```
.
├── __init__.py
├── assistant_service.py
├── database.py
├── main.py
├── models.py
└── schemas.py
```

```
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb app % find .
-type f -name "*.py" -exec echo -e
"\n\n==============================\nFile: {}
\n==============================\n" \; -exec cat {} \; -exec echo -e
"\n\n" \;

-e \n\n==============================\nFile: ./
models.py\n==============================\n
# app/models.py
from sqlalchemy import Column, Integer, String, Enum, Text, ForeignKey,
TIMESTAMP, func
from sqlalchemy.orm import relationship
from .database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String(255), unique=True, nullable=False)
    password = Column(String(255), nullable=False)

    messages = relationship("Message", back_populates="user")
    threads = relationship("UserThread", back_populates="user")

class UserThread(Base):
    __tablename__ = "user_threads"

    thread_id = Column(String(255), primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)

    user = relationship("User", back_populates="threads")
    messages = relationship("Message", back_populates="thread")

class Message(Base):
    __tablename__ = "messages"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    thread_id = Column(String(255),
ForeignKey("user_threads.thread_id"), nullable=False)
    role = Column(Enum('user', 'assistant'), nullable=False)
```

```
    content = Column(Text, nullable=False)
    timestamp = Column(TIMESTAMP, server_default=func.now())

    user = relationship("User", back_populates="messages")
    thread = relationship("UserThread", back_populates="messages")
-e \n\n
-e \n\n==============================\nFile: ./
database.py\n==============================\n
# app/database.py
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

# Carica le variabili d'ambiente dal file .env
load_dotenv()

DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")
DB_NAME = os.getenv("DB_NAME")



DATABASE_URL = f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:
{DB_PORT}/{DB_NAME}"

engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()


-e \n\n
-e \n\n==============================\nFile: ./
__init__.py\n==============================\n
-e \n\n
-e \n\n==============================\nFile: ./
schemas.py\n==============================\n
# app/schemas.py
from pydantic import BaseModel

class MessageRequest(BaseModel):
    user_id: int
    message: str

class MessageResponse(BaseModel):
    assistant_response: str
-e \n\n
-e \n\n==============================\nFile: ./
assistant_service.py\n==============================\n
# app/assistant_service.py
import os
import time
```

```
from openai import OpenAI
from dotenv import load_dotenv
from sqlalchemy.orm import Session
from . import models

# Carica le variabili d'ambiente
load_dotenv()

# Configura la tua API key e l'ID dell'assistente
api_key = os.getenv("OPENAI_API_KEY")
assistant_id = os.getenv("ASSISTANT_ID")

# Inizializza il client OpenAI
client = OpenAI(api_key=api_key)

# Dizionario per mappare user_id a thread_id in memoria
user_threads = {}

def wait_on_run(run, thread_id):
    while run.status in ["queued", "in_progress"]:
        run = client.beta.threads.runs.retrieve(
            thread_id=thread_id,
            run_id=run.id,
        )
        time.sleep(0.5)
    return run

def submit_message(thread_id, user_message):
    client.beta.threads.messages.create(
        thread_id=thread_id,
        role="user",
        content=user_message
    )
    return client.beta.threads.runs.create(
        thread_id=thread_id,
        assistant_id=assistant_id,
    )

def get_response(thread_id):
    return client.beta.threads.messages.list(thread_id=thread_id,
order="asc")

def get_or_create_thread(user_id, db: Session):
    if user_id in user_threads:
        return user_threads[user_id]
    else:
        thread = client.beta.threads.create()
        user_threads[user_id] = thread.id

        user_thread_entry = models.UserThread(
            thread_id=thread.id,
            user_id=user_id
        )
        db.add(user_thread_entry)
        db.commit()
        db.refresh(user_thread_entry)
```

```
        return thread.id-e \n\n
-e \n\n==============================\nFile: ./
main.py\n==============================\n
# app/main.py
from fastapi import FastAPI, HTTPException, Depends
from fastapi.middleware.cors import CORSMiddleware
from sqlalchemy.orm import Session
from .database import SessionLocal, engine
from . import models
from .schemas import MessageRequest, MessageResponse
from .assistant_service import submit_message, wait_on_run,
get_response, get_or_create_thread
import logging

models.Base.metadata.create_all(bind=engine)

app = FastAPI()

origins = [
    "http://localhost",
    "http://localhost:51096",
]

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

# Configurazione del logger
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
handler = logging.StreamHandler()
handler.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

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

@app.post("/send_message/", response_model=MessageResponse)
def send_message(request: MessageRequest, db: Session =
Depends(get_db)):
    try:
        user_id = request.user_id
        message = request.message
```

```
        logger.info(f"Ricevuto messaggio da utente {user_id}:
{message}")

        user = db.query(models.User).filter(models.User.id ==
user_id).first()
        if not user:
            logger.error(f"Utente con ID {user_id} non trovato.")
            raise HTTPException(status_code=404, detail="Utente non
trovato")

        thread_id = get_or_create_thread(user_id, db)
        logger.info(f"Thread ID recuperato o creato: {thread_id}")

        user_message_entry = models.Message(
            user_id=user_id,
            thread_id=thread_id,
            role="user",
            content=message
        )
        db.add(user_message_entry)
        db.commit()
        db.refresh(user_message_entry)
        logger.info(f"Messaggio dell'utente salvato nel database:
{message}")

        run = submit_message(thread_id, message)
        logger.info(f"Messaggio inviato all'assistente, Run ID:
{run.id}")

        run = wait_on_run(run, thread_id)
        logger.info(f"Run completato: {run.status}")

        messages = get_response(thread_id)
        logger.debug(f"Risposte recuperate dal thread: {messages}")

        assistant_response = ""
        if messages.data:
            for msg in reversed(messages.data):
                if msg.role == "assistant":
                    if isinstance(msg.content, list) and
len(msg.content) > 0:
                        text_block = msg.content[0]
                        if hasattr(text_block, 'text') and
hasattr(text_block.text, 'value'):
                            assistant_response = text_block.text.value
                            logger.info("Risposta valida trovata
nell'assistente")
                            break

        assistant_response = assistant_response.strip()

        if assistant_response:
            assistant_message_entry = models.Message(
                user_id=user_id,
                thread_id=thread_id,
```

```
                role="assistant",
                content=assistant_response
            )
            db.add(assistant_message_entry)
            db.commit()
            db.refresh(assistant_message_entry)
            logger.info(f"Risposta dell'assistente salvata nel database:
{assistant_response}")

            return
MessageResponse(assistant_response=assistant_response)
        else:
            logger.error("Nessuna risposta valida trovata
nell'assistente.")
            raise HTTPException(status_code=500, detail="Nessuna
risposta valida dall'assistente")

    except Exception as e:
        logger.error(f"Errore durante l'invio del messaggio: {str(e)}")
        raise HTTPException(status_code=500, detail=str(e))
#ok-e \n\n
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb app %
```