```
.
├── app
│   ├── __init__.py
│   ├── main.py
│   ├── database.py
│   └── models.py
├── Pipfile
└── test.db
```

• **app/init.py**: 패키지로 인식되기 위해 빈 파일로 둡니다.

• **app/database.py**: 데이터베이스 연결 설정 및 세션 생성.

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "sqlite:///./test.db"  # SQLite 데이터베이스 URL 설정

engine = create_engine(DATABASE_URL, connect_args=
{"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()

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

• **app/models.py**: 데이터베이스 모델 정의.

```
from sqlalchemy import Column, Integer, String
from .database import Base
```

```
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)
```

• **app/main.py**: FastAPI 애플리케이션 설정 및 경로 정의.

> 이 코드에 ORM 으로 만든 INSERT와 SELECT역할을 하는 함수가 존재합니다.

```
from fastapi import FastAPI, Depends, HTTPException, status
from sqlalchemy.orm import Session
from . import models, database

app = FastAPI()

# 데이터베이스의 테이블을 생성합니다.
models.Base.metadata.create_all(bind=database.engine)

@app.post("/users/", status_code=status.HTTP_201_CREATED)
def create_user(name: str, email: str, db: Session =
Depends(database.get_db)):
    # INSERT 쿼리: 새로운 사용자를 데이터베이스에 추가합니다.
    db_user = db.query(models.User).filter(models.User.email ==
email).first()
    if db_user:
        raise HTTPException(status_code=400, detail="Email already
registered")
    new_user = models.User(name=name, email=email)
    db.add(new_user)
    db.commit()
    db.refresh(new_user)
    return new_user

@app.get("/users/")
def read_users(skip: int = 0, limit: int = 10, db: Session =
Depends(database.get_db)):
    # SELECT 쿼리: 데이터베이스에서 사용자 목록을 조회합니다.
```

```
    users = db.query(models.User).offset(skip).limit(limit).all()
    return users
```

구현 코드 설명(과제)

이 코드는 FastAPI와 SQLAlchemy를 사용한 사용자 추가, 탐색 코드입니다. SQLite를 데이터베이스로 사용했으며, 사용자 정보를 저장하고 조회하는 기능을 구현했습니다.

- **사용자 생성 (INSERT 쿼리)**:

- 새로운 사용자를 데이터베이스에 추가합니다.

- `POST /users/` 엔드포인트를 통해 구현됩니다.

- 입력된 이름과 이메일을 기반으로 새로운 `User` 객체를 생성하고 데이터베이스에 저장합니다.

- 예시:

  ```json
  {
    "name": "John Doe",
    "email": "johndoe@example.com"
  }
  ```

- **사용자 조회 (SELECT 쿼리)**:

- 데이터베이스에서 사용자 목록을 조회합니다.

- `GET /users/` 엔드포인트를 통해 구현됩니다.

- 데이터베이스에서 `User` 테이블의 내용을 조회하여 반환합니다.

구현된 쿼리

- **INSERT 쿼리**:

- `main.py` 파일의 `create_user` 함수 내에서 구현됩니다.

- 사용자 이름과 이메일을 입력받아 데이터베이스에 새로운 사용자로 추가합니다.

- 구현 코드:

  ```python

  new_user = models.User(name=name, email=email)

  db.add(new_user)

  db.commit()

  db.refresh(new_user)

  ```

- **SELECT 쿼리**:

- `main.py` 파일의 `read_users` 함수 내에서 구현됩니다.

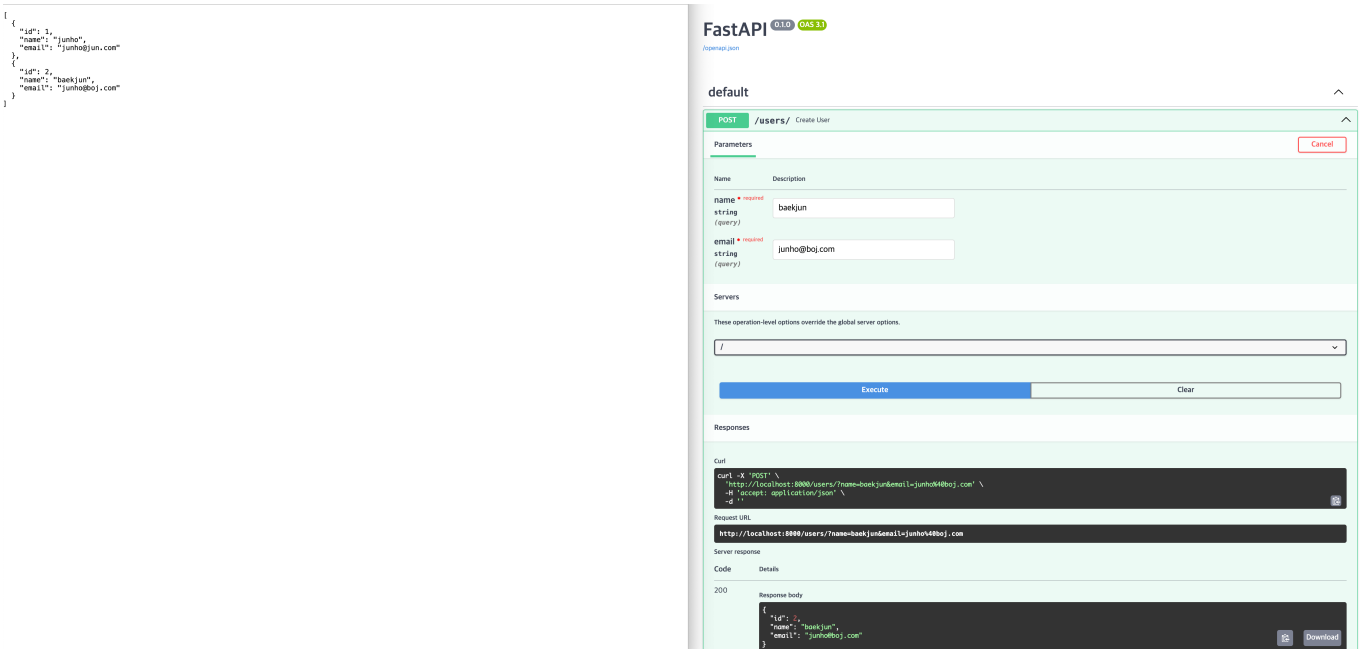
- 사용자 목록을 데이터베이스에서 조회하여 반환합니다.

- 구현 코드:

  ```python

  users = db.query(models.User).offset(skip).limit(limit).all()

  ```



- fastapi 서버를 띄우고 swaggerUI로 post(INSERT) 요청 후 /user/ get(SELECT)으로 탐색

```
❯ sqlite3 test.db
SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
sqlite> .tables
]alembic_version  users
sqlite> .tables
alembic_version  users
sqlite> .schema users
CREATE TABLE users (
        id INTEGER NOT NULL,
        name VARCHAR,
        email VARCHAR,
        PRIMARY KEY (id)
);
CREATE UNIQUE INDEX ix_users_email ON users (email);
CREATE INDEX ix_users_id ON users (id);
CREATE INDEX ix_users_name ON users (name);
sqlite> SELECT * FROM users;
1|junho|junho@jun.com
sqlite> sqlite> SELECT * FROM users;
1|junho|junho@jun.com
2|baekjun|junho@boj.com
```

- 터미널에서 sqllit3 를 실행하고 데이터를 조회

느낀점

ORM(Object Relational Mapping)을 사용하면, 프로그래밍 언어의 객체를 데이터베이스 테이블과 연결하여 데이터베이스 작업을 쉽게 할 수 있습니다. 직접 SQL 쿼리를 작성하는 방식은 보안에 취약하고, 데이터베이스 구조가 변경되면 수정하기 어렵지만, ORM을 사용하면 이런 문제를 해결할 수 있습니다. 이러한 이유로, 앞으로의 프로젝트에서는 ORM을 사용하는 것이 훨씬 더 효율적이고 안전하다고 느꼈습니다.