}”)

install(TARGETS ${BINARY\_NAME} RUNTIME DESTINATION "${CMAKE\_INSTALL\_PREFIX}” COMPONENT Runtime)

install(FILES “${FLUTTER\_ICU\_DATA\_FILE}" DESTINATION "${INSTALL\_BUNDLE\_DATA\_DIR}” COMPONENT Runtime)

install(FILES “${FLUTTER\_LIBRARY}" DESTINATION "${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime)

if(PLUGIN\_BUNDLED\_LIBRARIES) install(FILES “${PLUGIN\_BUNDLED\_LIBRARIES}"
DESTINATION "${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime) endif()

# Copy the native assets provided by the build.dart from all packages.

set(NATIVE\_ASSETS\_DIR “${PROJECT\_BUILD\_DIR}native\_assets/windows/")
install(DIRECTORY "${NATIVE\_ASSETS\_DIR}” DESTINATION “${INSTALL\_BUNDLE\_LIB\_DIR}” COMPONENT Runtime)

# Fully re-copy the assets directory on each build to avoid having stale files

# from a previous install.

set(FLUTTER\_ASSET\_DIR\_NAME “flutter\_assets”) install(CODE ” file(REMOVE\_RECURSE "{FLUTTER\_ASSET\_DIR\_NAME}") ” COMPONENT Runtime) install(DIRECTORY “{FLUTTER\_ASSET\_DIR\_NAME}” DESTINATION “${INSTALL\_BUNDLE\_DATA\_DIR}” COMPONENT Runtime)

# Install the AOT library on non-Debug builds only.

install(FILES “${AOT\_LIBRARY}" DESTINATION "${INSTALL\_BUNDLE\_DATA\_DIR}” CONFIGURATIONS Profile;Release COMPONENT Runtime) -e — FILE: ./windows/runner/CMakeLists.txt — cmake\_minimum\_required(VERSION 3.14) project(runner LANGUAGES CXX)

# Define the application target. To change its name, change BINARY\_NAME in the

# top-level CMakeLists.txt, not the value here, or flutter run will no longer

# work.

# Any new source files that you add to the application should be added here.

add\_executable(${BINARY\_NAME} WIN32
"flutter\_window.cpp"
"main.cpp"
"utils.cpp"
"win32\_window.cpp"
"${FLUTTER\_MANAGED\_DIR}/generated\_plugin\_registrant.cc” “Runner.rc” “runner.exe.manifest” )

# Apply the standard set of build settings. This can be removed for applications

# that need different build settings.

apply\_standard\_settings(${BINARY\_NAME})

# Add preprocessor definitions for the build version.

target\_compile\_definitions(${BINARY\_NAME} PRIVATE "FLUTTER\_VERSION=\"${FLUTTER\_VERSION}"“) target\_compile\_definitions(${BINARY\_NAME} PRIVATE "FLUTTER\_VERSION\_MAJOR=${FLUTTER\_VERSION\_MAJOR}”) target\_compile\_definitions(${BINARY\_NAME} PRIVATE "FLUTTER\_VERSION\_MINOR=${FLUTTER\_VERSION\_MINOR}“) target\_compile\_definitions(${BINARY\_NAME} PRIVATE "FLUTTER\_VERSION\_PATCH=${FLUTTER\_VERSION\_PATCH}”) target\_compile\_definitions(${BINARY\_NAME} PRIVATE "FLUTTER\_VERSION\_BUILD=${FLUTTER\_VERSION\_BUILD}“)

# Disable Windows macros that collide with C++ standard library functions.

target\_compile\_definitions(${BINARY\_NAME} PRIVATE “NOMINMAX”)

# Add dependency libraries and include directories. Add any application-specific

# dependencies here.

target\_link\_libraries({BINARY\_NAME} PRIVATE “dwmapi.lib”) target\_include\_directories(${BINARY\_NAME} PRIVATE "${CMAKE\_SOURCE\_DIR}“)

# Run the Flutter tool portions of the build. This must not be removed.

add\_dependencies(${BINARY\_NAME} flutter\_assemble) -e — FILE: ./windows/flutter/CMakeLists.txt — # This file controls Flutter-level build steps. It should not be edited. cmake\_minimum\_required(VERSION 3.14)

set(EPHEMERAL\_DIR “${CMAKE\_CURRENT\_SOURCE\_DIR}/ephemeral”)

# Configuration provided via flutter tool.

include(${EPHEMERAL\_DIR}/generated\_config.cmake)

# TODO: Move the rest of this into files in ephemeral. See

# https://github.com/flutter/flutter/issues/57146.

set(WRAPPER\_ROOT “${EPHEMERAL\_DIR}/cpp\_client\_wrapper”)

# Set fallback configurations for older versions of the flutter tool.

if (NOT DEFINED FLUTTER\_TARGET\_PLATFORM) set(FLUTTER\_TARGET\_PLATFORM “windows-x64”) endif()

# === Flutter Library ===

set(FLUTTER\_LIBRARY “${EPHEMERAL\_DIR}/flutter\_windows.dll”)

# Published to parent scope for install step.

set(FLUTTER\_LIBRARY ${FLUTTER\_LIBRARY} PARENT\_SCOPE)
set(FLUTTER\_ICU\_DATA\_FILE "${EPHEMERAL\_DIR}/icudtl.dat” PARENT\_SCOPE) set(PROJECT\_BUILD\_DIR “${PROJECT\_DIR}/build/" PARENT\_SCOPE)
set(AOT\_LIBRARY "${PROJECT\_DIR}/build/windows/app.so” PARENT\_SCOPE)

list(APPEND FLUTTER\_LIBRARY\_HEADERS “flutter\_export.h” “flutter\_windows.h” “flutter\_messenger.h” “flutter\_plugin\_registrar.h” “flutter\_texture\_registrar.h” ) list(TRANSFORM FLUTTER\_LIBRARY\_HEADERS PREPEND “${EPHEMERAL\_DIR}/")
add\_library(flutter INTERFACE)
target\_include\_directories(flutter INTERFACE
"${EPHEMERAL\_DIR}” ) target\_link\_libraries(flutter INTERFACE “${FLUTTER\_LIBRARY}.lib”) add\_dependencies(flutter flutter\_assemble)

# === Wrapper ===

list(APPEND CPP\_WRAPPER\_SOURCES\_CORE “core\_implementations.cc” “standard\_codec.cc” ) list(TRANSFORM CPP\_WRAPPER\_SOURCES\_CORE PREPEND “${WRAPPER\_ROOT}/")
list(APPEND CPP\_WRAPPER\_SOURCES\_PLUGIN
"plugin\_registrar.cc"
)
list(TRANSFORM CPP\_WRAPPER\_SOURCES\_PLUGIN PREPEND "${WRAPPER\_ROOT}/”) list(APPEND CPP\_WRAPPER\_SOURCES\_APP “flutter\_engine.cc” “flutter\_view\_controller.cc” ) list(TRANSFORM CPP\_WRAPPER\_SOURCES\_APP PREPEND “${WRAPPER\_ROOT}/”)

# Wrapper sources needed for a plugin.

add\_library(flutter\_wrapper\_plugin STATIC ${CPP\_WRAPPER\_SOURCES\_CORE} ${CPP\_WRAPPER\_SOURCES\_PLUGIN}
)
apply\_standard\_settings(flutter\_wrapper\_plugin)
set\_target\_properties(flutter\_wrapper\_plugin PROPERTIES
POSITION\_INDEPENDENT\_CODE ON)
set\_target\_properties(flutter\_wrapper\_plugin PROPERTIES
CXX\_VISIBILITY\_PRESET hidden)
target\_link\_libraries(flutter\_wrapper\_plugin PUBLIC flutter)
target\_include\_directories(flutter\_wrapper\_plugin PUBLIC
"${WRAPPER\_ROOT}/include” ) add\_dependencies(flutter\_wrapper\_plugin flutter\_assemble)

# Wrapper sources needed for the runner.

add\_library(flutter\_wrapper\_app STATIC ${CPP\_WRAPPER\_SOURCES\_CORE} ${CPP\_WRAPPER\_SOURCES\_APP}
)
apply\_standard\_settings(flutter\_wrapper\_app)
target\_link\_libraries(flutter\_wrapper\_app PUBLIC flutter)
target\_include\_directories(flutter\_wrapper\_app PUBLIC
"${WRAPPER\_ROOT}/include” ) add\_dependencies(flutter\_wrapper\_app flutter\_assemble)

# === Flutter tool backend ===

# *phony* is a non-existent file to force this command to run every time,

# since currently there’s no way to get a full input/output list from the

# flutter tool.

set(PHONY\_OUTPUT “${CMAKE\_CURRENT\_BINARY\_DIR}/\_phony\_")
set\_source\_files\_properties("${PHONY\_OUTPUT}” PROPERTIES SYMBOLIC TRUE) add\_custom\_command( OUTPUT ${FLUTTER\_LIBRARY} ${FLUTTER\_LIBRARY\_HEADERS} ${CPP\_WRAPPER\_SOURCES\_CORE} ${CPP\_WRAPPER\_SOURCES\_PLUGIN} ${CPP\_WRAPPER\_SOURCES\_APP} ${PHONY\_OUTPUT} COMMAND ${CMAKE\_COMMAND} -E env ${FLUTTER\_TOOL\_ENVIRONMENT}
"${FLUTTER\_ROOT}/packages/flutter\_tools/bin/tool\_backend.bat” ${FLUTTER\_TARGET\_PLATFORM} $<CONFIG>
VERBATIM
)
add\_custom\_target(flutter\_assemble DEPENDS
"${FLUTTER\_LIBRARY}” ${FLUTTER\_LIBRARY\_HEADERS} ${CPP\_WRAPPER\_SOURCES\_CORE} ${CPP\_WRAPPER\_SOURCES\_PLUGIN} ${CPP\_WRAPPER\_SOURCES\_APP} )