Ссылка на офф. руководство FastAPI.

Замечания к второй лабораторной работе

Важные замечания:

- Понадобилось пересмотреть файловую структуру проекта для того, чтобы всё заработало: теперь в app находится всё кроме папки alembic и main.py, в самой же папке app находится `__init__.py`, из-за которого python при импорте воспринимает его как модуль и может добраться до models.py и достать оттуда констурктор таблицы `User`.
- В файле `main.py` (или любое другое имя которым назван файл, который запускает север FastAPI) можно добавить код ниже и тогда можно запускать из консольной командой `python main.py` сервер. Только для этого нужно создать в папке app/api py-файл где лежат обычные константные переменные хранящие информацию о порте и ip сервера (туда же можно добавить эндпоинты):

```
    if __name__ == "__main__":
        uvicorn.run(app, host=FastApiServerInfo.IP,
port=FastApiServerInfo.PORT)
```

- Если появится желание (или нужда) добавить новую таблицу в базу данных которую смотрит alembic, то:
  - Редактируем файл где хранятся конструкторы структур таблиц, в моём случае этом `models.py`
  - Теперь в командной строке `alembic upgrade head` и `alembic revision --autogenerate -m "some test"`

Сервер - это `main.py`, запускать из терминала командой `uvicorn main:app --host 127.0.0.1 --port 12000`, где **server_script** - название py файла, **server_app** - название переменной которой присвоено FastAPI.

Alembic - по сути git, но для бд в рамках одного проекта

Для того чтобы добавить alemic в проект делаем следующее:

- Переходим через консоль в папку проекта и пишем `alembic init alembic`
- Создаётся папка alembic, в ней находим `alembic.ini`, ищем строку `sqlalchemy.url = ...` и пишем `sqlite:///(относительный путь к бд, которой желательно лежать в папке проекта)`
  Очень важно докинуть в папку app файл init.py (он может быть пустым), иначе python не будет понимать, что из app можно что-то экспортировать
- В файле `env.py` импортируем модель SQLAlchemy, например так `from app.models import (класс которым назван конструктор таблицы)` и редактируем строку `target_metadata = (класс которым назван конструктор таблицы).metadata`

Теперь заставляем это работать:

- В командной строке пишем `alembic revision --autogenerate -m "create users table"` - это создаст в alemic/versions py-файл. Его можно отредактировать (а имено методы **upgrade** и **downgrade**), если есть желание и умение. Это у нас создалась та самая миграция (или коммит).
- Всё в той же командной строке пишем `alembic upgrade head` для применение миграции (коммита) и `alembic downgrade -1` чтобы откатиться на предыдущую миграцию (коммит)

SQLAlchemy - автоматизация создания структур бд через python.

Пример кода ниже пишем в отедльном файле, а потом из него импортиуем нужный класс.

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

# Создаём базовый класс для моделей
Base = declarative_base()

# Определяем модель User
class User(Base):
    __tablename__ = 'users'  # Имя таблицы в базе данных

    # Колонки таблицы
    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    password = Column(String)
```

Замечания к второй лабораторной работе

Этапы выполнения:

1. Нужна вторая лабораторная работа, хотя бы её базовая часть (как её сделать описано выше, также можно взять мой проект, в коде достаточно коментариев);
2. Начинается самое интересное. Redislite не работает на Windows, поэтому будем ставить Docker на котором и равзернем большой Redis. Ссылка на Docker. Устанавливаем (нужна будет перезагрузка) и переход во вкладку `Containers`, в поиске пишем `Redis` и тыкаем `Run`. В нижнем правом углу тыкаем на `Terminal` и переходим в папку своей лабы через команду `cd`, затем прописываем `docker run -d --name redis-container -p 6379:6379 redis` - контейнер с редисом работает.