DBMS 및 SQL 활용 #2

▼ 문제

- FastAPI 기반 /register_design API를 구현해보세요(Python)
- Streamlit 를 통해 입력 UI를 만들고 위에 만든 FastAPI를 호출하는 방식으로 해보세요.

아래의 순서대로 진행해보세요.
1. PostgreSQL에 `design` 테이블 생성 (이미 완료되셨을 거고요..)
2. FastAPI 서버 실행: `uvicorn app:app --reload`
3. Streamlit 클라이언트 실행: `streamlit run streamlit_client.py`
4. 입력 → POST → 등록 확인

▼ 코드

▼ 전체 코드

#1 FastAPI 서버

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import psycopg2
from dotenv import load_dotenv
from openai import OpenAI
import os

# 경로 설정
os.chdir("/Users/yshmbid/Documents/home/github/SQL")

# .env 로드
load_dotenv()

# OpenAI 클라이언트
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

# DB 연결
conn = psycopg2.connect(
    host="localhost",
    port=5432,
    database="postgres",
    user="postgres",
    password=os.getenv("PG_PASSWORD"),
)
cursor = conn.cursor()

# FastAPI 앱 객체 생성
app = FastAPI()

# 요청 데이터 모델 정의
class DesignRequest(BaseModel):
    description: str

# 임베딩 함수
def get_embedding(text: str):
    response = client.embeddings.create(
```

```
        model="text-embedding-3-small",
        input=text
    )
    return response.data[0].embedding

@app.post("/register_design")
def register_design(req: DesignRequest):
    try:
        cursor.execute("BEGIN;")
        embedding = get_embedding(req.description)

        cursor.execute(
            "INSERT INTO design (description, embedding) VALUES (%s, %s)",
            (req.description, embedding)
        )
        conn.commit()
        return {"status": "success", "message": "등록 성공"}
    except Exception as e:
        conn.rollback()
        raise HTTPException(status_code=500, detail=f"등록 실패: {e}")
```

#2 Streamlit 클라이언트

```
import streamlit as st
import requests
import os

# 경로 설정
os.chdir("/Users/yshmbid/Documents/home/github/SQL")

st.title("Design 등록 클라이언트")

# 입력 박스
description = st.text_area("설계안 입력", "")

if st.button("등록하기"):
    if description.strip() == "":
        st.warning("설계안을 입력해주세요.")
    else:
        try:
            response = requests.post(
                "http://127.0.0.1:8000/register_design",
                json={"description": description}
            )
            if response.status_code == 200:
                st.success(response.json())
            else:
                st.error(response.json())
        except Exception as e:
            st.error(f"서버 연결 실패: {e}")
```

▼ FastAPI - class DesignRequest

```
# FastAPI 앱 객체 생성
app = FastAPI()
```

```
# 요청 데이터 모델 정의
class DesignRequest(BaseModel):
    description: str
```

- app
  - FastAPI 애플리케이션 인스턴스
- DesignRequest(BaseModel)
  - BaseModel
    - json 형태 요청 데이터를 자동으로 python 객체로 변환하는 클래스
- description: str
  - ‘json 형태 요청 데이터’ 형태가 str.

▼ FastAPI - register_design()

```
@app.post("/register_design") # 요청이 오면 register_design를 실행
def register_design(req: DesignRequest):
    try:
        cursor.execute("BEGIN;") # 트랜잭션 시작
        embedding = get_embedding(req.description) # 요청(json)에서 꺼낸 텍스트의 임베딩 벡터

        cursor.execute(
            "INSERT INTO design (description, embedding) VALUES (%s, %s)",
            (req.description, embedding) # design 테이블에 req.description과 embedding을 넣음
        )
        conn.commit() # 커밋
        return {"status": "success", "message": "등록 성공"}
    except Exception as e:
        conn.rollback() # 롤백
        raise HTTPException(status_code=500, detail=f"등록 실패: {e}")
```

- @app.post("/register_design")
  - POST 방식으로 http://내서버주소:8000/register_design에 요청이 오면 register_design를 실행
- register_design(req: DesignRequest)
  - 요청에 들어온 JSON 데이터 req를 자동으로 DesignRequest라는 모델로 바꿔서 req라는 변수로 넣는다
- cursor.execute("INSERT INTO design (description, embedding) VALUES (%s, %s)", (req.description, embedding))
  - design 테이블에 description, embedding 컬럼으로 req.description과 embedding을 넣는다
    - req.description
      - 클라이언트가 보낸 요청(json)에서 꺼낸 텍스트
    - embedding

```
def get_embedding(text: str):
    response = client.embeddings.create(
        model="text-embedding-3-small",
        input=text
    )
    return response.data[0].embedding
```

```
embedding = get_embedding(req.description)
```

- 클라이언트가 보낸 description 즉 요청(json)에서 꺼낸 텍스트의 임베딩 벡터

- conn.commit()
  - 지금까지 트랜잭션 안에서 실행된 SQL을 DB에 영구 반영
- return {"status": "success", "message": "등록 성공"}
  - 클라이언트(Streamlit)로 json 성공 응답 보냄
- except Exception as e: conn.rollback()
  - 트랜잭션 중간에 에러가 나면 지금까지 했던 SQL 변경을 취소
  - 클라이언트(Streamlit)로 json 실패 응답 보냄

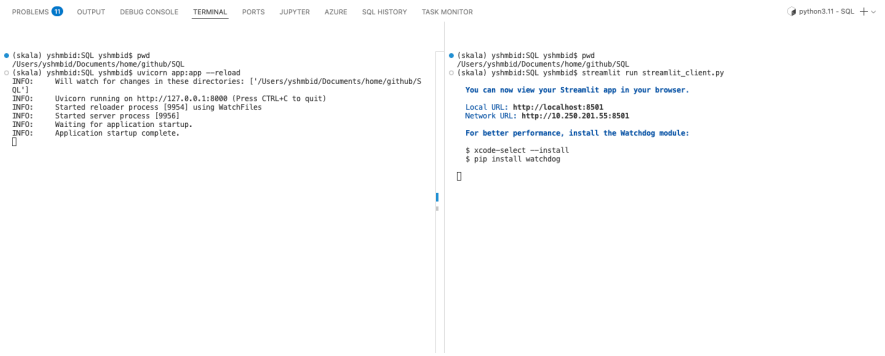
▼ Streamlit

```
st.title("Design 등록 클라이언트")

# 입력 박스
description = st.text_area("설계안 입력", "")

if st.button("등록하기"):
    if description.strip() == "":
        st.warning("설계안을 입력해주세요.")
    else:
        try:
            response = requests.post(
                "http://127.0.0.1:8000/register_design",
                json={"description": description}
            )
            if response.status_code == 200:
                st.success(response.json())
            else:
                st.error(response.json())
        except Exception as e:
            st.error(f"서버 연결 실패: {e}")
```

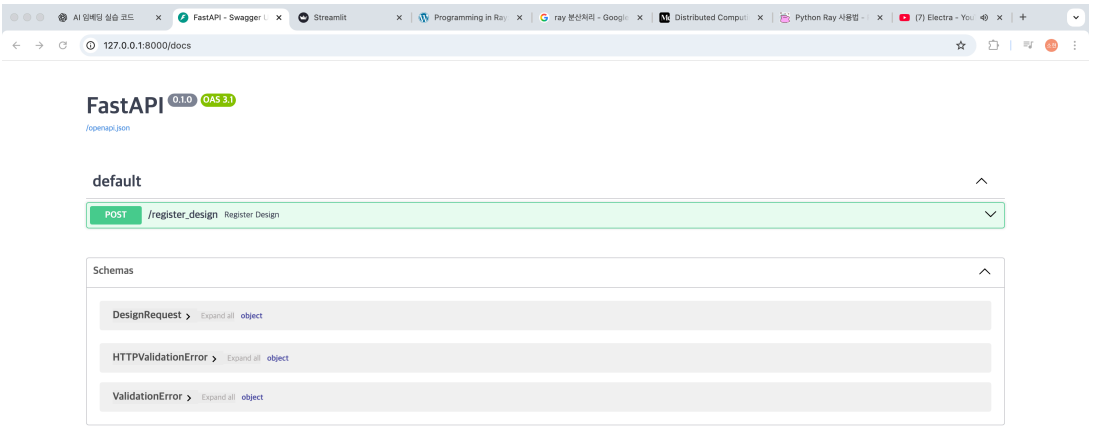
- st.button("등록하기")
  - 사용자 입력 문자열이 description에 이 들어옴
- response = requests.post("http://127.0.0.1:8000/register_design", json={"description": description})
  - 로컬 머신에서 실행 중인 FastAPI 서버 "http://127.0.0.1:8000/register_design" 연결, description을 json으로 전송
  - if response.status_code == 200: st.success(response.json())
    - http 응답에서 상태 코드 200은 요청 성공
    - 성공시 성공 메시지 st.success 출력
    - 실패시 실패 메시지 st.error 출력
- except Exception as e
  - 서버 연결 실패 시 st.error(f"서버 연결 실패: {e}")

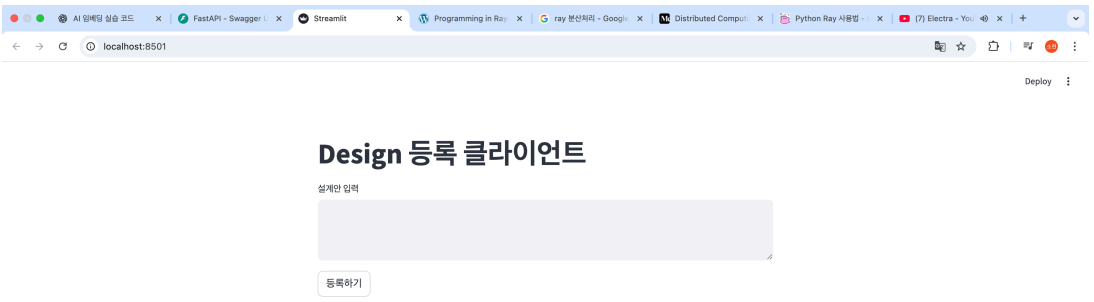
▼ 실행 및 실행 결과

- 실행


PROBLEMS 11
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
JUPYTER
AZURE
SQL HISTORY
TASK MONITOR
python3.11 - SQL
(skala) yshmbid:SQL yshmbid$ pwd
/Users/yshmbid/Documents/home/github/SQL
(skala) yshmbid:SQL yshmbid$ uvicorn app:app --reload
INFO: Will watch for changes in these directories: ['/Users/yshmbid/Documents/home/github/SQL']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [9954] using WatchFiles
INFO: Started server process [9956]
INFO: Waiting for application startup.
INFO: Application startup complete.
(skala) yshmbid:SQL yshmbid$ pwd
/Users/yshmbid/Documents/home/github/SQL
(skala) yshmbid:SQL yshmbid$ streamlit run streamlit_client.py
You can now view your Streamlit app in your browser.
Local URL: http://localhost:8501
Network URL: http://10.250.201.55:8501
For better performance, install the Watchdog module:
$ xcode-select --install
$ pip install watchdog

- 실행 결과


127.0.0.1:8000/docs
FastAPI
0.1.0
OAS 3.1
/openapi.json
default
POST
/register_design
Register Design
Schemas
DesignRequest
HTTPValidationError
ValidationError
Expand all
object


localhost:8501
Deploy
Design 등록 클라이언트
설계안 입력
등록하기

▼ 실습 시나리오 구현

- FastAPI 서버- class DesignRequest
  - Input: 사용자가 Streamlit 화면에서 입력한 설계안 텍스트를 json {"description": "텍스트"} 로 변환
  - Pydantic이 json을 검증후 python 객체(req.description)로 변환
  - Output: req.description (문자열)
- FastAPI 서버- register_design()
  - Input: req.description (문자열)
  - OpenAI API 호출해서 임베딩 벡터 생성 → PostgreSQL design 테이블에 (description, embedding) 저장
  - Output: 성공/실패 메시지 JSON 응답 ({"status": "success", "message": "등록 성공"})
- Streamlit
  - Input: 사용자가 입력 설계안 description 텍스트
  - FastAPI에 전송하면 json {"description": "텍스트"} 로 감싸서 fastapi에 POST 요청 → description 데이터 등록
  - Output: 성공 실패 메시지 표시

▼ class DesignRequest-register_design과 롤백

- class DesignRequest와 register_design()와의 호환?
  - JSON을 파싱해서 Python 객체로 바꾸고 description이 문자열인지 검증한 뒤 통과하면 register_design()에서 DesignRequest 객체를 만들어 req에 넣는다.
- 롤백?
  - rollback을 안 하면 “INSERT는 됐는데 commit 전에 에러 발생” 같은 상태가 DB에 남을 수 있음.