Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up»

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ**

**на итоговый проект «Разработка web-сайта (backend) "Клиенты компании"»**

**по ДПП ПП «Backend разработка на Python»**

|  |  |  |  |
| --- | --- | --- | --- |
| п/п  № | Задание | Исполнитель | Рабочий график (план) выполнения |
| 1 | Выполнить анализ предметной области: выделить основные сущности для моделирования, виды и типы запросов к серверу  Подготовить отчётные документы. | Харчев А.С. | До 12.05.2024   до 30.05.2024 |
| 2 | Разработать модели для построения реляционных таблиц в СУБД PostgreSQL. Определить и реализовать необходимую систему ограничений для обеспечения целостности моделей (Pydantic-модели).  Установить и настроить платформу для администрирования и настройки PostgreSQL, выполнить необходимые построения для тестирования запросов к этой БД.  Установить и настроить автоматическое генерирование и выполнение миграции для БД (Alembic) | Воронкин К.С. | до 20.05.2024 |
| 3 | Подготовить и настроить шаблон проекта: установить менеджер зависимостей Poetry, создать и настроить свой репозиторий в системе контроля версий Git, настроить установить необходимые библиотеки для разработки backend-части приложения на FastAPI | Мороз Д.П. | до 20.05.2024 |
| 4 | Реализация операций CRUD для API-приложения (создание, чтение, обновление и удаление данных, хранящихся в БД), используя систему роутеров, асинхронный (или синхронный) способ подключения к БД и SQLAlchemy для синхронизации объектов Python и записей БД. | Плотников Д.А. | до 20.05.2024 |
|  | Протестировать работу API-приложения через Swagger UI. Создать необходимые настроечные компоненты для развёртывания приложения в рабочей среде: на гостевом хостинге. Выполнить deploy созданного приложения. | Пенкин А.А. | до 21.05.2024 |

Руководитель проекта   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / «\_\_\_ » \_\_\_\_\_\_\_\_\_ 2024 г.

(подпись)

СОГЛАСОВАНО:

Руководитель ДПП ПП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2024 г.

(подпись)

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный университет»

Школа развития цифровых компетенций «Digital Up»

Отчет о выполнении группового итогового проекта по ДПП ПП

«Backend разработка на Python»

**Разработка web-сайта (backend) "Клиенты компании"**

Исполнители:

Харчев А.С.

Воронкин К.С.

Мороз Д.П.

Плотников Д.А.

Пенкин А.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2024 г.

Руководитель проекта

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_ 2024 г.

г. Барнаул, 2024

## Цель проекта

Разработать, протестировать и выполнить развёртывание на хостинге web-сайта «Клиенты компании».

## Задачи проекта и исполнители

Выполнить анализ предметной области: выделить основные сущности для моделирования, виды и типы запросов к серверу

Подготовить отчётные документы. Харчев А.С.

Разработать модели для построения реляционных таблиц в СУБД PostgreSQL. Определить и реализовать необходимую систему ограничений для обеспечения целостности моделей (Pedantic-модели).

Установить и настроить платформу для администрирования и настройки PostgreSQL, выполнить необходимые построения для тестирования запросов к этой БД.

Установить и настроить автоматическое генерирование и выполнение миграции для БД (Alembic)

Воронкин К.С.

Подготовить и настроить шаблон проекта: установить менеджер зависимостей Poetry, создать и настроить свой репозиторий в системе контроля версий Git, настроить установить необходимые библиотеки для разработки backend-части приложения на FastAPI.

Мороз Д.П.

Реализация операций CRUD для API-приложения (создание, чтение, обновление и удаление данных, хранящихся в БД), используя систему роутеров, асинхронный (или синхронный) способ подключения к БД и SQLAlchemy для синхронизации объектов Python и записей БД.

Плотников Д.А.

Протестировать работу API-приложения через Swagger UI. Создать необходимые настроечные компоненты для развёртывания приложения в рабочей среде: на гостевом хостинге. Выполнить deploy созданного приложения.

Пенкин А.А.

## Актуальность и востребованность

На сегодняшний день практически любая компания, которая работает на рынке имеет свой web-сайт, который интегрируем в единую корпоративную систему данной организации. В эту систему входят подсистемы разного уровня. Корпоративная информационная система предназначена для создания единого информационного пространства организации и обеспечения эффективного управления всеми ее ресурсами, включая производственные процессы, ведение продаж и учет заказов. Такой подход обеспечивают полноценный контроль для операционной и административной деятельностью предприятий, создавая единую цепочку, в которую входят: учет финансовой деятельности, планирование, управление материальными потоками, производственные процессы, сбыт продуктов, сотрудничество с поставщиками сырья и партнерскими организациями.

В этом разнообразии подсистем web-сайт компании выступает одним из необходимых элементов корпоративной системы. Это необходимый инструмент для ведения успешного бизнеса. Соответственно, побудительными мотивами для [разработки сайта](https://ratingruneta.ru/web+seo/) являются либо его отсутствие, либо неспособность существующего сайта решать актуальные для компании задачи.

Таким образом, выбранная проблематика проекта обладает актуальностью, а разработанный web-сайт имеет практическую значимость.

## Общие сведения о проделанной работе

В рамках реализации проекта были выполнены следующие шаги

### Подготовка окружения

Создано виртуальное окружение для нашего проекта, в которое установим необходимые зависимости.

### Установка библиотек

Были установлены следующие библиотеки:

fastapi uvicorn pydantic aiosqlite sqlalchemy  
  
**Создание приложения**

Создано простейшее приложение на FastAPI с одним эндпоинтом. Запущен web-сервер Uvicorn. Через браузер проверили, что сервер работает

### Работа с базой данных

Установили и настроили платформу для администрирования и настройки PostgreSQL, выполнили необходимые построения для тестирования запросов к этой БД. Установили и настроили автоматическое генерирование и выполнение миграции для БД (Alembic)

Создали асинхронное подключение, которое будет отвечать за отправку запросов в базу данных. Обратите внимание, что мы говорим SQLAlchemy, что будем использовать драйвер для асинхронного кода aiosqlite. После создания engine, с которым уже можно работать, мы дополнительно создаем фабрику сессий new\_session. Сессия позволяет работать не с обычными списками и словарями, а с моделями данных, которые создаются через классы.

Далее создаем модели. Пример модели *Клиент:*

Base = declarative\_base()

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, Identity(start=10), primary\_key=True)

name = Column(String, index=True, nullable=False)

hashed\_password = Column(String)

Модель соответствует одной таблице в базе данных. Название таблицы мы задаем в атрибуте \_\_tablename\_\_. В базах данных каждая таблица обычно имеет столбец с уникальными значениями с названием id. SQLAlchemy, как и Pydantic, использует аннотации типов для задания категорий столбцов.

### Запросы к базе данных

Для создания запросов к базе данных использовалась SQLAlchemy Object Relational Mapper. В дальнейшем вся работа с БД будет выполнятся через миграции.

### Описание роутеров

Роутер (сущность FastAPI) позволяет организовывать гибкую работу с *эндпоинтами*.

Пример реализации роутера для выводы всех клиентов:

@users\_router.get("/", response\_model=Union[list[Main\_User], New\_Respons], tags=[Tags.users])

def get\_user\_db(DB: Session = Depends(get\_session)):

'''

получаем все записи таблицы

'''

users = DB.query(User).all()

# если не найден, отправляем статусный код и сообщение об ошибке

if users == None:

return JSONResponse(status\_code=404, content={"message": "Пользователи не найдены"})

return users

@users\_router.post("/", response\_model=Union[Main\_User, New\_Respons], tags=[Tags.users], status\_code=status.HTTP\_201\_CREATED)

### Тестирование API-приложения через Swagger UI

Проверим работу роутеров для добавления, удаления, просмотра и изменения записей в таблицах. Пример просмотра записей в таблице Клиенты:

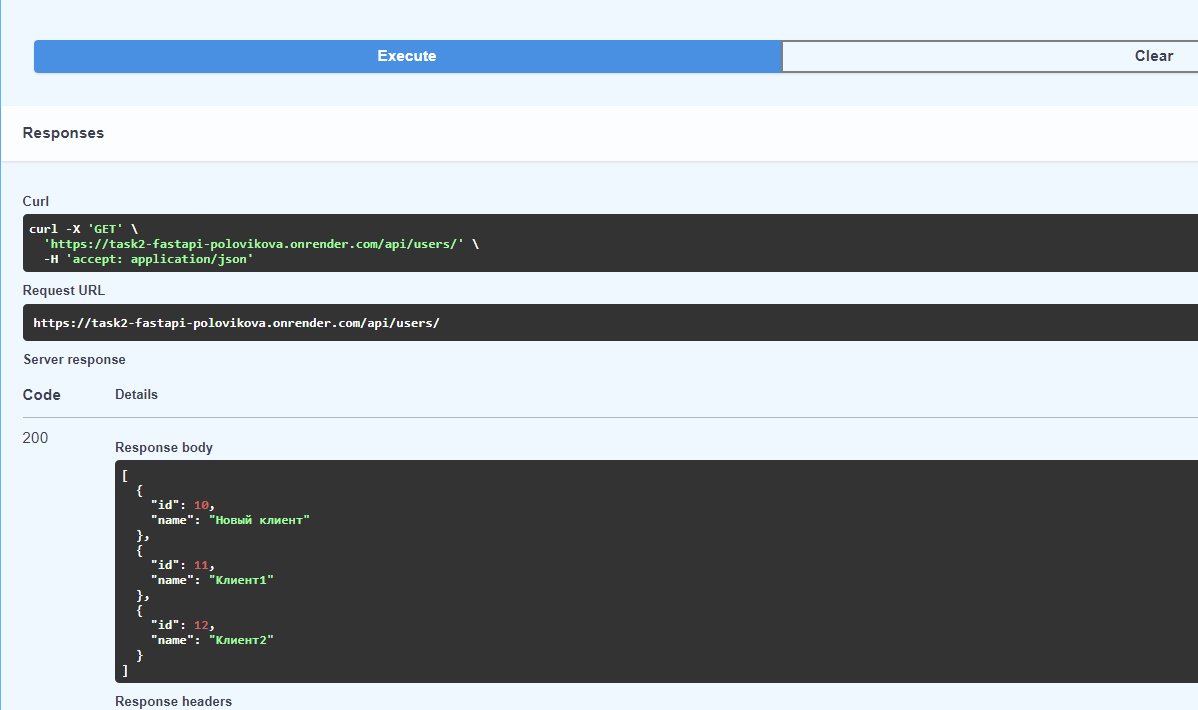


Рисунок 1. – Тестирование API-приложения

### Настройка миграций

Пример блока кода с настройками переменных среды окружения для работы с БД файла env.py (библиотека alembic)

config.set\_section\_option(section, 'POSTGRES\_USER', settings.POSTGRES\_USER)

config.set\_section\_option(section, 'POSTGRES\_PORT', settings.POSTGRES\_PORT)

config.set\_section\_option(section, 'POSTGRES\_PASSWORD', settings.POSTGRES\_PASSWORD)

config.set\_section\_option(section, 'POSTGRES\_DB', settings.POSTGRES\_DB)

config.set\_section\_option(section, 'POSTGRES\_HOST', settings.POSTGRES\_HOST)

### Подготовка к развёртыванию и развёртывание на хостинге

Создание файл со всеми зависимостями, которые используются в проекте. Создание файла Dockerfile. Обновляем репозиторий. Выполняем deploy проекта см. Рисунок 2.

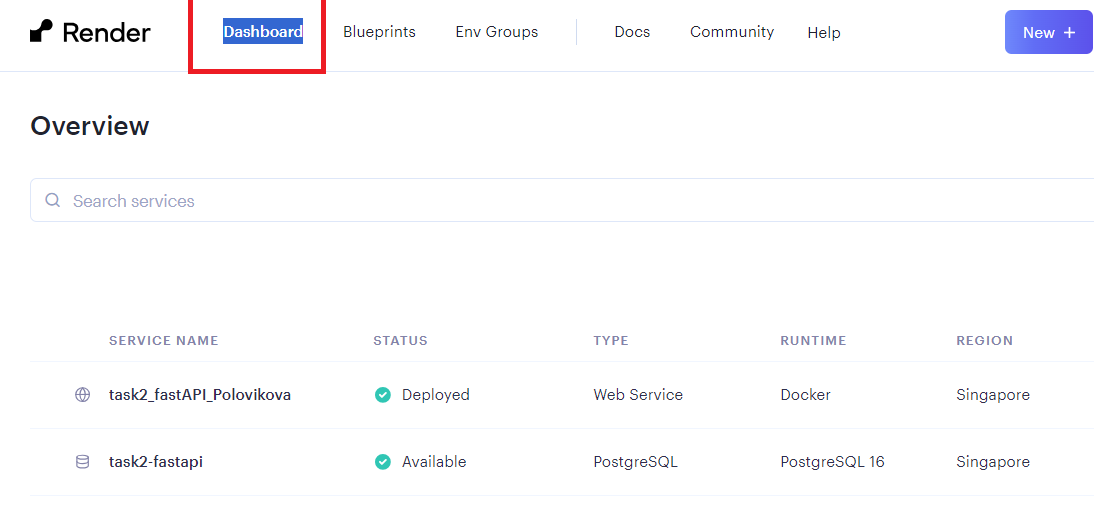


Рисунок 2. – Развёртывание проекта на хостинге

## Результаты проекта

В результате проделанной работы получили готовый к использованию backend web-сайта "Клиенты компании". Пример реализованных endpoint-ов в Swagger UI представлены на странице браузера Рисунок 3.

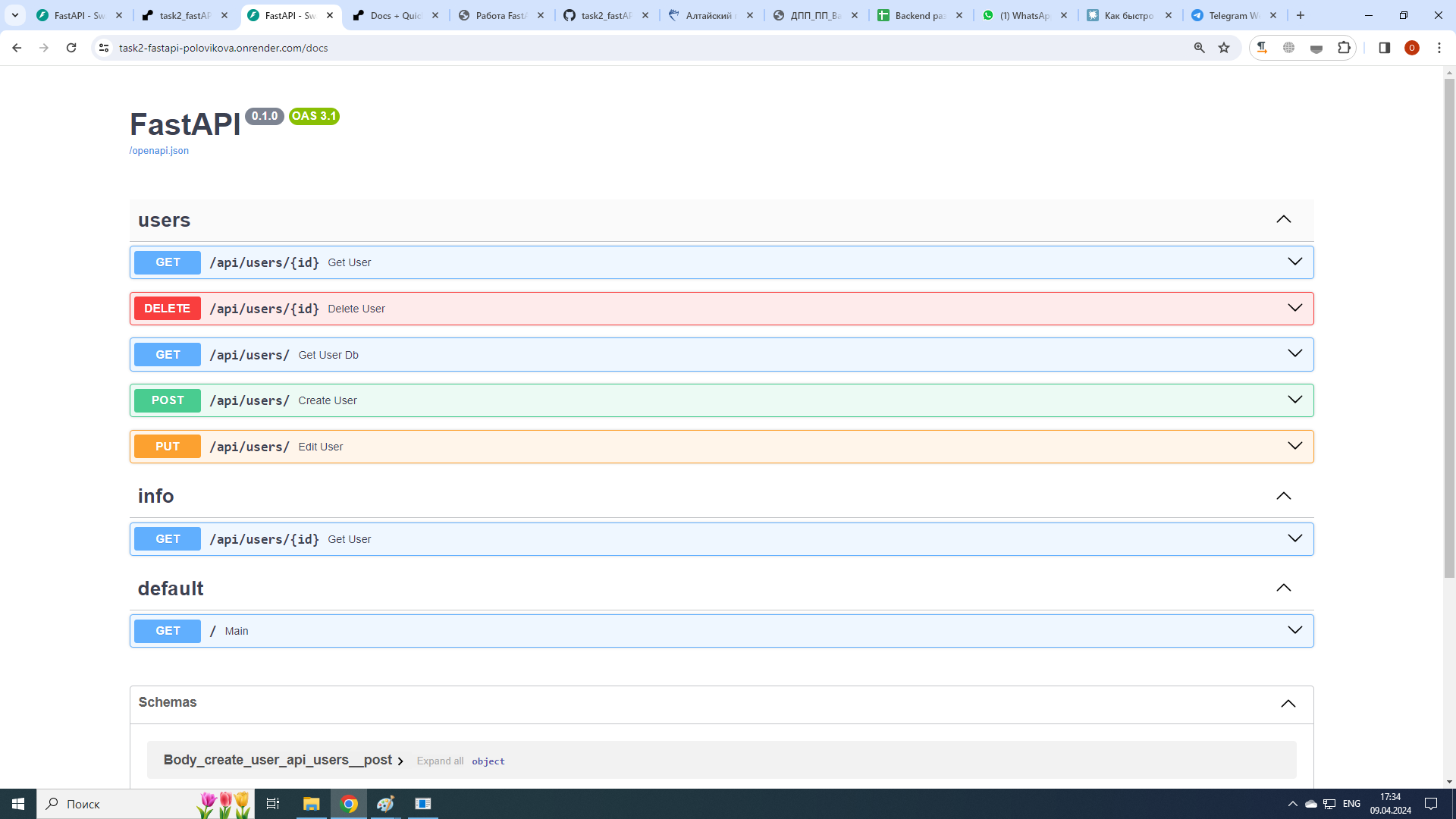


Рисунок 3. – Демонстрация работы проекта на бесплатном сервере Render.com

Работа с таблицами БД PostgreSQL через модели данных SQLAlchemy. Реализована CRUD-система, обеспечивающая выполнение основных запросов на сервер со стороны клиента: Вывод, Обновление (Изменение), Удаление и Создание объектов основных сущностей для данного приложения. На Рисунке 4 представлен код методов: Вывод всех записей Пользователей, Вывод данных конкретного Пользователя по id.

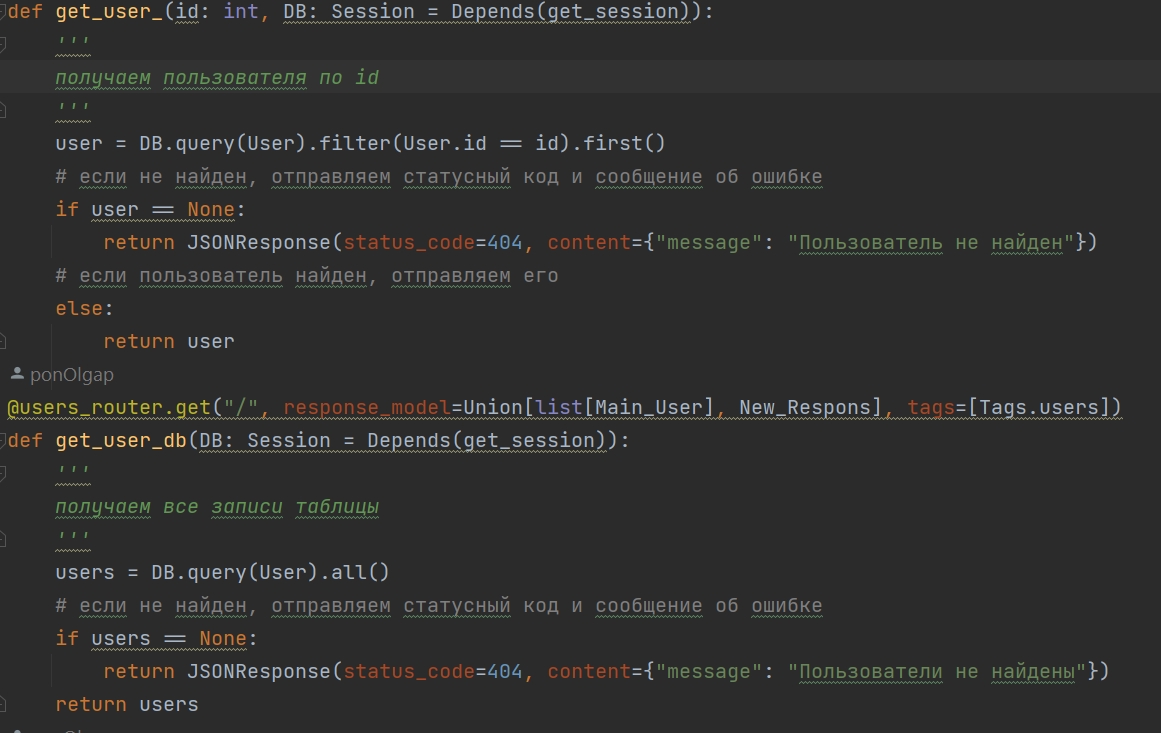


Рисунок 4. –Реализация методов вывода информации по объектам

Вся структура созданного приложения представлено на Рисунке 5.

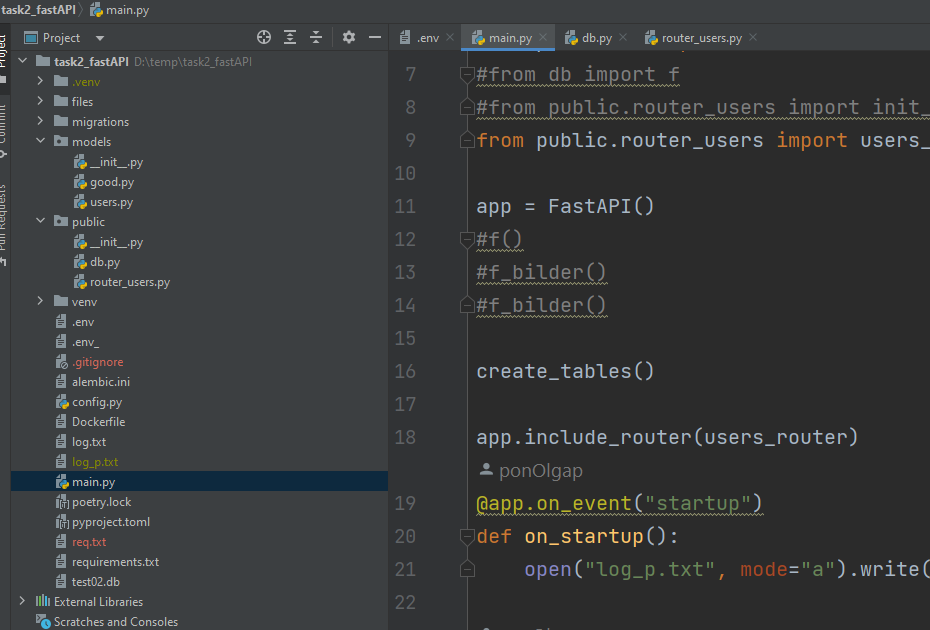


Рисунок 5. –Структура API-проекта

Таким образом, поставленная задача была выполнена, цель достигнута. Web-приложение размещено на бесплатном хостинге и готово к тестировани.

## Приложение 1

(скриншоты, демонстрирующие работу созданного программного продукта)

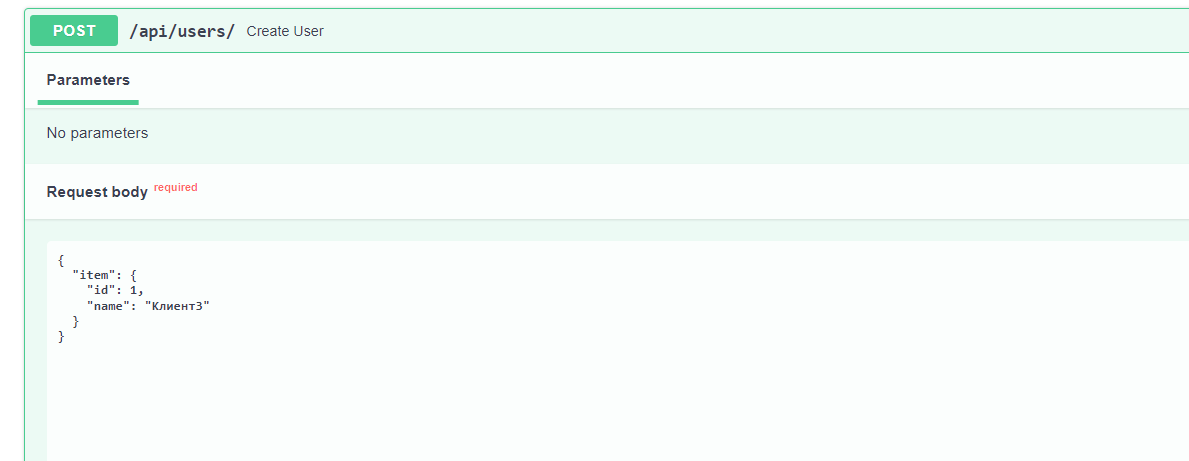


Рисунок 6. – Демонстрация запроса на добавления нового клиента в БД

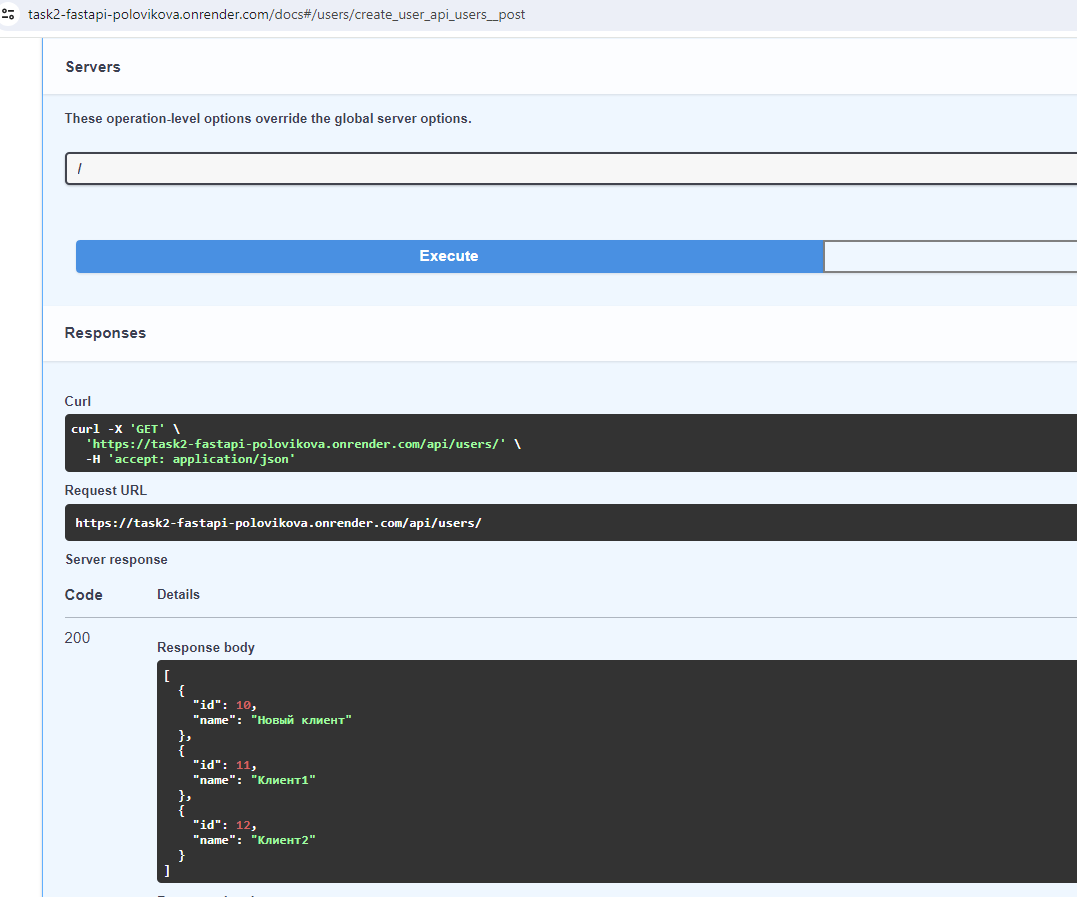


Рисунок 7. – Демонстрация запроса на «получение» клиентов из БД

## Приложение 2

(блок программного кода)

**Код программного файла main.py**

import uvicorn

from fastapi import FastAPI

from fastapi.responses import FileResponse

from datetime import datetime

from public.db import create\_tables

#from db import f

#from public.router\_users import init\_db

from public.router\_users import users\_router

app = FastAPI()

#f()

#f\_bilder()

#f\_bilder()

create\_tables()

app.include\_router(users\_router)

@app.on\_event("startup")

def on\_startup():

open("log\_p.txt", mode="a").write(f'{datetime.utcnow()}: Begin\n')

@app.on\_event("shutdown")

def shutdown():

open("log\_p.txt", mode="a").write(f'{datetime.utcnow()}: End\n')

@app.get("/")

def main():

return FileResponse("files/index.html")

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run(app, host="127.0.0.1", port=8000)

**Код скрипта, описывающего логику работы роутера**

from fastapi import APIRouter, Depends, Body, HTTPException

from fastapi.responses import JSONResponse

from starlette import status

from models.good import \*

from sqlalchemy.orm import sessionmaker, Session

from public.db import engine\_s

def get\_session():

with Session(engine\_s) as session:

try:

yield session

finally:

session.close()

#реализация маршрутов для операций c конкретными тегами - конкретизация роутера

users\_router = APIRouter(tags=[Tags.users], prefix='/api/users')

info\_router = APIRouter(tags=[Tags.info])

def coder\_passwd(cod: str):

return cod\*2

@users\_router.get("/{id}", response\_model=Union[New\_Respons, Main\_User], tags=[Tags.info])

# далее идут опуерации пути для CRUD

def get\_user\_(id: int, DB: Session = Depends(get\_session)):

'''

получаем пользователя по id

'''

user = DB.query(User).filter(User.id == id).first()

# если не найден, отправляем статусный код и сообщение об ошибке

if user == None:

return JSONResponse(status\_code=404, content={"message": "Пользователь не найден"})

# если пользователь найден, отправляем его

else:

return user

@users\_router.get("/", response\_model=Union[list[Main\_User], New\_Respons], tags=[Tags.users])

def get\_user\_db(DB: Session = Depends(get\_session)):

'''

получаем все записи таблицы

'''

users = DB.query(User).all()

# если не найден, отправляем статусный код и сообщение об ошибке

if users == None:

return JSONResponse(status\_code=404, content={"message": "Пользователи не найдены"})

return users

@users\_router.post("/", response\_model=Union[Main\_User, New\_Respons], tags=[Tags.users], status\_code=status.HTTP\_201\_CREATED)

def create\_user(item: Annotated[Main\_User, Body(embed=True, description="Новый пользователь")],

DB: Session = Depends(get\_session)):

try:

user = User(name=item.name, hashed\_password=coder\_passwd(item.name))

if user is None:

raise HTTPException(status\_code=404, detail="Объект не определен")

DB.add(user)

DB.commit()

DB.refresh(user)

return user

except Exception as e:

raise HTTPException(status\_code=500, detail=f"Произошла ошибка при добавлении объекта {user}")

@users\_router.put("/", response\_model=Union[Main\_User, New\_Respons], tags=[Tags.users])

def edit\_user\_(item: Annotated[Main\_User,Body(embed=True, description="Изменяем данные для пользователя по id")],

DB: Session = Depends(get\_session)):

# получаем пользователя по id

user = DB.query(User).filter(User.id == item.id).first()

# если не найден, отправляем статусный код и сообщение об ошибке

if user == None:

return JSONResponse(status\_code=404, content={"message": "Пользователь не найден"})

# если пользователь найден, изменяем его данные и отправляем обратно клиенту

user.name = item.name

try:

DB.commit()

DB.refresh(user) # сохраняем изменения

except HTTPException:

return JSONResponse(status\_code=404, content={"message": ""})

return user

@users\_router.delete("/{id}", response\_class=JSONResponse, tags=[Tags.users])

def delete\_user(id: int, DB: Session = Depends(get\_session)):

# получаем пользователя по id

user = DB.query(User).filter(User.id == id).first()

# если не найден, отправляем статусный код и сообщение об ошибке

if user == None:

return JSONResponse(status\_code=404, content={"message": "Пользователь не найден"})

try:

DB.delete(user)

DB.commit() # сохраняем изменения

except HTTPException:

JSONResponse(content={'message' : f'Ошибка'})

return JSONResponse(content={'message' : f'Пользователь удалён {id}'})

## Приложение 3

(ссылка на проект, ссылка на репозиторий с кодом)

<https://github.com/ponOlgap/task2_fastAPI_Polovikova>

[https://task2-fastapi-polovikova.onrender.com](https://task2-fastapi-polovikova.onrender.com/)