BÁO CÁO TIẾN ĐỘ TUẦN 6

Phạm Công Minh- B22DCCN542

Nội dung công việc:Thực hiện chức năng lưu lịch sử trò chuyện vào mongodb, làm chức năng tóm tắt cơ bản cho chat bot

---

Trong tuần 6, em đã dành thời gian để làm lại cấu trúc của database Mongodb, học fastapi cơ bản để xây dựng các route giúp lưu thông tin user, thông tin môn học, các message_history. Nội dung cụ thể em đã thực hiện như sau:

1. Cấu trúc lại database model với fastapi

   Phần model user và các hàm sử lý được lưu tại file Controller.py với đường dẫn: src/model/scripts.Đầu tiên import thư viện BaseModel để tạo classmodel

```
from pydantic import BaseModel
```

Tiếp đến là định nghĩa các classmodel:

```
class Message(BaseModel):

    role: str

    content: str

class Subject(BaseModel):

    subjectId: str

    subjectName: str

    messages: List[Message]



class User(BaseModel):

    fullName: str

    userId: str

    fbLink: str

    subject: List[Subject] = Field(default_factory=list)
```

Với Message để lưu lịch sử trò chuyện có các trường:

+ role có 2 giá trị là user và assistant
+ content: lưu nội dung cuộc trò chuyện

Với Subject để lưu thông tin của tài liệu được dùng để hỏi đáp: gồm có id tài liệu, tên tài liệu và cuối cùng là các List message

Cuối cùng là class User là class chính, có các thông tin cơ bản như tên, userId, fbLink để cho chatbot kết nối với messenger sau này, và list danh sách các file tài liệu(ở đây em dùng thuộc tính Field để tự động tạo 1 List rỗng các subject)

Tiếp theo là kết nối với Mongodb và sử dụng collection có tên là Users

```
client = motor.motor_asyncio.AsyncIOMotorClient()

client =
motor.motor_asyncio.AsyncIOMotorClient("mongodb+srv://phamminh7292004:Emh1ozB80dM675ve@cluster
0.un3mk.mongodb.net/Auth?retryWrites=true&w=majority&appName=Cluster0")

db = client.Auth

db = client["Auth"]



collection = db.Users

collection = db["Users"]
```

Cuối cùng là các hàm xử lý để gửi và nhân dữ liệu từ mongodb:

+ Signup:truyền vào 1 biến user có kiểu là User được định nghĩa ở trên, sau đó gọi hàm insert_one để lưu thông tin user

```
async def signup(user: User):



    thisUser = jsonable_encoder(user)

    result = await collection.insert_one(thisUser)

    return {"status": "success", "user_id": str(result.inserted_id)}
```

+ Add_subject: hàm thêm môn học, trong hàm này ta sẽ lấy giá trị các tham số trên đường truyền là user_id và các giá trị từ body của lệnh req gửi lên, sau đó sẽ khởi tạo 1 object là new_subject, và cuối cùng là gọi hàm update_one để lưu thông tin

```
async def add_subject(

    user_id: str = Path(...),

    subject_id: str = Body(...),

    subject_name: str = Body(...)
```

```
):
    new_subject = {
        "subjectId": subject_id,
        "subjectName": subject_name,
        "messages": []
    }

    result = await collection.update_one(
        {"userId": user_id},
        {"$push": {"subject": new_subject}}
    )

    if result.modified_count == 0:
        return {"status": "fail", "message": "User not found or subject not added"}

    return {"status": "success", "message": "Subject added successfully"}
```

+ Hàm update_message: để lưu lịch sử trò chuyện của người dùng, lấy user_id, subject_id từ Path(),và các messages từ body của req.Sau đó encode các message để về dạng json, cuối cùng là gọi hàm find_one_and_update để lưu messages

```
async def update_message(user_id :str = Path(...), subject_id: str = Path(...)
,messages: List[Message] = Body(...) ):

    message_json = jsonable_encoder(messages)
    result = await collection.find_one_and_update(
        {"userId": user_id, "subject.subjectId": subject_id},
        {"$set": {"subject.$.messages":message_json}}
    )
    if result:
```

```
        return {
            "status": "success",
            "message": "Message updated successfully"
        }
    else:
        return {
            "status": "fail",
            "message": "Error on updating message"
        }
```

+ Cuối cùng là hàm get_message: cũng có user_id và subject_id được lấy từ url. tiếp đó dùng hàm find_one tìm user có id trùng với user_id và lưu vào biến thisUser, cuối cùng duyệt qua danh sách các subject và trả về danh sách message mà có subjectid trùng với subject-id

```
async def get_message(user_id :str = Path(...), subject_id: str = Path(...)):
    thisUser = await collection.find_one({"userId": user_id})
    if not thisUser:
        return {
            "status": "fail",
            "message": "User not found"
        }
    for subject in thisUser.get("subject", []):
        if subject["subjectId"] == subject_id:
            return {
                "status": "success",
                "messages": subject.get("messages", [])
            }
    return {
        "status": "fail",
```

```
        "message": "Subject not found"

    }
```

2. Tạo các route đơn giản với fastapi, được lưu trong file main.py có đường dẫn là src/model/scripts”

```
from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from controller import signup, get_message, update_message, add_subject

app = FastAPI()

@app.get("/")

def read_root():

    return "you must signup to use AI free"

# Route từ controller

app.post("/signup/")(signup)

app.get("/get_message/{user_id}/{subject_id}")(get_message)

app.post("/update_message/{user_id}/{subject_id}")(update_message)

app.post("/add_subject/{user_id}")(add_subject)
```

trong phần này em có cấu hình CROS để giúp ứng dụng frontends có thể gửi được data đến backend fastapi để xử lý

```
origins = [

    "http://localhost:5173",

]

app.add_middleware(

    CORSMiddleware,

    allow_origins=origins,

    allow_credentials=True,

    allow_methods=["*"],

    allow_headers=["*"],)
```

3. Thay đổi file app.py để chatbot tích hợp chat_history:

Đầu tiên là lấy lịch sử trò chuyện và lưu vào biến mongo_mess_history, sau đó tạo mảng chat_history = [] để lưu dưới dạng langchain-format

```
mongo_mess_history = (asyncio.run(get_message("B22DCCN542","lsd"))).get('messages')

    chat_history = []

    for msg in mongo_mess_history:

        if msg["role"] == "user":

            chat_history.append(HumanMessage(content=msg["content"]))

        elif msg["role"] == "assistant":

            chat_history.append(AIMessage(content=msg["content"]))
```

Sau đó truyền vào hàm invoke biến chat_history để chatbot lấy thông tin lịch sử hội thoại:

```
response = agent_executor.invoke({

        "input": user_input,

        "chat_history": chat_history

    })
```

Cuối cùng sử lý message và lưu lại vào database:

```
thisUserMessage = {"role": "user", "content": user_input} #Tạo message mới cho người dùng

    thisAssistantMessage = {"role": "assistant", "content": response["output"]} #Tạo message
mới cho trợ lý

    mongo_mess_history.append(thisUserMessage) #Thêm message mới vào lịch sử

    mongo_mess_history.append(thisAssistantMessage)

    mongo_mess_history = mongo_mess_history[-20:] #Giới hạn lịch sử tin nhắn trong 20 tin nhắn
gần nhất

    # Cập nhật lịch sử tin nhắn vào MongoDB

    asyncio.run(update_message("B22DCCN542","lsd",mongo_mess_history))
```

Kế hoạch tuần tới:

- Hoàn thiện tính năng tóm tắt nội dung cho chatbot
- Thêm tính năng nhắc lịch học, tự động