FastAPI: базы данных и миграции | Я.Шпора

Для асинхронного подключения к SQLite нужно установить драйвер `aiosqlite`.

```
pip install aiosqlite==0.17.0
```

В качестве ORM можно применить SQLAlchemy:

```
pip install sqlalchemy==1.4.36
```

Настройки подключения обычно хранят в классе `Settings`, унаследованный от класса `BaseSettings` из библиотеки `pydantic`. Традиционно его располагают в файле *app/core/config.py*. Класс можно расположить в любом удобном месте.

```
from pydantic import BaseSettings


class Settings(BaseSettings):
        ...
    database_url: str
        ...

        class Config:
                env_file = '.env'
```

```
settings = Settings()
```

Создавать *.env*-файл не обязательно, можно использовать значения по умолчанию в полях класса настроек. Но в целях большей безопасности принято убирать все критичные данные в переменные окружения.

Строка для подключения к базе данных SQLite файле *.env*:

```
DATABASE_URL=sqlite+aiosqlite:///./the_app.db
```

- `sqlite+aiosqlite://` — это указание типа базы данных. В примере используется SQLite и `aiosqlite` драйвер для взаимодействия с SQLite в асинхронном режиме.
- `./the_app.db` — путь к файлу базы данных. В примере файл называется `the_app.db` и расположен в текущей директории (представленной `./`).

Базовый класс Base

Для подключения к базе данных необходимо создать базовый класс `Base`, от которого будут наследоваться все модели проекта. Обычно его размещают в файле *app/core/db.py*.

В примере кода показан класс `PreBase`, на основе которого создан класс `Base`. Класс `PreBase` описывает общее поведение для всех таблиц: примере он создаёт во всех наследниках-моделях поле типа PK с именем `id` и устанавливает способ именования таблиц в БД.

```
from sqlalchemy import Column, Integer
from sqlalchemy.ext.asyncio import AsyncSession, declared_attr, create_async_engine
```

```python
from sqlalchemy.orm import declarative_base, sessionmaker

from app.core.config import settings


class PreBase:
    @declared_attr
    def __tablename__(cls):
        return cls.__name__.lower()  # Трюк: имя таблицы создаётся из названия модели.
    id = Column(Integer, primary_key=True)  # Все таблицы будут содержать PK-поле id.

# Base - традиционное название для этого класса; такие традиции лучше соблюдать.
Base = declarative_base(cls=PreBase)  # Класс-родитель для моделей.

# Организация асинхронной работы:
engine = create_async_engine(settings.database_url)
AsyncSessionLocal = sessionmaker(engine, class_=AsyncSession)
# Асинхронный генератор сессий:
async def get_async_session():
    async with AsyncSessionLocal() as async_session:
        yield async_session
```

Модели обычно создаются в каталоге *app/models/*.

```
from sqlalchemy import Column, String, ForeignKey
from sqlalchemy.orm import relationship

from app.core.db import Base


class TheModel(Base):
    the_field = Column(String(100), unique=True, nullable=False)
    the_other_id = Column(Integer, ForeignKey('the_other.id'))


class TheOther(Base):
    the_model = relationship(TheModel, cascade='delete')
```

Другие типы данных для полей можно посмотреть в документации.

Миграции через Alembic

Для создания миграций можно применить библиотеку Alembic:

```
pip install alembic==1.7.7
```

Инициализация Alembic:

```
alembic init --template async alembic
```

Параметр `--template async` необходимо указывать в тех случаях, когда используется асинхронное подключение к БД.

В корне проекта появятся папка `alembic` и файл `alembic.ini`.

Для импорта моделей в файлы настроек Alembic есть смысл создать в приложении FastAPI файл и импортировать в него все модели проекта:

```
# Файл app/core/base.py
from app.core.db import Base
from app.models.the_models import TheModel
...  # Импорт всех остальных моделей проекта.
```

При настройке Alembic в файле *alembic/env.py* необходимо указать пути до всех моделей, используемых в проекте:

```
# Файл alembic/env.py
...
from dotenv import load_dotenv

from app.core.base import Base
...

load_dotenv('.env')
...
```

```
config = context.config
# Следующая строка должны быть ниже строки config = context.config
config.set_main_option('sqlalchemy.url', os.environ['DATABASE_URL'])
...

# Исправление значения target_metadata:
target_metadata = Base.metadata
```

При миграциях строка `target_metadata` указывает Alembic на те метаданные, которые он должен сравнить с текущей схемой базы данных для определения изменений, которые необходимо внести.

Создание миграции:

```
alembic revision --autogenerate -m "комментарий к миграции"
```

Выполнение всех неприменённых миграций:

```
alembic upgrade head
```

Отмена всех миграций, которые были в проекте:

```
alembic downgrade base
```

Применение всех миграций до указанной (указываем ID миграции):

```
alembic upgrade befcaa650c3f
```

В командах можно применять сокращённые ID миграции, важно лишь, чтобы сокращение было уникально. Можно выполнить команду `alembic upgrade be` или даже `alembic upgrade b`, если с буквы `b` начинается название только одного файла миграций. Если сокращённый ID неуникален — команда не выполнится и будет выведено сообщение об ошибке.

Откат миграций (отменить все миграции до миграции с ID 466f1da3d4b1):

```
alembic downgrade 466f1da3d4b1
```

В команде даунгрейда можно использовать сокращенные ID миграций.

Посмотреть историю миграций:

```
alembic history
```

Посмотреть последнюю применённую миграцию:

```
alembic current
```