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.")