**Азбукин Даниил БСБО-10-21**

**Практическая №4**

App.py:

from flask import Flask  
  
app = Flask(\_\_name\_\_)  
  
  
@app.route('/')  
def hello\_world():  
 return 'Hello, Docker!'  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', port=8081)

Dockerfile:

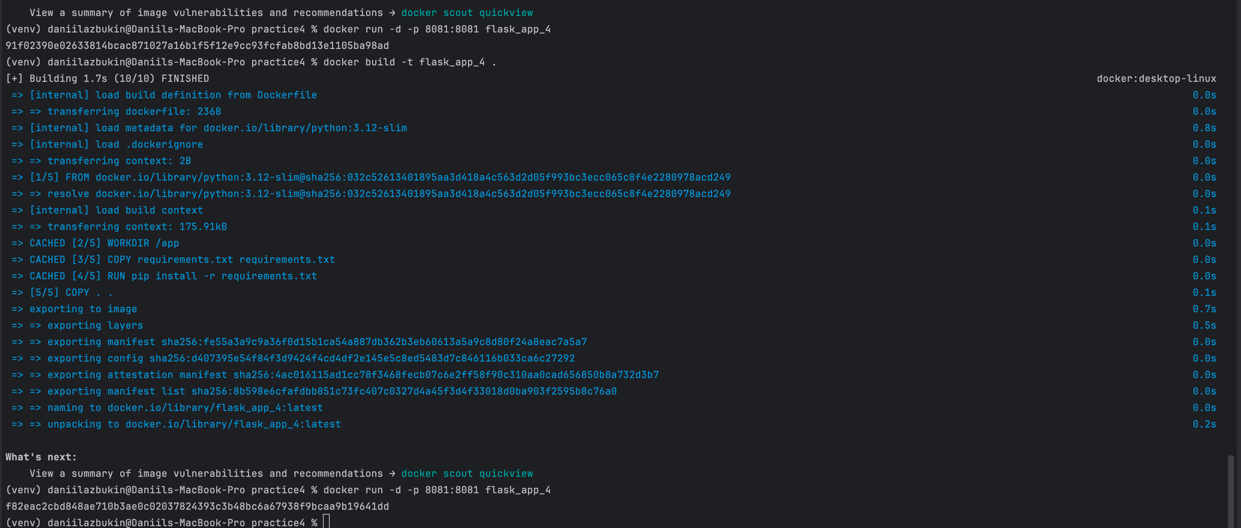
FROM python:3.12-slim  
  
WORKDIR /app  
  
COPY requirements.txt requirements.txt  
RUN pip install -r requirements.txt  
  
COPY . .  
  
EXPOSE 8081  
  
CMD ["python", "app.py"]

Requirements.txt

Flask~=3.0.3

docker build -t flask\_app\_4 . – Создаем docker образ нашего приложения

docker run -p 8081:8081-d flask\_app\_4 – Запускаем контейнер с приложением



Чтобы уменьшить размер образа и контейнера, используем более легкую версию python:

Dockerfile:

FROM python:3.12-alpine  
  
WORKDIR /app  
  
COPY requirements.txt requirements.txt  
RUN pip install -r requirements.txt  
  
COPY . .  
  
EXPOSE 8081  
  
CMD ["python", "app.py"]

Создадим тесты и добавим их запуск в Dockerfile:

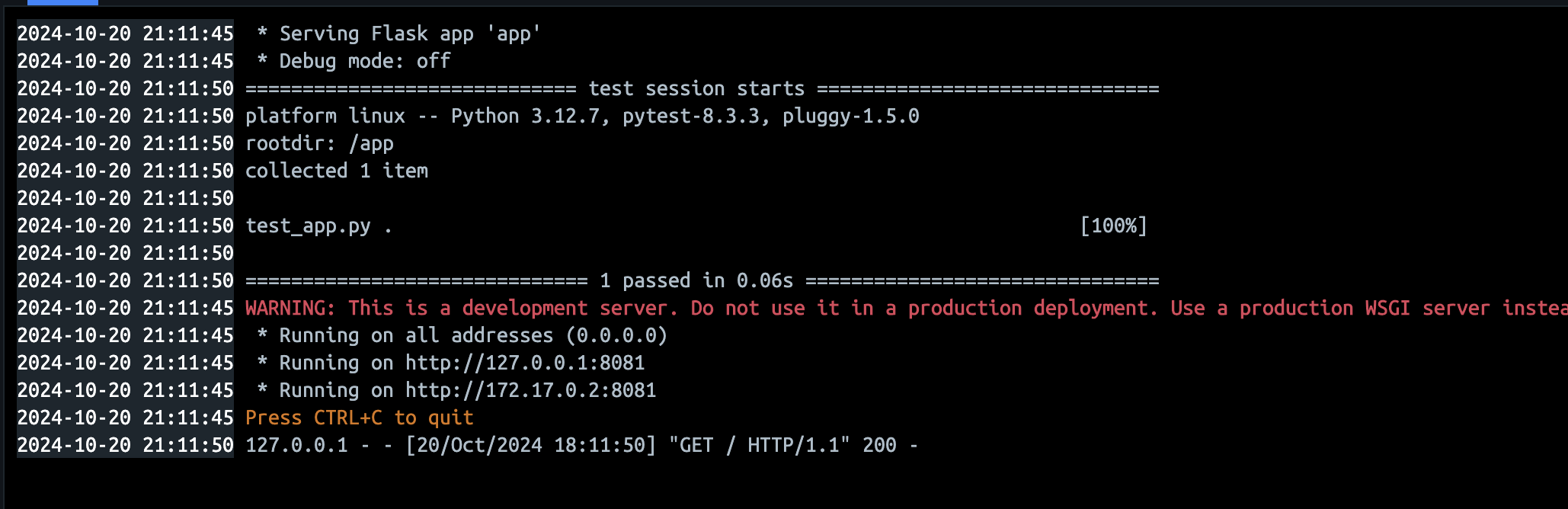
test\_app.py:

from pytest import mark, fixture  
from requests import Session, Response  
  
  
class BaseSession(Session):  
 def \_\_init\_\_(self, \*\*kwargs):  
 self.base\_url = kwargs.pop('base\_url')  
 super().\_\_init\_\_(\*\*kwargs)  
  
 def request(self, method, url, \*\*kwargs) -> Response:  
 response = super().request(method, url=f'{self.base\_url}{url}', \*\*kwargs)  
 return response  
  
  
@fixture  
def session():  
 return BaseSession(base\_url='http://localhost:8081')  
  
  
def test\_check\_status\_code(session):  
 response = session.get('/')  
 assert response.status\_code == 200

Dockerfile:

FROM python:3.12-alpine  
  
WORKDIR /app  
COPY requirements.txt requirements.txt  
RUN pip install -r requirements.txt  
  
COPY . .  
  
EXPOSE 8081  
  
CMD ["sh", "-c", "python app.py & sleep 5 && pytest"]

Перебилдим и запустим проект и посмотрим логи контейнера:



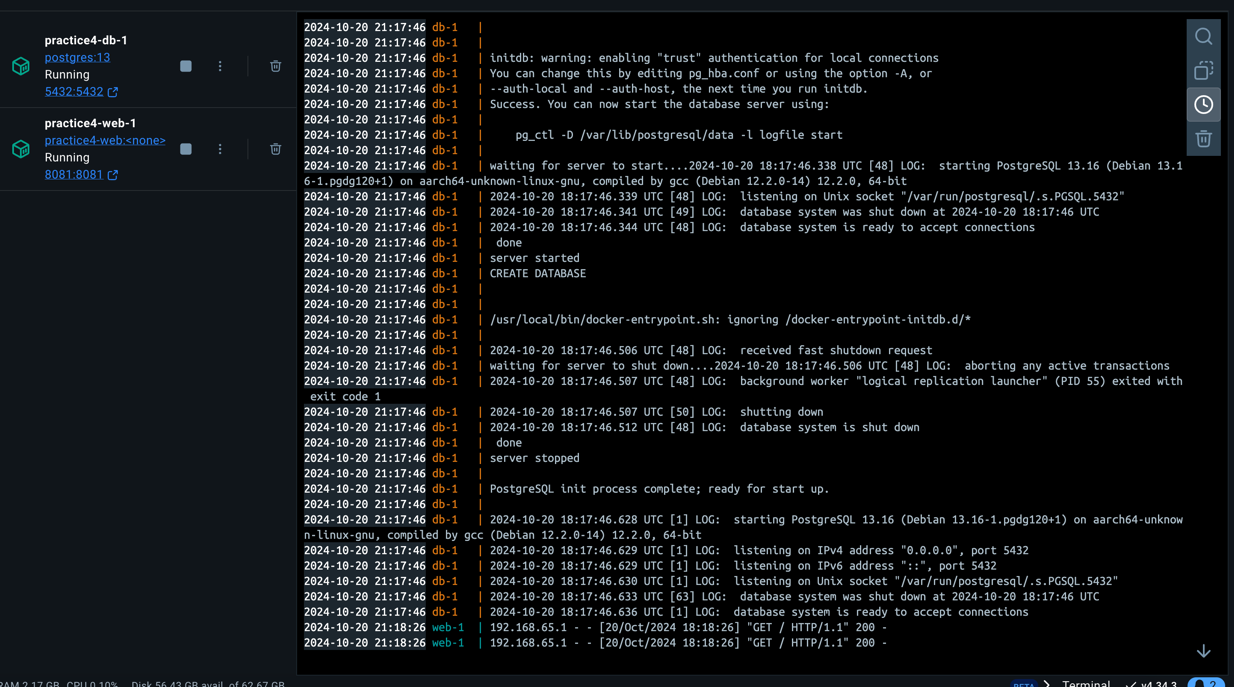
Создадим docker-compose.yaml с сервисами: наше приложение и postgres:

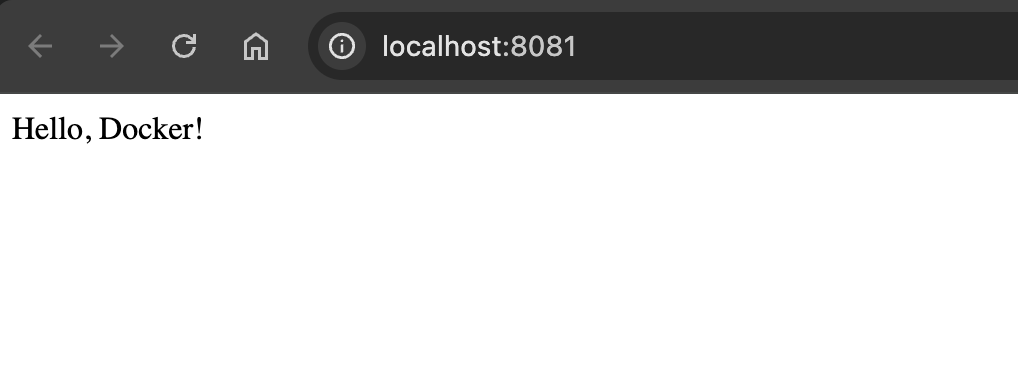
docker-compose.yaml:

version: '3'  
services:  
 web:  
 build: .  
 ports:  
 - "8081:8081"  
 depends\_on:  
 - db  
 db:  
 image: postgres:13  
 environment:  
 POSTGRES\_USER: user  
 POSTGRES\_PASSWORD: password  
 POSTGRES\_DB: flask\_db  
 ports:  
 - "5432:5432"

docker-compose build – Билдим проект

docker-compose up – Поднимаем приложение с базой данных





**Практическая №5**

1. Создание структуры из 2 приложений из Flask и Postgres

app.py – запуск приложения и прокидывание конфига

from config import Config  
from app import app, db  
  
  
with app.app\_context():  
 db.create\_all()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host=Config.FLASK\_APP\_HOST,  
 port=Config.FLASK\_APP\_PORT,  
 debug=Config.FLASK\_DEBUG)

config.py – сам конфиг приложения креды из env

import os  
from dotenv import load\_dotenv  
  
BASE\_DIR = os.path.abspath(os.path.dirname(\_\_file\_\_))  
load\_dotenv(os.path.join(BASE\_DIR, ".env"))  
  
  
class Config:  
  
 FLASK\_APP = "app.py"  
 FLASK\_APP\_HOST = os.getenv("FLASK\_APP\_HOST")  
 FLASK\_APP\_PORT = os.getenv("FLASK\_APP\_PORT")  
 FLASK\_DEBUG = os.getenv("FLASK\_DEBUG")  
  
 SQLALCHEMY\_DATABASE\_URI = os.getenv("SQLALCHEMY\_DATABASE\_URI")  
 SQLALCHEMY\_TRACK\_MODIFICATIONS = os.getenv("SQLALCHEMY\_TRACK\_MODIFICATIONS")  
  
 CACHE\_TYPE = 'redis'  
 CACHE\_REDIS\_HOST = os.getenv("REDIS\_HOST")  
 CACHE\_REDIS\_PORT = os.getenv("REDIS\_PORT")  
 CACHE\_REDIS\_DB = os.getenv("REDIS\_DB")  
 CACHE\_REDIS\_URL = f"redis://{CACHE\_REDIS\_HOST}:{CACHE\_REDIS\_PORT}/{CACHE\_REDIS\_DB}"  
  
  
settings = Config

app.\_\_init\_\_.py – пакет для импорта базы данных и самого потока приложения

from flask import Flask  
from flask\_sqlalchemy import SQLAlchemy  
from config import Config  
from flask\_migrate import Migrate  
from flask\_caching import Cache  
  
app = Flask(\_\_name\_\_)  
app.config.from\_object(Config)  
  
cache = Cache(app)  
  
db = SQLAlchemy(app)  
migrate = Migrate(app, db)  
  
from app import routes, models, constants

app.routes.py – роуты приложения

from flask import request  
from app.models import User  
from app import app, db, cache  
from app.utils import internal\_error\_response, is\_user\_exist, make\_user\_id\_cache\_key  
from app.constants import USER\_ALREADY\_EXISTS, USER\_NOT\_FOUNDED  
  
  
@app.get('/users')  
def get\_users():  
 *"""Get all users from database"""* users = User.query.all()  
 users\_list = [user.to\_dict() for user in users]  
 return users\_list  
  
  
@app.route('/user/<int:user\_id>')  
@cache.cached(timeout=300, make\_cache\_key=make\_user\_id\_cache\_key)  
def get\_user(user\_id):  
 *"""Gets user by id"""* user = User.query.get(user\_id)  
  
 if not user:  
 return USER\_NOT\_FOUNDED, 404  
  
 return user.to\_dict(), 200  
  
  
@app.post('/users')  
def create\_user():  
 *"""Create new user in database"""* data = request.get\_json()  
 username, email = data['username'], data['email']  
  
 if is\_user\_exist(email):  
 return USER\_ALREADY\_EXISTS, 404  
  
 try:  
 new\_user = User(username=username, email=email)  
 db.session.add(new\_user)  
 db.session.commit()  
 except Exception as err:  
 return internal\_error\_response(err)  
  
 return {'message': f'User with id = {new\_user.id} created successfully'}  
  
  
@app.put('/users/<int:user\_id>')  
def update\_user(user\_id):  
 *"""Update user in database"""* user = User.query.get(user\_id)  
 data = request.get\_json()  
  
 if not user:  
 return USER\_NOT\_FOUNDED, 404  
  
 try:  
 user.username = data.get('username', user.username)  
 user.email = data.get('email', user.email)  
 db.session.commit()  
 except Exception as err:  
 return internal\_error\_response(err)  
  
 return {'message': 'User updated successfully'}  
  
  
@app.route('/users/<int:user\_id>', methods=['DELETE'])  
def delete\_user(user\_id):  
 *"""Delete user from database"""* user = User.query.get(user\_id)  
  
 if not user:  
 return USER\_NOT\_FOUNDED, 404  
  
 try:  
 db.session.delete(user)  
 db.session.commit()  
 except Exception as err:  
 return internal\_error\_response(err)  
  
 return {'message': 'User deleted successfully'}  
  
  
@app.get('/clear\_cache/<int:user\_id>')  
def clear\_user\_cache(user\_id):  
 *"""Clear cache for user with id"""* cache.delete(f'user\_data::{user\_id}')  
 return {'message': f'Cache for user {user\_id} cleared'}

app.models.py – модели для базы данных

from app import db  
  
  
class User(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(80), unique=True, nullable=False)  
 email = db.Column(db.String(120), unique=True, nullable=False)  
  
 def \_\_repr\_\_(self):  
 return f'<User {self.username}>'  
  
 def to\_dict(self):  
 return {  
 'id': self.id,  
 'username': self.username,  
 'email': self.email  
 }

app.utils.py – доп функции

from app.constants import ERROR\_500\_TEXT, USER\_ALREADY\_EXISTS  
from app import db  
from app.models import User  
from flask import request  
  
  
def make\_user\_id\_cache\_key(\*args, \*\*kwargs):  
 *"""User id cache key"""* user\_id = kwargs['user\_id']  
 return f'user\_data::{user\_id}'  
  
  
def internal\_error\_response(err):  
 *"""  
 Returns error response  
 :param err: Exception  
 :return: dict  
 """* return {  
 'message': ERROR\_500\_TEXT,  
 'error': str(err)  
 }, 500  
  
  
def is\_user\_exist(email):  
 *"""  
 Check if user with email exist in database  
 :param email:  
 :return:  
 """* user\_with\_email = User.query.filter\_by(email=email).first()  
 return user\_with\_email is not None

.env – envшный конфиг

FLASK\_APP\_HOST=0.0.0.0  
FLASK\_APP\_PORT=8081  
FLASK\_DEBUG=1  
  
SQLALCHEMY\_DATABASE\_URI=postgresql://postgres:secret@db:5432/flask\_app  
SQLALCHEMY\_TRACK\_MODIFICATIONS=0  
  
REDIS\_HOST=  
REDIS\_PORT=  
REDIS\_DB=0

requirements.txt

alembic==1.13.3  
blinker==1.8.2  
cachelib==0.9.0  
click==8.1.7  
Flask==3.0.3  
Flask-Caching==2.3.0  
Flask-Migrate==4.0.7  
Flask-SQLAlchemy==3.1.1  
itsdangerous==2.2.0  
Jinja2==3.1.4  
Mako==1.3.6  
MarkupSafe==3.0.2  
psycopg2-binary==2.9.10  
python-dotenv==1.0.1  
redis==5.2.0  
SQLAlchemy==2.0.36  
typing\_extensions==4.12.2  
Werkzeug==3.0.4

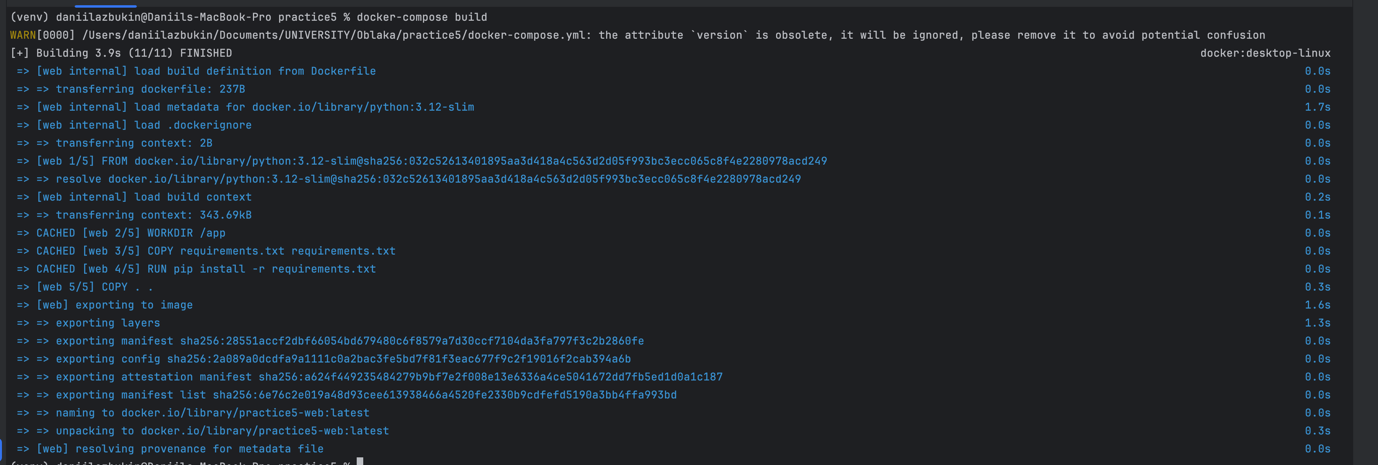
Dockerfile – докерфайл для запуска приложения фласки

FROM python:3.12-slim  
  
WORKDIR /app  
  
COPY requirements.txt requirements.txt  
  
RUN pip install -r requirements.txt  
  
COPY . .  
  
EXPOSE 5000  
  
CMD ["python", "app.py"]

Docker-compose.yaml – композ файл для запуска bd и приложения в контейнерах

version: '3.7'  
services:  
  
 web:  
 build: .  
 ports:  
 - "8081:8081"  
 depends\_on:  
 db:  
 condition: service\_healthy  
 redis:  
 condition: service\_healthy  
 environment:  
 - SQLALCHEMY\_DATABASE\_URI=postgresql://postgres:secret@db:5432/flask\_app  
 - REDIS\_HOST=redis  
  
 db:  
 image: postgres:13  
 environment:  
 POSTGRES\_USER: postgres  
 POSTGRES\_PASSWORD: secret  
 POSTGRES\_DB: flask\_app  
 ports:  
 - "5432:5432"  
 healthcheck:  
 test: ["CMD-SHELL", "pg\_isready -U postgres"]  
 interval: 10s  
 timeout: 5s  
 retries: 5  
  
 redis:  
 image: redis:alpine  
 ports:  
 - "6379:6379"  
 healthcheck:  
 test: ["CMD", "redis-cli", "ping"]  
 interval: 10s  
 timeout: 5s  
 retries: 5

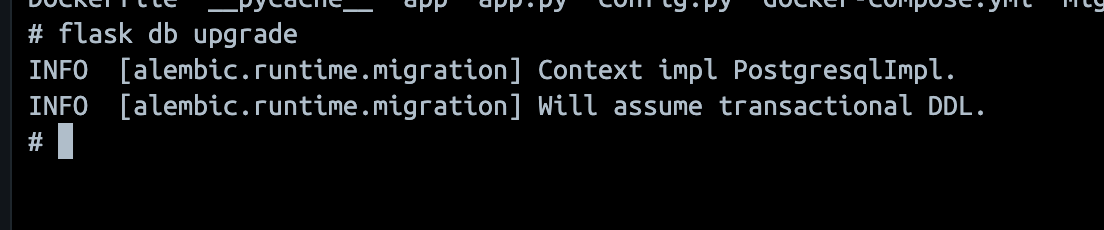
Билд приложения:



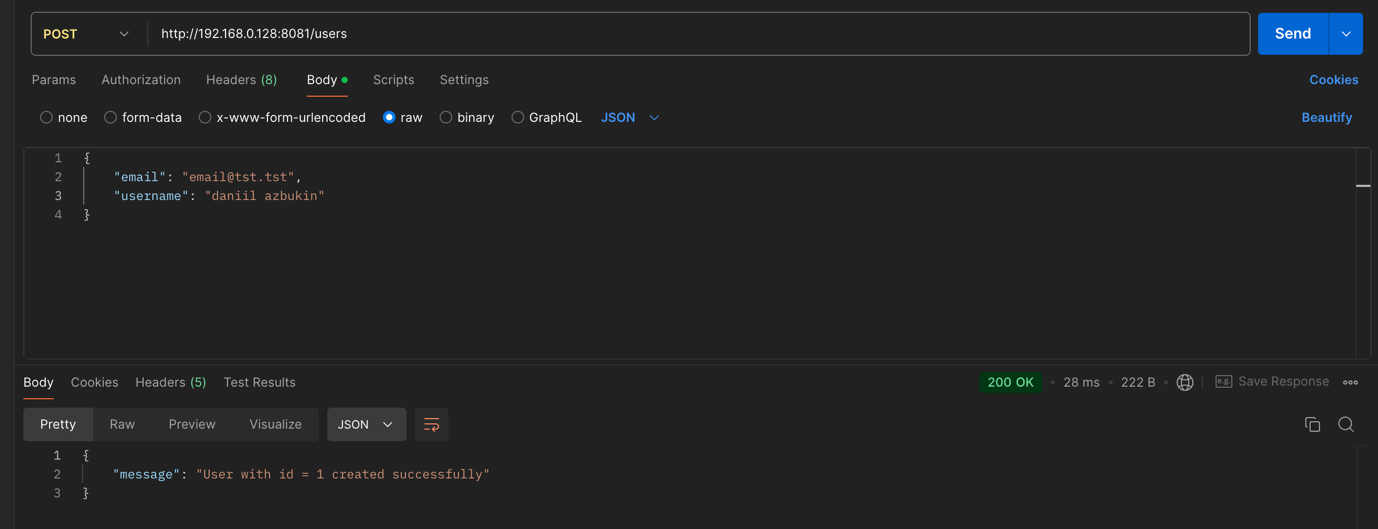
Запуск в композе:



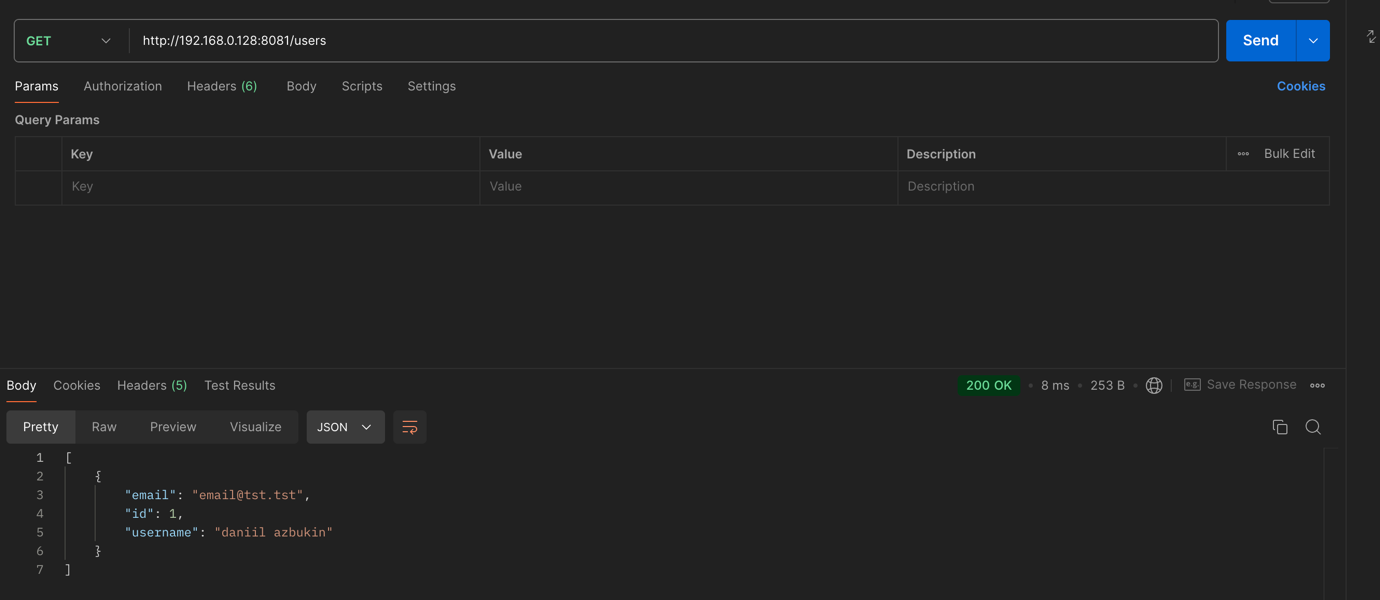
Миграции:



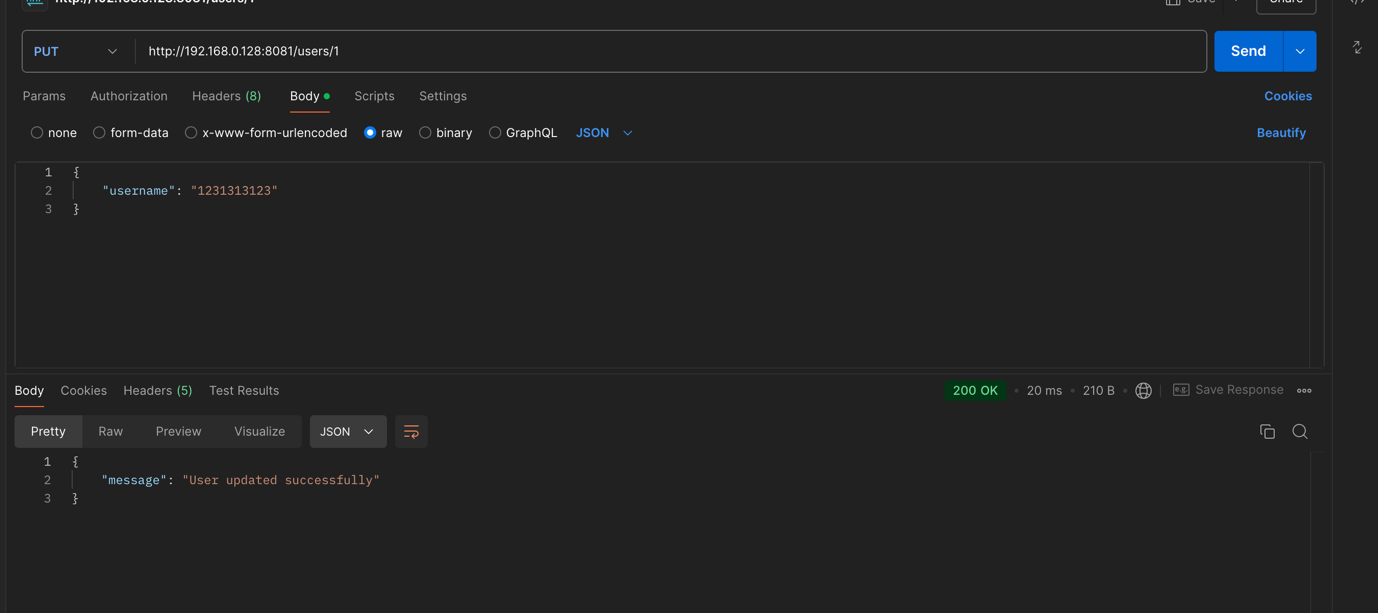
Создание пользователя:



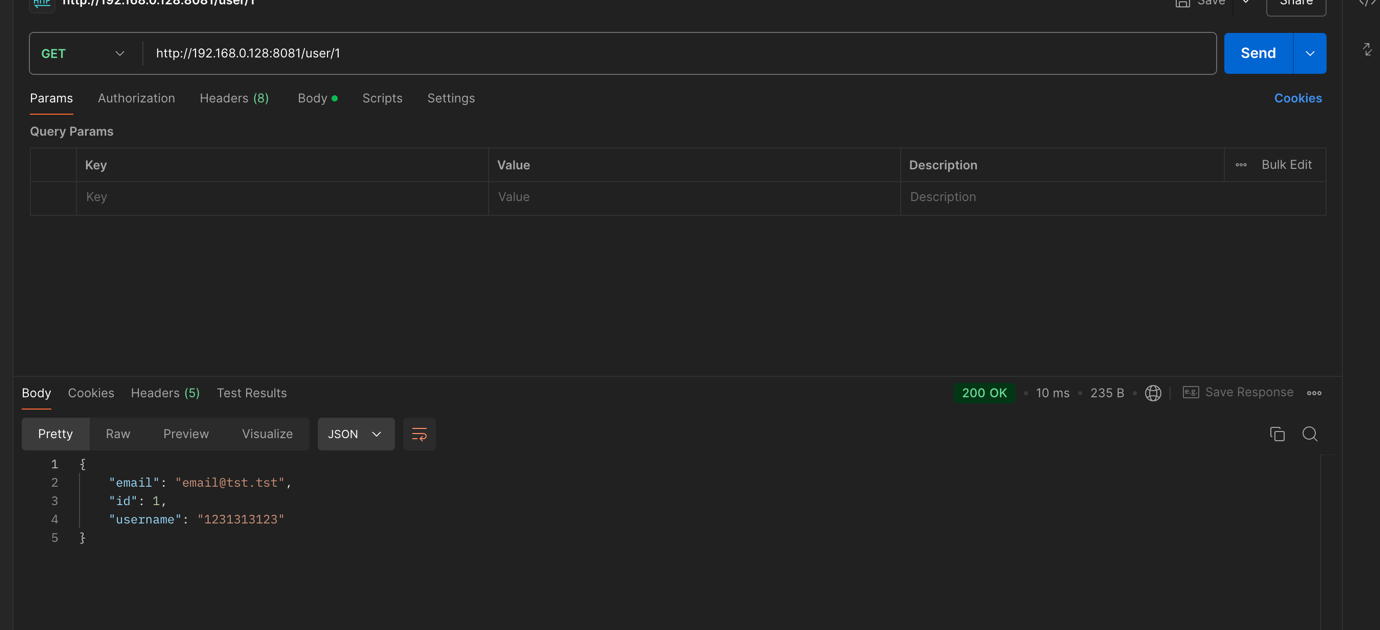
Получение пользователей:



Изменение пользователей:

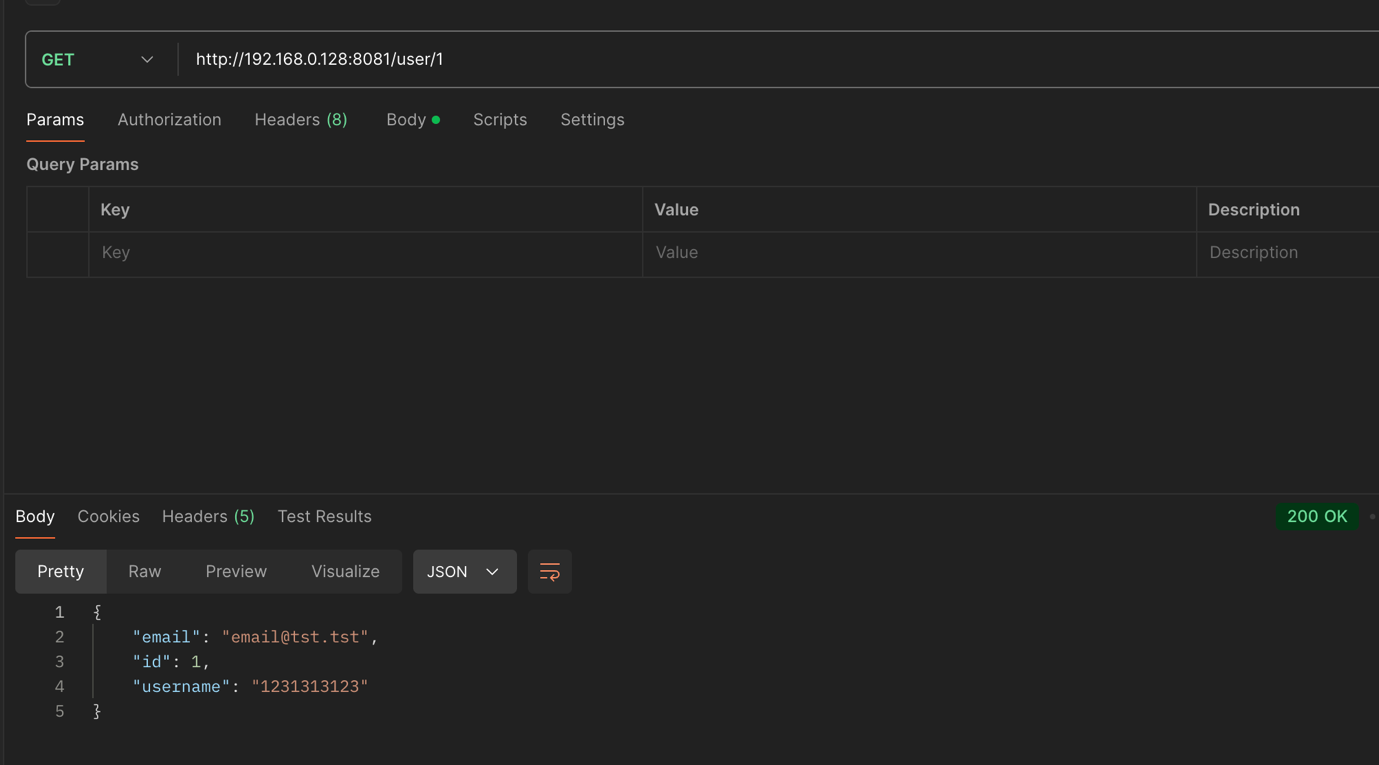


Получение пользователя по ID:

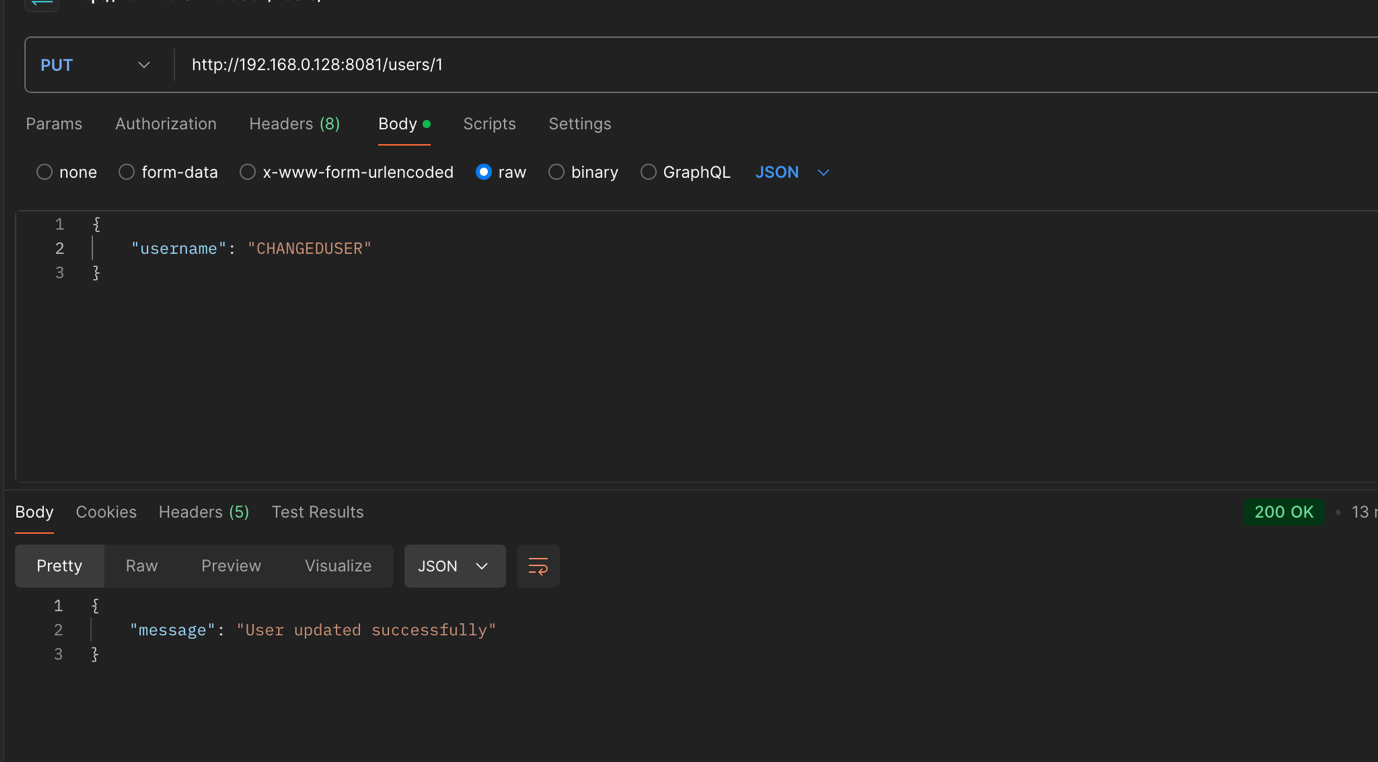


Очистка кеша:

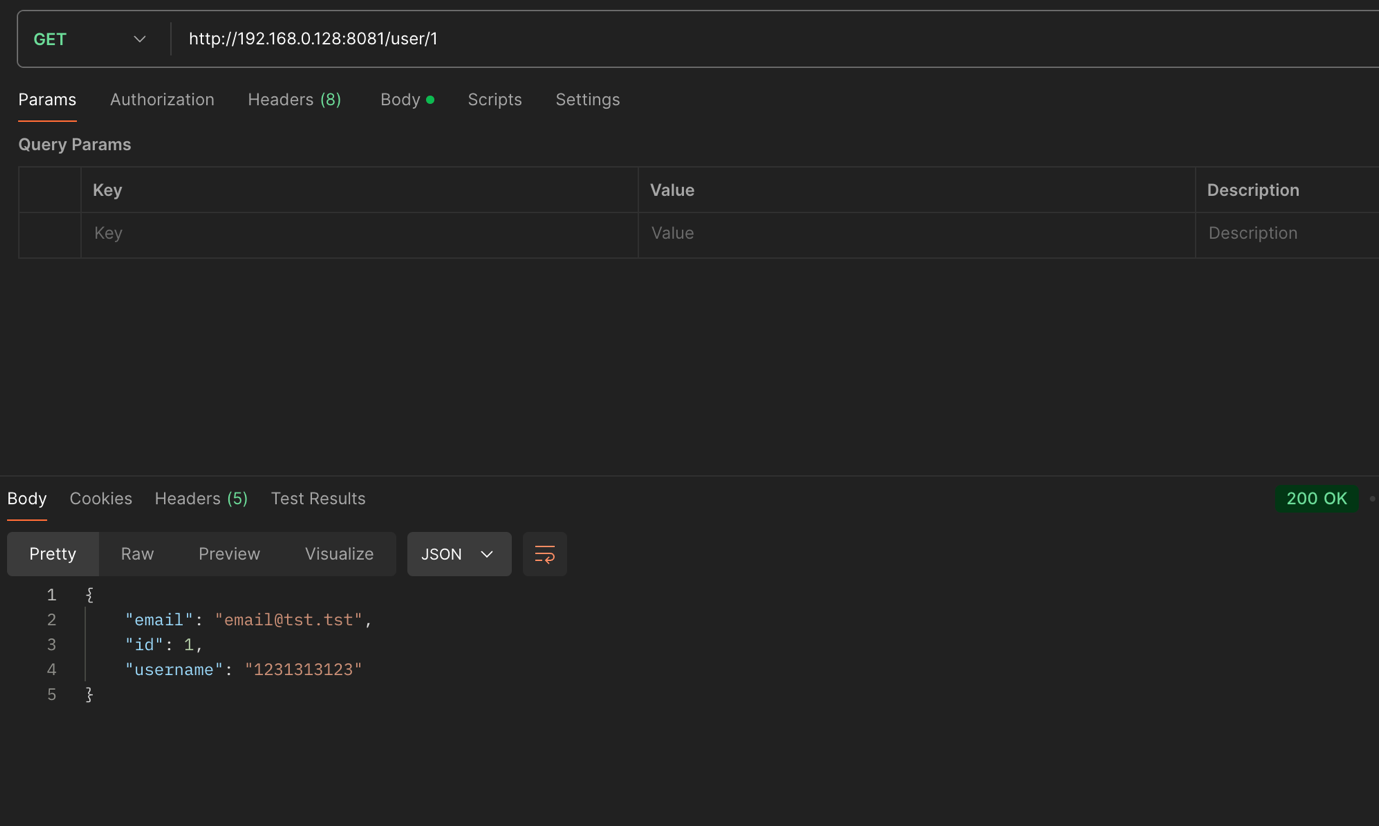
1. Кешируем пользователя отправив запрос



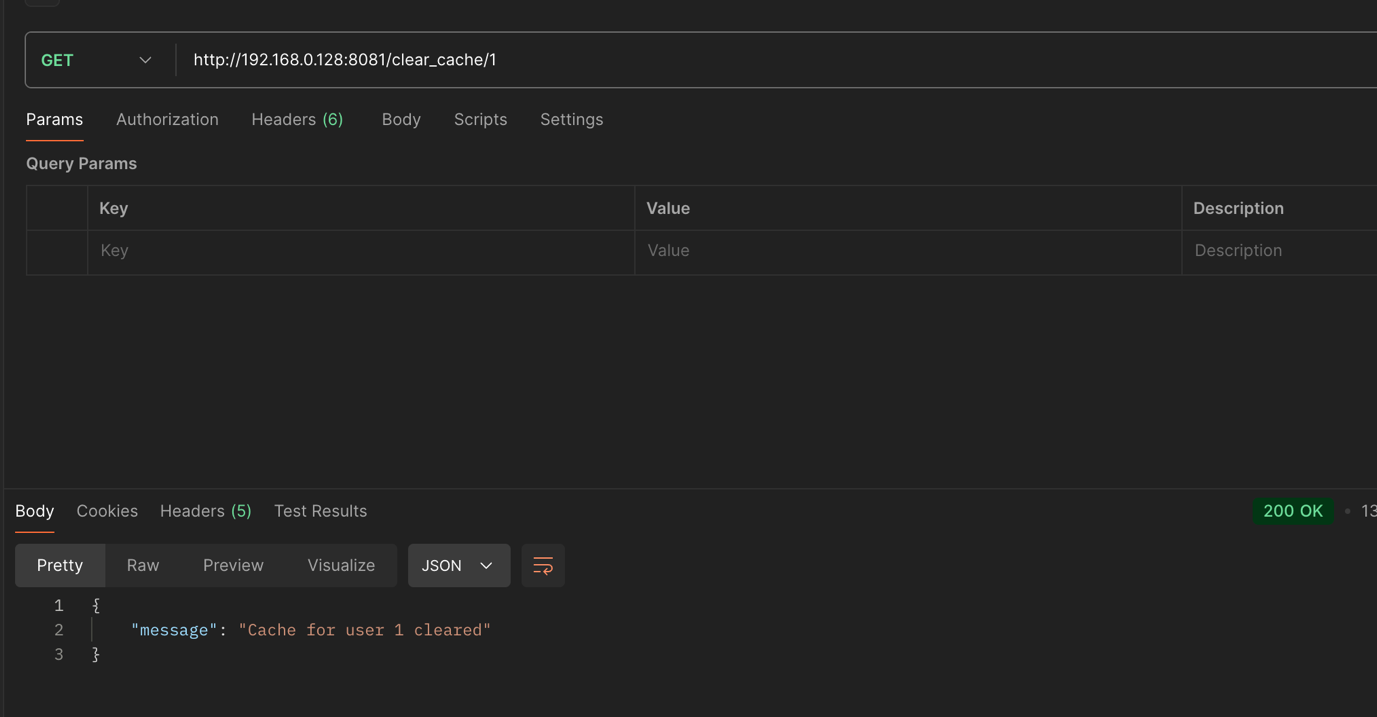
1. Меняем пользователю username



1. Проверка, что пользователь закеширован



1. Чистим кеш для пользователя



1. Проверяем что кеш почистился

