План урока

- Объединим эндпоинты в роутер
- Напишем первые запросы через SQLAlchemy
- Разберемся, почему асинхронные запросы быстрее синхронных
- Выделим работу с БД в отдельный слой через паттерн Репозиторий/DAO

Выделение работы с базой данных в отдельный слой

В приложениях любого масштаба (от 1 до 100+ эндпоинтов) необходимо выделять работу с БД в отдельный слой или, на языке Python, в отдельные классы и файлы. Это позволяет не сваливать в кучу внутри эндпоинта и работу с БД, сторонними API, и валидацию данных и т.п. На это есть причины:

1. Мы хотим разбить наш проект на **небольшие, легко поддерживаемые участки кода**. Так, например, одна команда может заниматься построением слоя работы с БД или сторонним API, а другая создавать эндпоинты.
2. Внутри эндпоинта мы хотим абстрагироваться от привязки к конкретной БД или конкретному API. **Эндпоинт не должен знать ничего о том, из какого источника он получает данные** о списке отелей: это может быть Postgres, Mongo, закэшированные данные внутри Redis и т.д.
3. Мы хотим иметь возможность **переиспользовать часто используемые обращения к БД**. Например, в трех эндпоинтах нам нужны данные по конкретному отелю. Гораздо удобнее обратиться к классу, ответственному за работу с отелями, чем три раза писать один и тот же код, нарушая принцип DRY.
4. Мы хотим иметь **возможность тестировать отдельно работу БД**. Если мы оставим написание запроса к БД внутри эндпоинта, мы никогда не сможем протестировать поведение БД в изоляции от самого эндпоинта (который еще и валидирует данные, проверяет авторизацию пользователя и т.д.)

Хорошее видео о том, как проектируются бэкенд-приложения по паттерну MVC
Ссылка: https://www.youtube.com/watch?v=HpL6ymFEuu4

Особенности работы с SQLAlchemy в FastAPI

Если мы попросим FastAPI вернуть объект, который нам дала Алхимия, мы получим ошибку. FastAPI не сможет автоматически конвертировать модель Алхимии в JSON (FastAPI автоматически конвертирует любой ответ пользователю в JSON). Для сериализации модели SQLAlchemy в модель Pydantic и затем в JSON нам необходимо создать Pydantic схему, которая будет в точности отражать ответ от Алхимии. Важно отметить, что для работы с ответом Алхимии нам необходимо обращаться к атрибутам через точку: `Hotels.name` или `Hotels.location`, а не `Hotels["name"]` и `Hotels["location"]`. Для того, чтобы Pydantic мог обращаться к атрибутам модели Алхимии через точку (по

умолчанию Pydantic обращается только через `Hotels["..."]`, мы обязаны указать внутри Pydantic схемы параметр `orm_mode` со значением `True` следующим образом:

```
class Hotel(BaseModel):
    id: int
    location: str

    class Config:
        orm_mode = True
```