Тестовое задание на вакансию стажер-разработчик Python

Соискатель: Ознобихин Михаил Витальевич

Контактный телефон: +7 (930) 275-89-94

Веб-сервис написан с помощью:

- FastAPI
- Postgresql

В файле **requirements.txt** описаны версии всех библиотек, которые были задействованы при разработке проекта.

При разработке использовалась ОС Windows 10

Проект был протестирован на ОС Linux (Kali Linux 5.15.0 2022.1, Debian)

Postgresql. Создание БД и таблицы.

Для того, что бы запустить веб-сервис нужно для начала создать и запустить Базу Данных.

Сначала создайте на своём ПК в Postgresql базу данных *photo_db* или с любым другим названием.

```
CREATE DATABASE photo_db;
```

После создания бд в файле **database.py** измените аргументы, принимаемые функцией create_engine на свои персонализированные.

```
engine = create_engine("postgresql://postgres:Messer0Mih@localhost/photo_db", echo=True ) # Где Messer0Mih - пароль от личной БД, а photo_db - название БД
```

Теперь, когда наша база данных создана, нам нужно создать таблицу index.

Также не стоит забывать о том, что для работы с бд, postgresql должна быть запущена. В ОС Debian запуск производится командой:

```
sudo /etc/init.d/postgresql start
```

Для проверки работоспособности кода можем импортировать объект, к примеру, пропишем в терминале:

```
python
>>> from model import Photo
>>> new_photo=(req_code = 1, name_ph = "123", date_time = 11112021)
>>> new_photo
```

В итоге вы должны получить ссылку на объект

В файле model.py содержится класс со структурой создаваемой БД

Теперь наконец-то создадим нашу таблицу, для этого запустим **create_db.py**

```
python create_db.py
```

Если таблица создана успешно, нам покажется структура нашей бд

Наша бд и таблица index созданы!

FastAPI. Методы. Аутентификация.

После создания БД и таблицы можем приступать к работе с самим сервисом.

Методы:

```
POST "/token"
PUT "/frame/"
GET "/frame/<код запроса>"
DELETE "/frame/<код запроса>"
```

Метод "/frame/": Подаётся от 1 до 15 изображений в формате .jpeg. Функция сохраняет данные в папку /data//.jpg и фиксирует в бд в таблице index.

Метод "/frame/<код запроса>": Подаётся req_code. Возвращается список соответствующих изображений в формате JSON, включая дату и время регистрации и имена файлов.

Метод "/frame/<код запроса>": Подаётся req_code. Функция удаляет данные по запросы из бд и из папки /data/ /

Метод "/token" нужен для того, чтобы проверить авторизацию пользователя

Запуск

Запустим наш проект через отладчик(файл **main.py**) или используя команду:

`uvicorn main:app --reload`

Если всё работает корректно, то мы можем перейти по ссылке, которую нам выведет терминал. Для того, что бы работать с функционалом программы, добавим в ссылку "/docs". ссылка.

Методы PUT и DELETE требуют авторизации, в то время как методом GET можно пользоваться не авторизовавшись. Аутентификация выполнена с помощью метода OAuth2.

Есть 2 пользователя:

```
username: greenatom, password: pass - активный пользователь
username: mihail, password: pass2 - не активный пользователь
```

Тесты

Тесты написаны в файле **tests.py**.

Функции тестов:

```
test_put_main_1ph() - Тест загрузки от 1 до 15 фотографий
test_put_main_0ph() - Тест, когда в метод ничего не подается
test_put_main_manyph() - Тест, когда подаётся больше 15 фотографий
test_put_main_1errph() - Тест, когда в метод подается файл не нужного нам формата
test_get_main() - Тест функции вывода
test_delete_main() - Тест удаления несуществующего кода запроса
test_delete_main5() - Тест удаления существующего кода запроса
```

Примечание: Тесты методов PUT и DELETE принимают токен активного юзера - greenatom

---

Вложения

В директории *files_for_test* находятся файлы для проведения тестов