streamlit>=1.24.0

sqlalchemy>=2.0.20

alembic>=1.11.0

pydantic>=1.10.7

python-dotenv>=1.0.0

bcrypt>=4.0.1

pandas>=2.0.0

plotly>=5.16.1

langchain>=0.0.266 # pin to a reasonably recent release; adjust if your environment needs

openai>=0.29.0

requests>=2.31.0

email-validator>=2.0.0

Env file

# Copy to .env and fill values

# LLM - either set OPENAI\_API\_KEY or set GEMINI\_API\_KEY and change llm\_provider accordingly

OPENAI\_API\_KEY=your\_openai\_key\_here

# GEMINI\_API\_KEY=your\_gemini\_key\_here

# Email for alerts (SMTP)

EMAIL\_USER=youremail@example.com

EMAIL\_PASS=your\_email\_password\_or\_app\_password

SMTP\_SERVER=smtp.gmail.com

SMTP\_PORT=587

# Database (SQLAlchemy URI)

DATABASE\_URL=sqlite:///./synermind.db

# App secret

SECRET\_KEY=change\_this\_secret\_value

# config.py

import os

from dotenv import load\_dotenv

load\_dotenv()

DATABASE\_URL = os.getenv("DATABASE\_URL", "sqlite:///./synermind.db")

OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

GEMINI\_API\_KEY = os.getenv("GEMINI\_API\_KEY")

EMAIL\_USER = os.getenv("EMAIL\_USER")

EMAIL\_PASS = os.getenv("EMAIL\_PASS")

SMTP\_SERVER = os.getenv("SMTP\_SERVER", "smtp.gmail.com")

SMTP\_PORT = int(os.getenv("SMTP\_PORT", 587))

SECRET\_KEY = os.getenv("SECRET\_KEY", "change\_this")

# models.py

from sqlalchemy import create\_engine, Column, Integer, String, DateTime, Text, Boolean, ForeignKey

from sqlalchemy.orm import declarative\_base, sessionmaker, relationship

from sqlalchemy.sql import func

from config import DATABASE\_URL

Base = declarative\_base()

engine = create\_engine(DATABASE\_URL, echo=False, future=True)

SessionLocal = sessionmaker(bind=engine, autoflush=False, autocommit=False)

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

username = Column(String(128), unique=True, index=True, nullable=False)

password\_hash = Column(String(256), nullable=False)

email = Column(String(256), nullable=False)

emergency\_contact = Column(String(128), nullable=True)

created\_at = Column(DateTime(timezone=True), server\_default=func.now())

mood\_logs = relationship("MoodLog", back\_populates="user")

interactions = relationship("Interaction", back\_populates="user")

alerts = relationship("Alert", back\_populates="user")

class MoodLog(Base):

\_\_tablename\_\_ = "mood\_logs"

id = Column(Integer, primary\_key=True, index=True)

user\_id = Column(Integer, ForeignKey("users.id"))

mood = Column(String(64))

intensity = Column(Integer, default=5)

note = Column(Text, nullable=True)

created\_at = Column(DateTime(timezone=True), server\_default=func.now())

user = relationship("User", back\_populates="mood\_logs")

class Interaction(Base):

\_\_tablename\_\_ = "interactions"

id = Column(Integer, primary\_key=True, index=True)

user\_id = Column(Integer, ForeignKey("users.id"))

agent\_type = Column(String(64))

user\_msg = Column(Text)

agent\_reply = Column(Text)

created\_at = Column(DateTime(timezone=True), server\_default=func.now())

user = relationship("User", back\_populates="interactions")

class Alert(Base):

\_\_tablename\_\_ = "alerts"

id = Column(Integer, primary\_key=True, index=True)

user\_id = Column(Integer, ForeignKey("users.id"))

alert\_type = Column(String(100))

message = Column(Text)

resolved = Column(Boolean, default=False)

created\_at = Column(DateTime(timezone=True), server\_default=func.now())

user = relationship("User", back\_populates="alerts")

def init\_db():

Base.metadata.create\_all(bind=engine)

if \_\_name\_\_ == "\_\_main\_\_":

init\_db()

print("DB initialized.")

# db\_ops.py

from sqlalchemy.exc import IntegrityError

from models import SessionLocal, User, MoodLog, Interaction, Alert

from utils import hash\_password, verify\_password

def create\_user(username: str, password: str, email: str, emergency\_contact: str):

db = SessionLocal()

try:

user = User(username=username, password\_hash=hash\_password(password),

email=email, emergency\_contact=emergency\_contact)

db.add(user)

db.commit()

db.refresh(user)

return user

except IntegrityError:

db.rollback()

return None

finally:

db.close()

def authenticate\_user(username: str, password: str):

db = SessionLocal()

try:

user = db.query(User).filter(User.username == username).first()

if not user:

return None

if verify\_password(password, user.password\_hash):

return user

return None

finally:

db.close()

def log\_interaction(user\_id: int, agent\_type: str, user\_msg: str, agent\_reply: str):

db = SessionLocal()

try:

inter = Interaction(user\_id=user\_id, agent\_type=agent\_type, user\_msg=user\_msg, agent\_reply=agent\_reply)

db.add(inter)

db.commit()

db.refresh(inter)

return inter

finally:

db.close()

def add\_mood(user\_id: int, mood: str, intensity: int = 5, note: str = None):

db = SessionLocal()

try:

ml = MoodLog(user\_id=user\_id, mood=mood, intensity=intensity, note=note)

db.add(ml)

db.commit()

db.refresh(ml)

return ml

finally:

db.close()

def get\_mood\_history(user\_id: int):

db = SessionLocal()

try:

rows = db.query(MoodLog).filter(MoodLog.user\_id == user\_id).order\_by(MoodLog.created\_at.asc()).all()

return rows

finally:

db.close()

def create\_alert(user\_id: int, alert\_type: str, message: str):

db = SessionLocal()

try:

a = Alert(user\_id=user\_id, alert\_type=alert\_type, message=message)

db.add(a)

db.commit()

db.refresh(a)

return a

finally:

db.close()

# llm\_provider.py

from config import OPENAI\_API\_KEY, GEMINI\_API\_KEY

import os

# Primary recommendation: use OpenAI-compatible LLM for LangChain integration.

# If you want to use Google Gemini, replace ChatOpenAI with the Gemini wrapper

# or use a custom LangChain LLM wrapper for Gemini (not included here).

#

# For now we expose a factory function returning a LangChain LLM object.

# By default uses ChatOpenAI. If you want Gemini, replace the ChatOpenAI

# import / instantiation below with Gemini-compatible one or a custom wrapper.

try:

from langchain.chat\_models import ChatOpenAI

LLM\_AVAILABLE = True

except Exception:

ChatOpenAI = None

LLM\_AVAILABLE = False

def get\_llm(model\_name: str = "gpt-4o-mini", temperature: float = 0.2):

"""

Returns a LangChain-compatible LLM.

By default tries ChatOpenAI which uses OPENAI\_API\_KEY.

If you want to use Gemini, replace this function with your Gemini wrapper

that implements LangChain LLM interface (or change agents.py to call Gemini directly).

"""

if OPENAI\_API\_KEY and ChatOpenAI:

os.environ["OPENAI\_API\_KEY"] = OPENAI\_API\_KEY

return ChatOpenAI(model\_name=model\_name, temperature=temperature)

# Fallback: return a very simple dummy LLM (not ideal for production)

class DummyLLM:

def \_\_init\_\_(self, temperature=temperature):

self.temperature = temperature

def \_\_call\_\_(self, \*args, \*\*kwargs):

return "Sorry — no LLM configured. Set OPENAI\_API\_KEY or implement a Gemini wrapper."

return DummyLLM()

# tools.py

from langchain.tools import Tool

from db\_ops import add\_mood, get\_mood\_history, create\_alert

from config import EMAIL\_USER, EMAIL\_PASS, SMTP\_SERVER, SMTP\_PORT

import smtplib

from email.message import EmailMessage

from typing import Dict, Any

from models import SessionLocal, User

def send\_email(to\_email: str, subject: str, body: str) -> Dict[str, Any]:

if not EMAIL\_USER or not EMAIL\_PASS:

return {"ok": False, "error": "Email not configured"}

try:

msg = EmailMessage()

msg["From"] = EMAIL\_USER

msg["To"] = to\_email

msg["Subject"] = subject

msg.set\_content(body)

with smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT) as smtp:

smtp.starttls()

smtp.login(EMAIL\_USER, EMAIL\_PASS)

smtp.send\_message(msg)

return {"ok": True}

except Exception as e:

return {"ok": False, "error": str(e)}

# Tool wrappers for LangChain

def tool\_log\_mood(args: str) -> str:

"""

Expects args: "user\_id|mood|intensity|note"

Example: "42|Sad|6|Felt anxious after meeting"

"""

try:

parts = args.split("|", 3)

user\_id = int(parts[0])

mood = parts[1]

intensity = int(parts[2]) if len(parts) > 2 and parts[2].isdigit() else 5

note = parts[3] if len(parts) > 3 else None

ml = add\_mood(user\_id=user\_id, mood=mood, intensity=intensity, note=note)

return f"OK: mood logged id={ml.id}"

except Exception as e:

return f"ERROR logging mood: {str(e)}"

def tool\_get\_mood\_history(args: str) -> str:

"""args: user\_id"""

try:

uid = int(args.strip())

rows = get\_mood\_history(uid)

lines = [f"{r.created\_at.strftime('%Y-%m-%d')} | {r.mood} | intensity:{r.intensity}" for r in rows]

return "\n".join(lines) if lines else "No mood history"

except Exception as e:

return f"ERROR: {str(e)}"

def tool\_send\_alert(args: str) -> str:

"""

Expects args: "user\_id|subject|message"

Sends email to emergency contact (if email) else to user email; also saves Alert row.

"""

try:

parts = args.split("|", 2)

uid = int(parts[0])

subject = parts[1]

message = parts[2] if len(parts) > 2 else ""

# save alert

a = create\_alert(user\_id=uid, alert\_type=subject, message=message)

# get contact

db = SessionLocal()

user = db.query(User).filter(User.id == uid).first()

db.close()

to\_email = user.emergency\_contact if user and user.emergency\_contact and "@" in user.emergency\_contact else (user.email if user else None)

if not to\_email:

return f"ALERT saved (id={a.id}) but no email to send."

res = send\_email(to\_email, f"Synermind Alert: {subject}", f"User: {user.username}\n\n{message}")

if res.get("ok"):

return f"ALERT sent to {to\_email}"

return f"Alert saved but email failed: {res.get('error')}"

except Exception as e:

return f"ERROR sending alert: {str(e)}"

# Construct LangChain Tool objects

LOG\_MOOD\_TOOL = Tool.from\_function(func=tool\_log\_mood, name="log\_mood", description="Log user's mood: 'user\_id|mood|intensity|note'")

GET\_MOOD\_HISTORY\_TOOL = Tool.from\_function(func=tool\_get\_mood\_history, name="get\_mood\_history", description="Get mood history for user: 'user\_id'")

SEND\_ALERT\_TOOL = Tool.from\_function(func=tool\_send\_alert, name="send\_alert", description="Send alert and save to DB: 'user\_id|subject|message'")

# agents.py

from langchain.memory import ConversationBufferWindowMemory

from langchain.agents import initialize\_agent, AgentType

from llm\_provider import get\_llm

from tools import LOG\_MOOD\_TOOL, GET\_MOOD\_HISTORY\_TOOL, SEND\_ALERT\_TOOL

LLM\_MODEL = "gpt-4o-mini" # default model name (change if using OpenAI or Gemini equivalent)

# Create one LLM instance for agents (LangChain uses this object)

llm = get\_llm(model\_name=LLM\_MODEL, temperature=0.2)

# Memories: small window memory per agent (keeps last N interactions)

mood\_memory = ConversationBufferWindowMemory(k=6, memory\_key="mood\_memory", return\_messages=True)

therapy\_memory = ConversationBufferWindowMemory(k=8, memory\_key="therapy\_memory", return\_messages=True)

routine\_memory = ConversationBufferWindowMemory(k=6, memory\_key="routine\_memory", return\_messages=True)

crisis\_memory = ConversationBufferWindowMemory(k=10, memory\_key="crisis\_memory", return\_messages=True)

# Define specialized agents as agent executors with specific tool access:

# Mood Agent: allowed to log mood and read mood history

mood\_agent = initialize\_agent(

tools=[LOG\_MOOD\_TOOL, GET\_MOOD\_HISTORY\_TOOL],

llm=llm,

agent=AgentType.CHAT\_CONVERSATIONAL\_REACT\_DESCRIPTION,

memory=mood\_memory,

verbose=False,

max\_iterations=3,

)

# Therapy Agent: conversational, no DB write tools (gives CBT guidance)

therapy\_agent = initialize\_agent(

tools=[],

llm=llm,

agent=AgentType.CHAT\_CONVERSATIONAL\_REACT\_DESCRIPTION,

memory=therapy\_memory,

verbose=False,

max\_iterations=3,

)

# Routine Agent: can read mood history and suggest routines (no alerts)

routine\_agent = initialize\_agent(

tools=[GET\_MOOD\_HISTORY\_TOOL],

llm=llm,

agent=AgentType.CHAT\_CONVERSATIONAL\_REACT\_DESCRIPTION,

memory=routine\_memory,

verbose=False,

max\_iterations=3,

)

# Crisis Agent: has send\_alert tool and can request mood history—safety-first

crisis\_agent = initialize\_agent(

tools=[SEND\_ALERT\_TOOL, GET\_MOOD\_HISTORY\_TOOL],

llm=llm,

agent=AgentType.CHAT\_CONVERSATIONAL\_REACT\_DESCRIPTION,

memory=crisis\_memory,

verbose=False,

max\_iterations=3,

)

# Expose in a dict for router usage

AGENTS = {

"mood": mood\_agent,

"therapy": therapy\_agent,

"routine": routine\_agent,

"crisis": crisis\_agent,

# router.py

from langchain.chains.router import LLMRouterChain, MultiPromptChain

from langchain.prompts import PromptTemplate

from llm\_provider import get\_llm

from agents import AGENTS

# Router LLM (can be same or different; keep deterministic)

router\_llm = get\_llm(model\_name="gpt-4o-mini", temperature=0.0)

router\_template = PromptTemplate.from\_template(

"""You are Synermind Router. Given a user message, choose exactly one destination from:

- mood

- therapy

- routine

- crisis

Output only the destination keyword on a single line.

Message: {input}"""

)

router\_chain = LLMRouterChain.from\_llm(router\_llm, router\_template)

# MultiPromptChain stitches router -> destination chains

multi\_agent\_chain = MultiPromptChain(router\_chain=router\_chain, destination\_chains=AGENTS, default\_chain=AGENTS["mood"])

# utils.py

import bcrypt

import re

import pandas as pd

import plotly.express as px

from db\_ops import get\_mood\_history

CRISIS\_KEYWORDS = ["suicide", "kill myself", "end my life", "self-harm", "hurt myself", "want to die", "i'm going to die"]

def hash\_password(password: str) -> str:

return bcrypt.hashpw(password.encode("utf-8"), bcrypt.gensalt()).decode()

def verify\_password(password: str, hashed: str) -> bool:

try:

return bcrypt.checkpw(password.encode("utf-8"), hashed.encode("utf-8"))

except Exception:

return False

def contains\_crisis\_keywords(text: str) -> bool:

t = text.lower()

return any(kw in t for kw in CRISIS\_KEYWORDS)

def mood\_history\_figure(user\_id: int):

rows = get\_mood\_history(user\_id)

if not rows:

return None

df = pd.DataFrame([{"timestamp": r.created\_at, "mood": r.mood, "intensity": r.intensity} for r in rows])

df["date"] = pd.to\_datetime(df["timestamp"]).dt.date

# aggregate intensity by date

agg = df.groupby("date").intensity.mean().reset\_index()

fig = px.line(agg, x="date", y="intensity", title="Mood intensity over time (avg)", markers=True)

return fig

# app.py

import streamlit as st

from models import init\_db

from db\_ops import create\_user, authenticate\_user, log\_interaction, add\_mood, get\_mood\_history

from router import multi\_agent\_chain

from agents import AGENTS

from utils import contains\_crisis\_keywords, mood\_history\_figure

from config import SECRET\_KEY

import time

st.set\_page\_config(page\_title="Synermind — Multi-Agent Mental Wellness", layout="wide")

init\_db()

# Session state

if "user" not in st.session\_state:

st.session\_state.user = None

# Auth UI

def sign\_up\_ui():

st.subheader("Sign Up")

uname = st.text\_input("Username", key="su\_username")

email = st.text\_input("Email", key="su\_email")

emergency = st.text\_input("Emergency contact (email or phone)", key="su\_emergency")

pwd = st.text\_input("Password", type="password", key="su\_password")

if st.button("Create account"):

if not uname or not pwd or not email:

st.error("Username, password and email are required.")

return

user = create\_user(uname, pwd, email, emergency)

if user:

st.success("Account created — please sign in.")

else:

st.error("Username already exists.")

def sign\_in\_ui():

st.subheader("Sign In")

uname = st.text\_input("Username", key="si\_username")

pwd = st.text\_input("Password", type="password", key="si\_password")

if st.button("Sign in"):

user = authenticate\_user(uname, pwd)

if user:

st.session\_state.user = {"id": user.id, "username": user.username}

st.experimental\_rerun()

else:

st.error("Invalid credentials.")

# Landing

if st.session\_state.user is None:

col1, col2 = st.columns(2)

with col1:

sign\_in\_ui()

with col2:

sign\_up\_ui()

st.stop()

# Main app

user = st.session\_state.user

st.sidebar.markdown(f"\*\*Signed in as:\*\* {user['username']}")

if st.sidebar.button("Sign out"):

st.session\_state.user = None

st.experimental\_rerun()

st.title("Synermind — Multi-Agent Mental Wellness")

# Layout: left chat, right mood + controls

left, right = st.columns([2,1])

with left:

st.header("Chat")

if "chat\_history" not in st.session\_state:

st.session\_state.chat\_history = []

user\_msg = st.text\_input("Say something to Synermind...", key="chat\_input")

if st.button("Send"):

if user\_msg:

# first do simple keyword crisis check — fast path

if contains\_crisis\_keywords(user\_msg):

dest = "crisis"

else:

# use LLM router to determine destination

# multi\_agent\_chain.run returns destination chain's response

dest = None

# If router step needed:

if not dest:

try:

# run the router chain; it will route and call appropriate agent chain

response = multi\_agent\_chain.run(user\_msg)

# MultiPromptChain by default returns the destination chain's output text

# We do not directly have the destination label, so do a small check by re-routing to get label

# or in practice you can use the router\_chain to extract the agent label; for simplicity,

# we'll call the router separately to get agent name

from router import router\_chain

chosen = router\_chain.predict\_and\_parse(input=user\_msg)

agent\_label = chosen.strip().lower() if chosen else "mood"

except Exception as e:

response = f"Error: {str(e)}"

agent\_label = "mood"

else:

agent\_label = dest

# call agent directly

agent = AGENTS.get(agent\_label)

if agent:

response = agent.run(user\_msg)

else:

response = "Sorry — no agent available."

# log interaction

log\_interaction(user\_id=user['id'], agent\_type=agent\_label, user\_msg=user\_msg, agent\_reply=response)

st.session\_state.chat\_history.append({"user": user\_msg, "agent": agent\_label, "reply": response, "time": time.time()})

# render latest

st.markdown(f"\*\*You:\*\* {user\_msg}")

st.markdown(f"\*\*{agent\_label.capitalize()} Agent:\*\* {response}")

# if crisis -> ensure alert triggered (the crisis agent can call send\_alert tool; also we double-check)

if agent\_label == "crisis" or contains\_crisis\_keywords(user\_msg):

st.error("Crisis detected — alert process initiated (email to emergency contact).")

st.divider()

st.subheader("Conversation history (last 30)")

history = st.session\_state.chat\_history[::-1][:30]

for h in history:

st.markdown(f"\*\*You:\*\* {h['user']}")

st.markdown(f"\*\*{h['agent'].capitalize()} Agent:\*\* {h['reply']}")

st.write("---")

with right:

st.header("Mood Logger")

mood = st.selectbox("How are you feeling?", ["Very Happy","Happy","Neutral","Sad","Very Sad","Anxious","Angry"], key="mood\_select")

intensity = st.slider("Intensity", 1, 10, 5, key="mood\_intensity")

note = st.text\_area("Optional note", key="mood\_note")

if st.button("Log Mood"):

ml = add\_mood(user\_id=user['id'], mood=mood, intensity=intensity, note=note)

st.success("Mood logged.")

st.divider()

st.header("Mood Insights")

fig = mood\_history\_figure(user['id'])

if fig:

st.plotly\_chart(fig, use\_container\_width=True)

else:

st.info("No mood data yet. Log moods to see trends.")

st.divider()

st.header("Quick Actions")

if st.button("Show recent interactions (DB)"):

from models import SessionLocal, Interaction

db = SessionLocal()

rows = db.query(Interaction).filter(Interaction.user\_id == user['id']).order\_by(Interaction.created\_at.desc()).limit(20).all()

db.close()

for r in rows:

st.write(f"[{r.created\_at.strftime('%Y-%m-%d %H:%M')}] {r.agent\_type} -> {r.agent\_reply[:200]}")

st.markdown("---")

st.caption("Synermind runs agents via LangChain. Make sure to set OPENAI\_API\_KEY (or replace llm\_provider to use Gemini) before running.")