FastAPI - w1

Hong Namsoo

Before to start

- I used poetry for package management
- I used project structure like below

```
├── README.md  # readme for project
├── fastapi_study  # actual package (actual project root)
│   ├── __init__.py
│   ├── core  # core module
│   │   └── exceptions.py  # custom exceptions
│   ├── database.py  # database connection
│   ├── models  # sqlalchemy models for database
│   │   ├── __init__.py  # model base
│   │   └── todo.py  # todo model
│   ├── routers  # fastapi routers for api
│   │   └── todo.py  # todo router
│   ├── schemas  # pydantic schemas for api
│   │   ├── errors.py  # error schemas
│   │   └── todo.py  # todo schemas
│   └── services  # business logic
│       └── todo.py  # todo service
├── main.py  # fastapi app
├── poetry.lock
├── pyproject.toml
├── requirements.txt
└── tests
    └── __init__.py
```

INDEX

1. `/todos`

graph LR; A[Client]-->B[FastAPI Server]; B-->C[Router]; C-->D[todos GET]; C-->E[todos POST]; C-->F[todos/id GET]; C-->G[todos/id PATCH]; D-->H[ToDoService]; E-->H; F-->H; G-->H; H-->I[ToDoModel]; I-->J[DB];

1.1. /todos - Router

- FastAPI는 특정API들을 처리하는 APIRouter라는 개념을 사용할 수 있음
- 이를 통해 API들을 논리적으로 그룹화할 수 있음

```
"""
routers/todo.py  /todos 하위 로직을 처리하는 라우터를 정의
"""
from typing import Annotated  # FastAPI에서 추천하는 방식

from fastapi import APIRouter, Depends, HTTPException
from fastapi import Body, Path, Query
from sqlalchemy.orm import Session
...
router = APIRouter()  # 새로운 라우터 생성
```

```
"""
main.py  FastAPI app에 /todos 라우터를 추가
"""
from fastapi_study.routers import todo
...
app.include_router(todo.router, prefix="/todos")
```

2.2. /todos - API

```python
from fastapi_study.services import todo as todo_service  # 비즈니스 로직

@router.get(
    "",  # /todos[GET]
    tags=["todo"],  # swagger에서 보여질 태그
    response_model=list[ToDoRead],  # 본 API의 응답 모델
    responses={  # 본 API의 응답 코드 예시
        400: {
            "description": "잘못된 요청",
            "model": ErrorMsg,
        },
    }
)
async def get_todos(
    db: Annotated[Session, Depends(get_db)]  # FastAPI에서 추천하는 방식
):
    return todo_service.get_all_todos(db)  # 비즈니스 로직 호출

@router.post(
    "",  # /todos[POST]
    tags=["todo"],  # swagger에서 보여질 태그
    response_model=ToDoRead  # 본 API의 응답 모델
)
async def create_todo(
    new_todo: ToDoCreate,  # 본 API의 요청 모델(pyadantic)
    db: Annotated[Session, Depends(get_db)]  # FastAPI에서 추천하는 방식
):
    return todo_service.create_new_todo(new_todo, db)  # 비즈니스 로직 호출
```

2.3. /todos - Service

```python
from fastapi_study.models.todo import ToDo as ToDoModel  # SQLAlchemy Model
from fastapi_study.schemas.todo import ToDoCreate as ToDoCreateSchema  # Pydantic Schema

from sqlalchemy import select  # SQLAlchemy의 select 함수
from sqlalchemy.orm import Session  # SQLAlchemy의 Session

def get_all_todos(db: Session) -> list[ToDoModel]:
    """
    모든 ToDo를 조회합니다.
    """

    stmt = select(ToDoModel)  # SQLAlchemy의 select를 이용하여 statement 생성
    return db.scalars(stmt).all()  # statement를 실행하여 DB에서 데이터를 가져옴
```

3. Migrating to SQLAlchemy2 - Base Model 상속

```
"""1.x 버전"""
from sqlalchemy import Integer, String, Boolean
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)
    user_id = Column(Integer, ForeignKey("user.id"))
```

3. Migrating to SQLAlchemy2 - Base Model 상속

```
"""2.x 버전"""
from sqlalchemy.orm import DeclarativeBase
from sqlalchemy import Integer, String, Boolean
from sqlalchemy.orm import mapped_column, Mapped

class Base(DeclarativeBase):
    pass

class ToDo(Base):
    __tablename__ = "todo"

    # Mapped[int]는 타입 힌트를 위한 것
    # mapped_column은 SQLAlchemy에서 명시적으로 orm을 사용하는 컬럼을 위한 것
    id: Mapped[int] = mapped_column(primary_key=True)
    contents: Mapped[str] = mapped_column(String(256), nullable=False)
    is_done: Mapped[bool] = mapped_column(Boolean, nullable=False, default=False)
```

3. Migrating to SQLAlchemy2 - Session 관련 query 사용법 변경

	1.x	2.x	비고
	`session.query(User).get(42)`	`session.get(User, 42)`	
	`session.query(User).all()`	`session.scalars(select(User)).all()`	
	`session.query(User).filter_by(name="some user").one()`	`session.execute(select(User).filter_by(name="some user")).scalar_one()`	

4. Migrating to Pydantic2

1.x	2.x	비고
`__fields__`	`model_fields`	
`copy()`	`model_copy()`	중요
`dict()`	`model_dump()`	
`json_schema()`	`model_json_schema()`	
`json()`	`model_dump_json()`	
`parse_obj()`	`model_validate()`	
`orm_mode`	`from_attributes`	중요
`@validator`	`@field_validator`	중요
`@root_validator`	`@model_validator`	중요