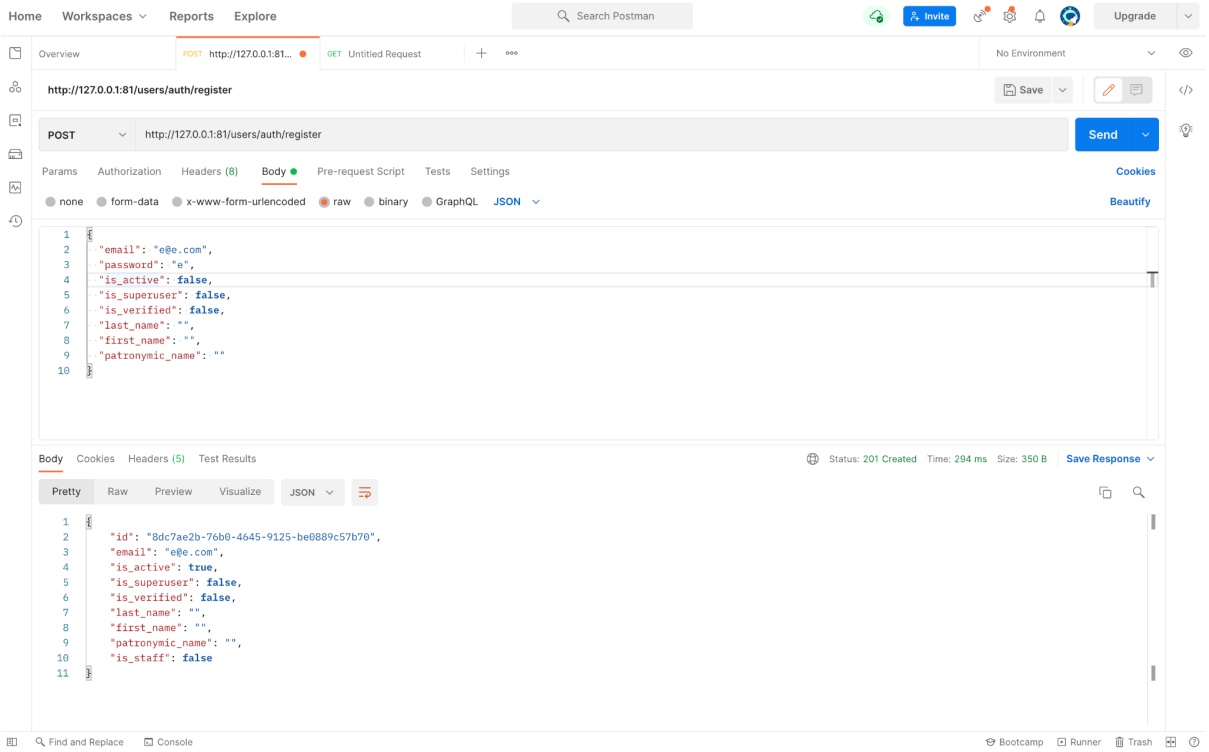
Лабораторная работа №3
Технологии web-программирования.
«Серверное программирование»

Цель: познакомиться с основами backend разработки web-приложений. Научится писать скрипты. Познакомиться с основами работы docker. Познакомиться с фреймворком fastapi и научиться разворачивать проект, производить его настройку. Научится работать с API в приложении Postman.

Задание к лабораторной работе:

1. Развернуть базовое приложение.
2. Настроить конфигурацию работы приложения с docker.
3. Добавить модуль для работы с API.
4. Добавить несколько контроллеров со статическими данными.
5. Продемонстрировать работу API в Postman.

Выполнение запроса регистрации в postman



Запущенные docker контейнеры

docker-compose.yml

```
version: '2.2'

services:
  backend:
    build:
      context: backend/
      dockerfile: Dockerfile
    container_name: app_backend
    command: uvicorn main:app --host 0.0.0.0 --port 8000
    env_file:
      - ./.test.env
    depends_on:
      - db
    expose:
      - 8000
    volumes:
      - static_volume:/home/app/static
    networks:
      - app_net

  db:
    image: postgres:11
    container_name: app_db
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    env_file:
      - ./.test.env
    ports:
      - 5435:5432
    networks:
      - app_net

  nginx:
    image: nginx:mainline-alpine
    container_name: app_nginx
    ports:
      - 80:80
      - 81:81
    volumes:
      - static_volume:/home/app/static
      - ./nginx:/etc/nginx/conf.d
    depends_on:
```

```
      - backend
      - frontend
    networks:
      - app_net

  frontend:
    build:
      context: frontend/
      dockerfile: Dockerfile
    container_name: app_frontend
    env_file:
      - ./.test.env
    depends_on:
      - backend
    expose:
      - 8080
    volumes:
      - node_modules:/app/node_modules
    networks:
      - app_net

volumes:
  postgres_data:
  static_volume:
  node_modules:

networks:
  app_net:
    driver: bridge
```

nginx.conf

```
upstream backend {
   server backend:8000;
}

 upstream frontend {
    server frontend:8080;
 }

server {
   listen 81;
   server_name 127.0.0.1 localhost;
   client_max_body_size 100m;
   proxy_ignore_client_abort on;
   if_modified_since off;
   add_header Last-Modified "";
   proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
   proxy_set_header Host $host;
   proxy_redirect off;
   uwsgi_read_timeout 300s;

   location / {
      proxy_pass http://backend;
   }
}

 server {
    listen 80;
    server_name 127.0.0.1 localhost;
    client_max_body_size 100m;
    if_modified_since off;
    add_header Last-Modified "";
    proxy_ignore_client_abort on;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $host;
    proxy_redirect off;
    uwsgi_read_timeout 300s;

    location / {
       proxy_pass http://frontend;
    }
 }
```

Dockerfile(frontend)

```
FROM node:lts-alpine

# устанавливаем простой HTTP-сервер для статики
RUN npm install -g http-server

# делаем каталог 'app' текущим рабочим каталогом
WORKDIR /app

# копируем оба 'package.json' и 'package-lock.json' (если есть)
COPY package*.json ./

# устанавливаем зависимости проекта
RUN npm install

# копируем файлы и каталоги проекта в текущий рабочий каталог (т.е. в каталог 'app')
COPY . .

# собираем приложение для production с минификацией
RUN npm run build

EXPOSE 8080
CMD [ "http-server", "dist" ]
```

Dockerfile(backend)

```
FROM python:3.8
WORKDIR /home/app

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN pip install --upgrade pip

COPY ./requirements.txt /home/app/requirements.txt
RUN pip install -r requirements.txt

RUN apt-get update && apt-get install netcat -y

COPY . /home/app

RUN ["chmod", "+x", "/home/app/scripts/entrypoint.sh"]
ENTRYPOINT ["/home/app/scripts/entrypoint.sh"]
```