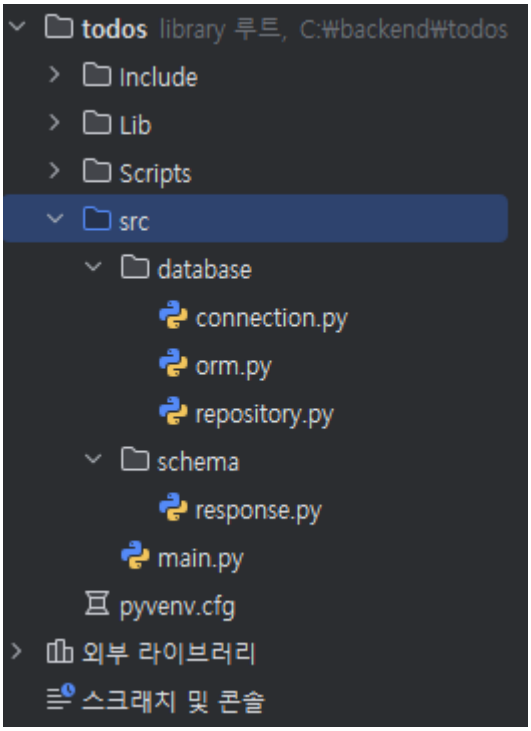
백엔드 week 3 추가 발표자료

todos 프로젝트 디렉토리 구조




```
C:\backend\todos\src
├── database/
│   ├── connection.py
│   ├── orm.py
│   └── repository.py
│
├── schema/
│   └── response.py
│
└── main.py
```

이 프로젝트는 FastAPI(백엔드 프레임워크)로 구성된 ToDo DB에 대한 API이다. 데이터베이스 관련 코드는 `database` 폴더에, 스키마 정의는 `schema` 폴더에 구성되어 있고, `main.py` 가 애플리케이션의 진입점 역할을 하고 있다.

main.py

```python
from typing import List

from database.orm import ToDo
from database.repository import get_todos, get_todo_by_todo_id
```

```python
from fastapi import FastAPI, Body, HTTPException, Depends
from pydantic import BaseModel
from schema.response import ListToDoResponse, ToDoSchema
from sqlalchemy.orm import Session
from database.connection import get_db

app = FastAPI()

@app.get("/")
def health_check_handler():
    return {"ping": "pong"}

todo_data = {
    1: {
        "id" : 1,
        "contents": "실전! FastAPI 섹션 0 수강",
        "is_done": True,
    },
    2: {
        "id": 2,
        "contents": "실전! FastAPI 섹션 1 수강",
        "is_done": False,
    },
    3: {
        "id": 3,
        "contents": "실전! FastAPI 섹션 2 수강",
        "is_done": False,
    },
}

@app.get("/todos", status_code=200)
def get_todos_handler(
        order: str | None = None,
        session: Session = Depends(get_db),
) -> ListToDoResponse:
    todos: List[ToDo] = get_todos(session=session)

    if order and order == "DESC":
        return ListToDoResponse(
            todos=[ToDoSchema.from_orm(todo) for todo in todos[::-1]]
        )
    return ListToDoResponse(
        todos=[ToDoSchema.from_orm(todo) for todo in todos]
    )

@app.get("/todos/{todo_id}", status_code=200)
def get_todo_handler(
        todo_id: int,
        session: Session = Depends(get_db),
) -> ToDoSchema:
```

```
    todo: ToDo | None = get_todo_by_todo_id(session=session,
todo_id=todo_id)
    if todo:
        return ToDoSchema.from_orm(todo)
    raise HTTPException(status_code=404, detail="ToDo Not Found")


# pydentic을 이용한 리퀘스트 바디 정의
class CreateToDoRequest(BaseModel):
    id: int
    contents: str
    is_done: bool



@app.post("/todos", status_code=201)
def create_todo_handler(request: CreateToDoRequest):
    todo_data[request.id] = request.dict()
    return todo_data[request.id]


@app.patch("/todos/{todo_id}", status_code=200)
def update_todo_handler(
    todo_id: int,
    is_done: bool = Body(..., embed=True),
):
    todo = todo_data.get(todo_id)
    if todo:
        todo["is_done"] = is_done
        return todo
    raise HTTPException(status_code=404, detail="ToDo Not Found")


@app.delete("/todos/{todo_id}", status_code=204)
def delete_todo_handler(todo_id: int):
    todo = todo_data.pop(todo_id, None)
    if todo:
        return
    raise HTTPException(status_code=404, detail="ToDo Not Found")
```

---

database/connection.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "mysql+pymysql://root:todos@127.0.0.1:3306/todos"
```

```
engine = create_engine(DATABASE_URL, echo=True)
SessionFactory = sessionmaker(autocommit=False, autoflush=False,
bind=engine)

def get_db():
    session = SessionFactory()
    try:
        yield session
    finally:
        session.close()
```

database/orm.py

```
from sqlalchemy import Boolean, Column, Integer, String
from sqlalchemy.orm import declarative_base

Base = declarative_base()


class ToDo(Base):
    __tablename__ = "todo"

    id = Column(Integer, primary_key=True, index=True)
    contents = Column(String(256), nullable=False)
    is_done = Column(Boolean, nullable=False)

    def __repr__(self):
        return f"ToDo(id={self.id}, contents={self.contents}, is_done=
{self.is_done})"
```

database/repository.py

```
from typing import List

from sqlalchemy import select
from sqlalchemy.orm import Session
from database.orm import ToDo

def get_todos(session: Session) -> List[ToDo]:
    return list(session.scalars(select(ToDo)))

def get_todo_by_todo_id(session: Session, todo_id: int) -> ToDo | None:
    return session.scalar(select(ToDo).where(ToDo.id == todo_id))
```

schema/response.py

```python
from typing import List

from pydantic import BaseModel


class ToDoSchema(BaseModel):
    id: int
    contents: str
    is_done: bool

    class Config:
        orm_mode = True


class ListToDoResponse(BaseModel):
    todos: List[ToDoSchema]
```

전체 코드에 대한 정리

1. 데이터베이스 연결 설정 (database/connection.py)

```python
engine = create_engine(DATABASE_URL, echo=True)
```

- SQLAlchemy로 DB 접속 위해서는 먼저 엔진이라는 개체를 만들어줘야 함
- echo=True: sqlalchemy에 의해 대신 처리되는 쿼리에 어떤 sql이 사용됐는지 사용되는 시점에 사용되는 sql을 출력해주는 옵션(실제 프로덕션 X, 개발 환경 디버깅 시)

```python
SessionFactory = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
```

- session을 생성해서 인스턴스를 통해 데이터베이스와 통신
- 커밋, 플러쉬 동작(연산)을 자동으로 수행시키지 않고 명시적으로 하겠다
- bind로 앞서 생성한 engine 연결, 전달한 DB URI를 통해 엔진이라는 객체가 생성됨. 그 엔진을 사용해 실제로 session을 만들 수 있게 된다.

```python
def get_db():
    session = SessionFactory()
    try:
        yield session
```

```
    finally:
        session.close()
```

- Dependency Injection을 위한 함수
- FastAPI의 Depends()와 함께 사용되어 각 API 요청마다 DB 세션을 생성하고 요청 완료 후 자동으로 닫아줌

2. ORM 모델 정의 (database/orm.py)

```
Base = declarative_base()
```

- SQLAlchemy의 declarative base 클래스 생성
- 모든 ORM 모델이 상속받을 기본 클래스

```
class ToDo(Base):
    __tablename__ = "todo"
    id = Column(Integer, primary_key=True, index=True)
    contents = Column(String(256), nullable=False)
    is_done = Column(Boolean, nullable=False)
```

- 데이터베이스의 'todo' 테이블과 매핑되는 ORM 클래스
- Python 객체로 DB 테이블의 행(row)을 표현

3. 리포지토리 패턴 (database/repository.py)

```
def get_todos(session: Session) -> List[ToDo]:
    return list(session.scalars(select(ToDo)))
```

- 데이터베이스 접근 로직을 캡슐화한 함수
- SQLAlchemy의 select() 함수로 모든 ToDo 레코드 조회
- session.scalars()로 쿼리 결과를 실제 객체로 변환

```
def get_todo_by_todo_id(session: Session, todo_id: int) -> ToDo | None:
    return session.scalar(select(ToDo).where(ToDo.id == todo_id))
```

- 특정 ID의 ToDo를 조회하는 함수
- WHERE 절을 사용하여 조건부 쿼리 수행

4. 응답 스키마 정의 (schema/response.py)

```
class ToDoSchema(BaseModel):
    id: int
    contents: str
```

```
    is_done: bool

    class Config:
        orm_mode = True
```

- Pydantic 모델로 API 응답 데이터 구조 정의
- orm_mode = True: SQLAlchemy ORM 객체를 Pydantic 모델로 자동 변환 가능
- from_orm() 메서드 사용 가능하게 함

```
class ListToDoResponse(BaseModel):
    todos: List[ToDoSchema]
```

- 여러 개의 ToDo를 담는 리스트 응답 구조 정의

5. API 엔드포인트 및 요청 흐름 (main.py)

GET /todos 요청 흐름:

```
@app.get("/todos", status_code=200)
def get_todos_handler(
        order: str | None = None,
        session: Session = Depends(get_db),
) -> ListToDoResponse:
```

1단계: 의존성 주입

- `Depends(get_db)` : FastAPI가 get_db() 함수를 호출하여 DB 세션을 생성하고 함수 파라미터로 전달

2단계: 리포지토리 호출

```
todos: List[ToDo] = get_todos(session=session)
```

- repository.py의 get_todos() 함수 호출
- 실제 SQL 쿼리 실행: `SELECT * FROM todo`

3단계: 응답 데이터 변환

```
return ListToDoResponse(
    todos=[ToDoSchema.from_orm(todo) for todo in todos]
)
```

- SQLAlchemy ORM 객체(ToDo)를 Pydantic 모델(ToDoSchema)로 변환
- from_orm() 메서드가 자동으로 ORM 객체의 속성을 Pydantic 모델에 매핑

GET /todos/{todo_id} 요청 흐름:

```
@app.get("/todos/{todo_id}", status_code=200)
def get_todo_handler(
        todo_id: int,
        session: Session = Depends(get_db),
) -> ToDoSchema:
```

1단계: 경로 매개변수 추출

- FastAPI가 URL에서 todo_id 자동 추출 및 int 타입 검증

2단계: 데이터베이스 조회

```
todo: ToDo | None = get_todo_by_todo_id(session=session, todo_id=todo_id)
```

- repository의 get_todo_by_todo_id() 함수 호출
- 실제 SQL 쿼리: `SELECT * FROM todo WHERE id = ?`

3단계: 응답 처리

```
if todo:
    return ToDoSchema.from_orm(todo)
raise HTTPException(status_code=404, detail="ToDo Not Found")
```

- 데이터 존재 시: ORM 객체를 Pydantic 모델로 변환하여 반환
- 데이터 없을 시: 404 에러 반환

전체 요청-응답 흐름 정리:

1. **클라이언트 요청** → FastAPI 애플리케이션
2. **FastAPI 라우팅** → 적절한 핸들러 함수 호출
3. **의존성 주입** → `Depends(get_db)` 로 DB 세션 생성
4. **비즈니스 로직** → repository 함수 호출
5. **데이터베이스 쿼리** → SQLAlchemy ORM을 통한 SQL 실행
6. **데이터 변환** → ORM 객체 → Pydantic 모델
7. **응답 반환** → JSON 형태로 클라이언트에 전송
8. **리소스 정리** → DB 세션 자동 종료

이런 구조로 FastAPI, SQLAlchemy, Pydantic이 유기적으로 연결되어 API 서버가 동작