FastAPI: роутеры и CRUD-операции | Я.Шпора

Роутеры

Роутеры в FatsAPI используются для организации и группировки view-функций, связанных с запрошенным URL.

Пример роутера в файле *app/api/endpoints.py*

```
from fastapi import APIRouter


router = APIRouter(
   prefix='/the_common',  # Начальный "/" обязателен, конечный "/" запрещён.
   tags=['общие теги',],  # Тег для документации.
)


@router.post('/the_path')
def the_view(...):
    ...
```

При использовании этого роутера функция `the_view()` будет обрабатывать POST-запросы на URL вида */the_common/the_path*.

Подключение роутера к объекту приложения производится в файле *main.py*

```python
from fastapi import FastAPI

from app.api.endpoints import router


app = FastAPI()
app.include_router(router)
```

CRUD-операции

Последовательность обработки любого HTTP-запроса в FastAPI:

- Получение JSON с данными из запроса.
- Валидация JSON средствами Pydantic и преобразование JSON в Python-объект с помощью Pydantic-объекта (схемы).
- Преобразование полученного Python-объекта в объект модели ORM SQLAlchemy.
- Возврат ответа; при необходимости происходит обращение в БД для получения информации о возвращаемом объекте.

Функция-обработчик (CRUD-функция) принимает объект Pydantic-схемы и возвращает объект ORM-модели. В работе удобно создавать классы, которые содержат все необходимые CRUD-методы для одной модели.

Для CRUD-функций помимо объекта Pydantic-модели понадобится объект сессии для работы с БД. Для получения сессии можно применить генератор сессии. Обычно его размещают в файле *app/core/db.py*

```
...
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
from sqlalchemy.orm import declarative_base, declared_attr, sessionmaker

...
AsyncSessionLocal = sessionmaker(engine, class_=AsyncSession)


# Асинхронный генератор сессий.
async def get_async_session():
    # Через асинхронный контекстный менеджер и sessionmaker
    # открывается сессия.
    async with AsyncSessionLocal() as async_session:
        # Генератор с сессией передается в вызывающую функцию.
        yield async_session
        # Когда HTTP-запрос отработает - выполнение кода вернётся сюда,
        # и при выходе из контекстного менеджера сессия будет закрыта.
```

Передача генератора сессий во view-функции осуществляется через механизм внедрения зависимостей. В FastAPI зависимости подключаются через класс `Depends`.

```
from fastapi import Depends

from app.core.db import get_async_session
```

```python
@router.post('/the_path/')
async def api_create_the_obj(
    jsc_obj: TheCreateSchema,
    session: AsyncSession = Depends(get_async_session),  # Внедрение зависимости
):
    # Использование сессии:
    obj_id = await вызов_crud_функции(jsc_obj.имя, session)
    if obj_id is not None: raise HTTPException(...)
    # Использование сессии:
    return await вызов_crud_функции(jsc_obj, session)
```

Создание объекта (Create)

Создание одного объекта:

```python
async def create(json_scheme_obj, session: AsyncSession):
    # Конвертация Pydantic-объекта в словарь:
    db_obj = Model(**json_scheme_obj.dict())
    session.add(db_obj)
    await session.commit()
    await session.refresh(db_obj)
    return db_obj
```

Чтение одного или нескольких объектов (Read)

Получение одного объекта из базы данных по заданному `obj_id`:

```
async def get(obj_id: int, session: AsyncSession):
    db_obj = await session.execute(select(Model).where(Model.id == obj_id))
    return db_obj.scalars().first()
```

Получение всех объектов заданной модели:

```
async def get_multi(session: AsyncSession):
    db_objs = await session.execute(select(Model))
    return db_objs.scalars().all()
```

Обновление объекта (Update)

Обновление отдельного объекта:

```
async def update(db_obj, json_scheme_obj, session: AsyncSession):
    obj_data = jsonable_encoder(db_obj)
    up_data = json_scheme_obj.dict(exclude_unset=True)
    for field in obj_data:
        if field in up_data:
            setattr(db_obj, field, up_data[field])
    session.add(db_obj)
```

```
    await session.commit()
    await session.refresh(db_obj)
    return db_obj
```

Удаление объекта (Delete)

Удаление отдельного объекта:

```
async def remove(db_obj, session: AsyncSession):
    await ses.delete(db_obj)
    await ses.commit()
    return db_obj
```