Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

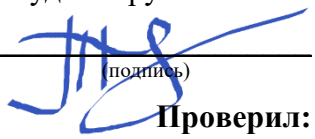
Факультет безопасности информационных технологий

Дисциплина:
«Web программирование»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Выполнил:

Рядовой Т.С., студент группы N3352



(подпись)

Проверил:
Менщиков Александр Алексеевич

(отметка о выполнении)

(подпись)

Санкт-Петербург
2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: разработка панели администратора для управления базой данных на языках программирования Python с использованием веб-фреймровка FastAPI и React.

1 ОПИСАНИЕ АРХИТЕКТУРЫ ПО

Проект представляет собой вею-приложение для администрации базой данных, состоящий из трех сервисов:

1. api – простейший JSON REST API на Python FastAPI, который позволяет управлять записями;
2. frontend – собранный статический сайт на React;
3. db - база данных, с которой взаимодействует API.

Хранение данных осуществляется в PostgreSQL. Проект упакован в Docker, с использованием docker-compose, где каждый компонент развернут в своем контейнере.

Компоненты взаимодействуют следующим образом:

- Пользователь заходит на сайт (раздаётся через Nginx);
- React-приложение запрашивает данные у API;
- API обращается к БД, запрашивая соответствующую информацию;
- Данные передаются обратно во frontend и отображаются в виде таблицы;
- Реализована поддержка пагинации и изменение количества отображаемых записей на странице.

1.1 Сервис api

Задачи:

- Выдача данных только в JSON;
- Поддержка пагинации с произвольным размером страницы;
- Доступ к БД с использованием ORM библиотек;
- Поддержка HTTP методов: GET, POST, DELETE, PATCH;

Технологии:

- FastAPI – фреймворк для Python;
- JSON – формат выдачи данных.
- SQLAlchemy – ORM для работы с PostgreSQL

1.2 Сервис frontend

Задачи:

- Должен быть разработан на языке JavaScript;
- Должен использоваться React;
- Должен запрашивать данные с использованием REST API;
- Поддержка пагинации с произвольным размером страницы;

– Итоговый бандл должен быть собран в виде статических файлов. Раздача с помощью Nginx.

Технологии:

– HTML, JavaScript – фронтенд;
– React – библиотека для разработки UI;
– Nginx – раздача статических файлов;
– Tailwind – CSS библиотека;

1.3 База данных

Выбрана PostgreSQL. Доступ к ней осуществляется по логину и паролю.

```
→ psql -h localhost -p 5433 -U fastapi_newbie -d lab2_db
Пароль пользователя fastapi_newbie:
psql (16.4 (Homebrew), сервер 15.11 (Debian 15.11-1.pgdg120+1))
Введите "help", чтобы получить справку.

lab2_db=# select * from students;
 id | last_name  | first_name |  middle_name   | course | group_name |           faculty
----+------------+------------+----------------+--------+------------+-----------------------------
  1 | Сидоров    | Сидор      |                |      1 | В-11       | Химический
  2 | Смирнов    | Алексей    |                |      4 | Г-42       | Биологический
  3 | Иванова    | Анна       | Сергеевна      |      2 | Д-23       | Географический
  4 | Кузнецов   | Дмитрий    | Андреевич      |      3 | Е-34       | Исторический
  5 | Попова     | Елена      | Владимировна   |      1 | Ж-15       | Филологический
  6 | Васильев   | Сергей     | Николаевич     |      4 | З-46       | Экономический
  7 | Петрова    | Ольга      | Александровна  |      2 | И-27       | Юридический
  8 | Михайлов   | Андрей     | Дмитриевич     |      3 | К-38       | Социологический
  9 | Федорова   | Наталья    | Ивановна       |      1 | Л-19       | Психологический
 10 | Соколов    | Игорь      | Сергеевич      |      4 | М-40       | Культурологический
 11 | Карпов     | Виктор     | Петрович       |      2 | Н-21       | Политологический
 12 | Морозов    | Александр  | Андреевич      |      3 | О-32       | Журналистика
 13 | Павлов     | Николай    | Владимирович   |      1 | П-13       | Международные отношения
 14 | Козлов     | Денис      | Сергеевич      |      4 | Р-44       | Туризм
 15 | Степанов   | Евгений    | Николаевич     |      2 | С-25       | Гостиничное дело
 16 | Андреев    | Антон      | Дмитриевич     |      3 | Т-36       | Ресторанное дело
 17 | Белов      | Максим     | Иванович       |      1 | У-17       | Маркетинг
 18 | Виноградов | Кирилл     | Сергеевич      |      4 | Ф-48       | Реклама
 19 | Герасимов  | Владимир   | Петрович       |      2 | Х-29       | PR
 20 | Дмитриев   | Илья       | Андреевич      |      3 | Ц-30       | IT
 21 | Егоров     | Артем      | Владимирович   |      1 | Ч-11       | Информационная безопасность
 22 | Жуков      | Константин | Сергеевич      |      4 | Ш-42       | Программирование
 23 | Зайцев     | Павел      | Николаевич     |      2 | Щ-23       | Веб-разработка
 24 | Иванов     | Игорь      | Дмитриевич     |      3 | Э-34       | Дизайн
 25 | Королев    | Сергей     | Иванович       |      1 | Ю-15       | Архитектура
 26 | Лазарев    | Андрей     | Сергеевич      |      4 | Я-46       | Строительство
 27 | Максимов   | Дмитрий    | Петрович       |      2 | АА-27      | Машиностроение
 28 | Николаев   | Александр  | Андреевич      |      3 | ББ-38      | Авиастроение
 29 | Орлов      | Виктор     | Владимирович   |      1 | ВВ-19      | Кораблестроение
 30 | Петров     | Петр       | Сергеевич      |      4 | ГГ-40      | Энергетика
 31 | Романов    | Роман      | Николаевич     |      2 | ДД-21      | Металлургия
 32 | Сидоров    | Сидор      | Дмитриевич     |      3 | ЕЕ-32      | Химическая промышленность
 33 | Тихонов    | Андрей     | Иванович       |      1 | ЖЖ-13      | Легкая промышленность
 34 | Филиппов   | Сергей     | Сергеевич      |      4 | ЗЗ-44      | Пищевая промышленность
 35 | Харламов   | Валерий    | Петрович       |      2 | ИИ-25      | Сельское хозяйство
 36 | Чернов     | Алексей    | Андреевич      |      3 | КК-36      | Лесное хозяйство
 37 | Шарапов    | Дмитрий    | Владимирович   |      1 | ЛЛ-17      | Рыбное хозяйство
 38 | Щербаков   | Игорь      | Сергеевич      |      4 | ММ-48      | Охотничье хозяйство
 39 | Юрьев      | Юрий       |                |      2 | НН-29      | Геология
 40 | Яковлев    | Яков       |                |      3 | ОО-30      | География
 41 | Виноградов | Сергей     | Павлович       |      3 | А-31       | IT
 42 | Соколов    | Максим     | Иванович       |      3 | А-31       | IT
(42 строки)

lab2_db=#
```

Рисунок 1 – Подключение к базе данных локально

Листинг 1 – init.sql

```
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    middle_name VARCHAR(255),
    course INTEGER NOT NULL,
    group_name VARCHAR(255) NOT NULL,
    faculty VARCHAR(255) NOT NULL
);

INSERT INTO students (last_name, first_name, middle_name, course, group_name,
faculty) VALUES
('Сидоров', 'Сидор', NULL, 1, 'В-11', 'Химический'),
('Смирнов', 'Алексей', NULL, 4, 'Г-42', 'Биологический'),
('Иванова', 'Анна', 'Сергеевна', 2, 'Д-23', 'Географический'),
('Кузнецов', 'Дмитрий', 'Андреевич', 3, 'Е-34', 'Исторический'),
('Попова', 'Елена', 'Владимировна', 1, 'Ж-15', 'Филологический'),
('Васильев', 'Сергей', 'Николаевич', 4, 'З-46', 'Экономический'),
('Петрова', 'Ольга', 'Александровна', 2, 'И-27', 'Юридический'),
('Михайлов', 'Андрей', 'Дмитриевич', 3, 'К-38', 'Социологический'),
('Федорова', 'Наталья', 'Ивановна', 1, 'Л-19', 'Психологический'),
('Соколов', 'Игорь', 'Сергеевич', 4, 'М-40', 'Культурологический'),
('Карпов', 'Виктор', 'Петрович', 2, 'Н-21', 'Политологический'),
('Морозов', 'Александр', 'Андреевич', 3, 'О-32', 'Журналистика'),
('Павлов', 'Николай', 'Владимирович', 1, 'П-13', 'Международные отношения'),
('Козлов', 'Денис', 'Сергеевич', 4, 'Р-44', 'Туризм'),
('Степанов', 'Евгений', 'Николаевич', 2, 'С-25', 'Гостиничное дело'),
('Андреев', 'Антон', 'Дмитриевич', 3, 'Т-36', 'Ресторанное дело'),
('Белов', 'Максим', 'Иванович', 1, 'У-17', 'Маркетинг'),
('Виноградов', 'Кирилл', 'Сергеевич', 4, 'Ф-48', 'Реклама'),
('Герасимов', 'Владимир', 'Петрович', 2, 'Х-29', 'PR'),
('Дмитриев', 'Илья', 'Андреевич', 3, 'Ц-30', 'IT'),
('Егоров', 'Артем', 'Владимирович', 1, 'Ч-11', 'Информационная
безопасность'),
('Жуков', 'Константин', 'Сергеевич', 4, 'Ш-42', 'Программирование'),
('Зайцев', 'Павел', 'Николаевич', 2, 'Щ-23', 'Веб-разработка'),
('Иванов', 'Игорь', 'Дмитриевич', 3, 'Э-34', 'Дизайн'),
('Королев', 'Сергей', 'Иванович', 1, 'Ю-15', 'Архитектура'),
('Лазарев', 'Андрей', 'Сергеевич', 4, 'Я-46', 'Строительство'),
('Максимов', 'Дмитрий', 'Петрович', 2, 'АА-27', 'Машиностроение'),
('Николаев', 'Александр', 'Андреевич', 3, 'ББ-38', 'Авиастроение'),
('Орлов', 'Виктор', 'Владимирович', 1, 'ВВ-19', 'Кораблестроение'),
('Петров', 'Петр', 'Сергеевич', 4, 'ГГ-40', 'Энергетика'),
('Романов', 'Роман', 'Николаевич', 2, 'ДД-21', 'Металлургия'),
('Сидоров', 'Сидор', 'Дмитриевич', 3, 'ЕЕ-32', 'Химическая промышленность'),
('Тихонов', 'Андрей', 'Иванович', 1, 'ЖЖ-13', 'Легкая промышленность'),
('Филиппов', 'Сергей', 'Сергеевич', 4, 'ЗЗ-44', 'Пищевая промышленность'),
('Харламов', 'Валерий', 'Петрович', 2, 'ИИ-25', 'Сельское хозяйство'),
('Чернов', 'Алексей', 'Андреевич', 3, 'КК-36', 'Лесное хозяйство'),
('Шарапов', 'Дмитрий', 'Владимирович', 1, 'ЛЛ-17', 'Рыбное хозяйство'),
('Щербаков', 'Игорь', 'Сергеевич', 4, 'ММ-48', 'Охотничье хозяйство'),
('Юрьев', 'Юрий', NULL, 2, 'НН-29', 'Геология'),
('Яковлев', 'Яков', NULL, 3, 'ОО-30', 'География'),
('Виноградов', 'Сергей', 'Павлович', 3, 'А-31', 'IT'),
('Соколов', 'Максим', 'Иванович', 3, 'А-31', 'IT');
```

Листинг 2 – Фрагмент из docker-compose.yml для настройки базы данных

```
db:
    image: postgres:15
    container_name: postgres-lab1
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5433:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
      interval: 1s
      timeout: 5s
      retries: 10
```

Добавляем healthcheck, чтобы база данных успела запуститься и записать данные в таблицу students, далее в каждом сервисе из docker-compose прописываем, что они в свою очередь запускаются после завершения конфигурации сервиса db.

1.4 Веб-сервер (Nginx)

Функции:

- Реализует SPA (single page app), перенаправляя все запросы на index.html;
- Проксирование запросов к сервису api.

Листинг 3 – Конфигурация сервера Nginx

```
server {
    listen 80;
    server_name localhost;

    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api/ {
        proxy_pass http://api:3000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

2 ОПИСАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ

В проекте используется база данных PostgreSQL. Она хранит информацию о студентах. Все данные управляются через ORM (SQLAlchemy) для удобства взаимодействия с кодом на Python.

2.1 Таблица students (Студенты)

Содержит данные о зарегистрированных студентах.

Таблица 1 – Таблица пользователей

Поле	Тип данных	Описание
id	SERIAL PRIMARY KEY	Уникальный идентификатор студента
last_name	VARCHAR(255) NOT NULL	Фамилия студента
first_name	VARCHAR(255) NOT NULL	Имя студента
middle_name	VARCHAR(255)	Отчество студента
course	INTEGER NOT NUL	Курс
group_name	VARCHAR(255) NOT NULL	Имя группы
faculty	VARCHAR(255) NOT NULL	Факультет

3 ОПИСАНИЕ ПРОТОКОЛА И ФОРМАТОВ ПЕРЕДАЧИ ДАННЫХ

В проекте используется протокол передачи данных HTTP (REST API) для поиска, создания, удаления, редактирования студентов.

3.1 HTTP

HTTP-протокол используется для взаимодействия между клиентом и сервером. Все данные передаются в формате JSON.

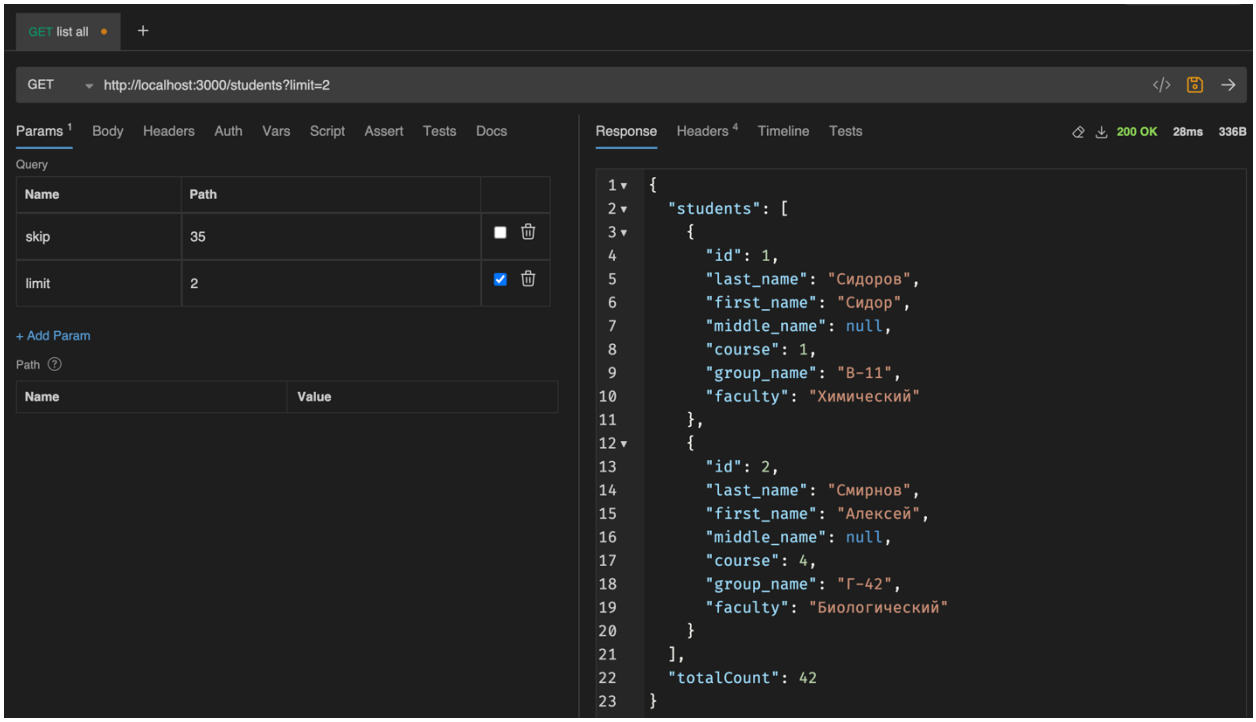


Рисунок 2 – Пример GET-запроса

4 ОПИСАНИЕ API

API реализовано в виде RESTful-сервиса с передачей данных в формате JSON.

4.1 Эндпоинт students

Выдает список всех студентов и их количество (totalCount). Также есть поддержка параметров skip и limit.

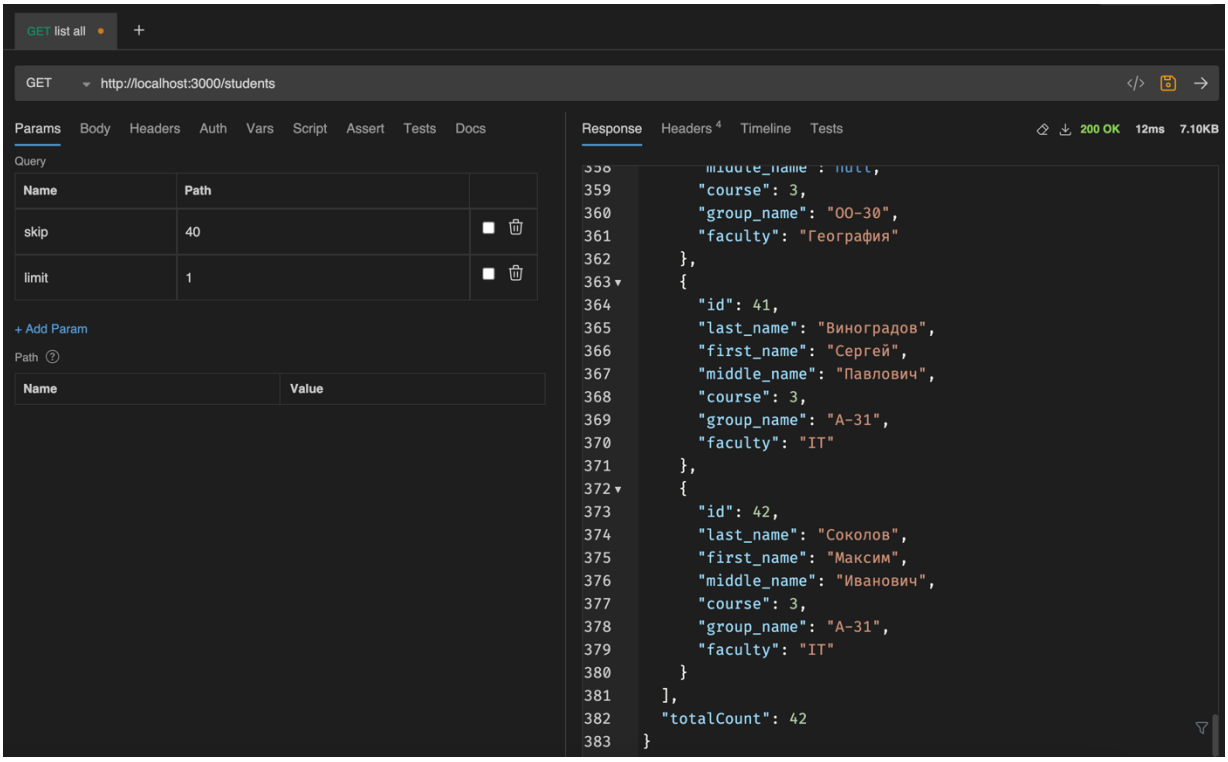


Рисунок 3 – Успешный GET-запрос эндпоинта students без параметров

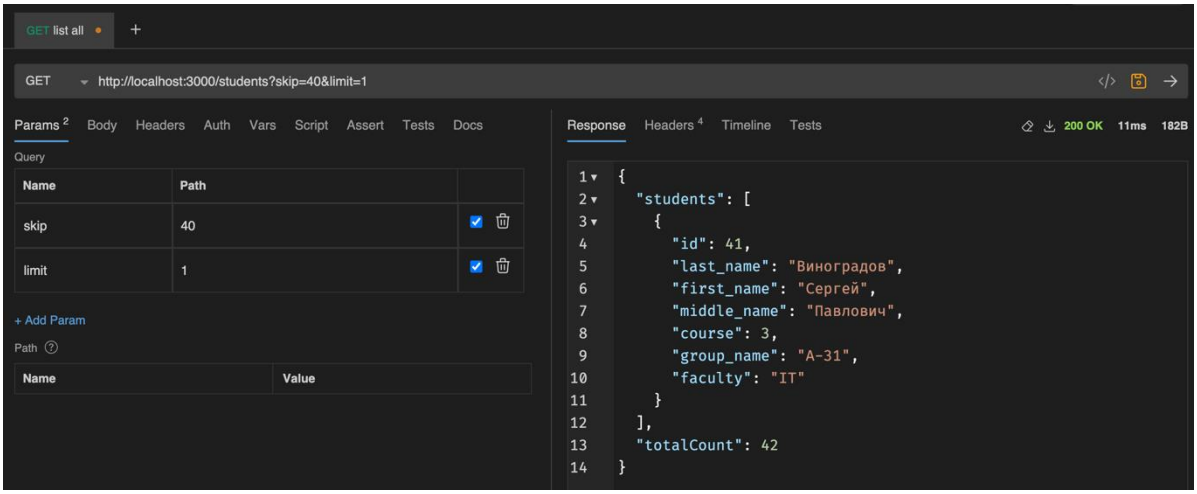


Рисунок 4 – Успешный GET-запрос эндпоинта students с параметрами

4.2 Эндпоинт student/create

Пример успешной и неуспешной попытки создания студента.

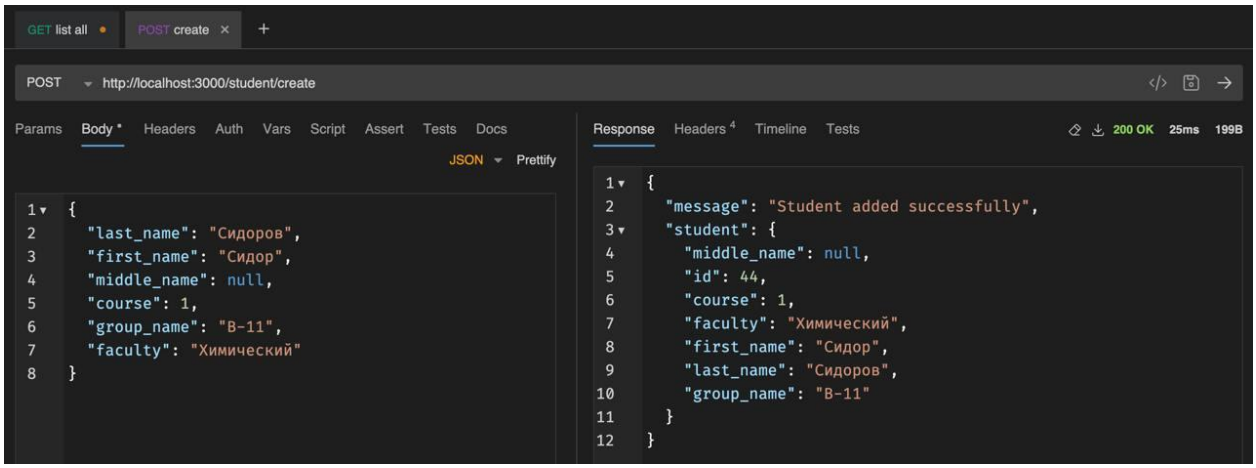


Рисунок 5 – Успешный POST-запрос эндпоинта student/create

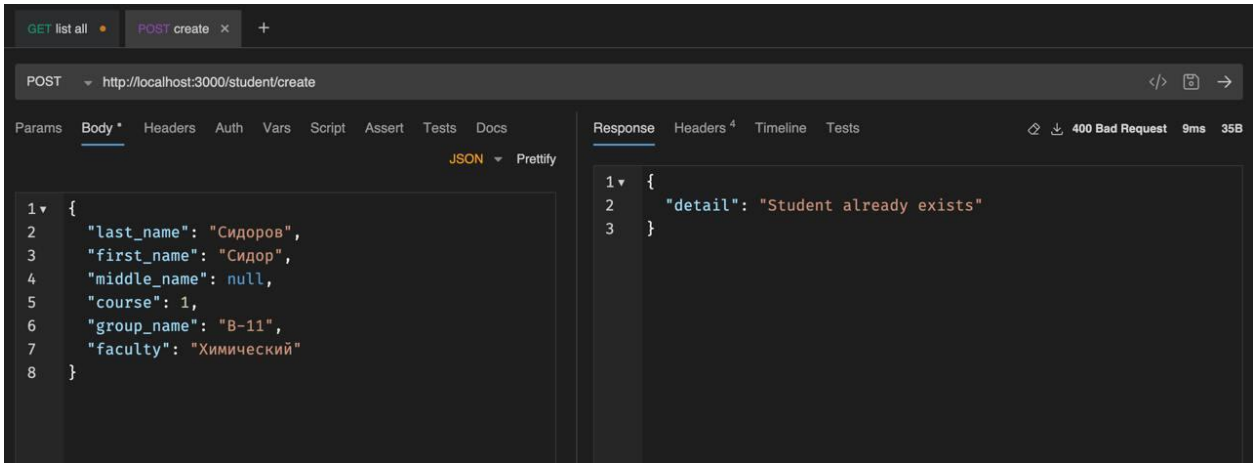


Рисунок 6 – Неуспешный POST-запрос эндпоинта student/create

4.3 Эндпоинт student/delete

Пример успешной и неуспешной попытки удаления студента.

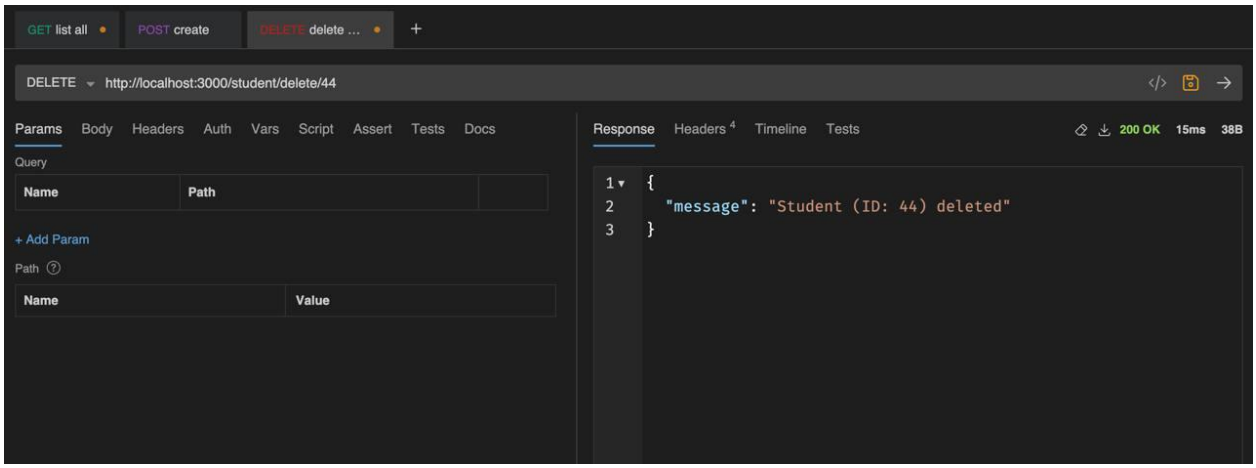


Рисунок 7 – Успешный DELETE-запрос эндпоинта student/delete

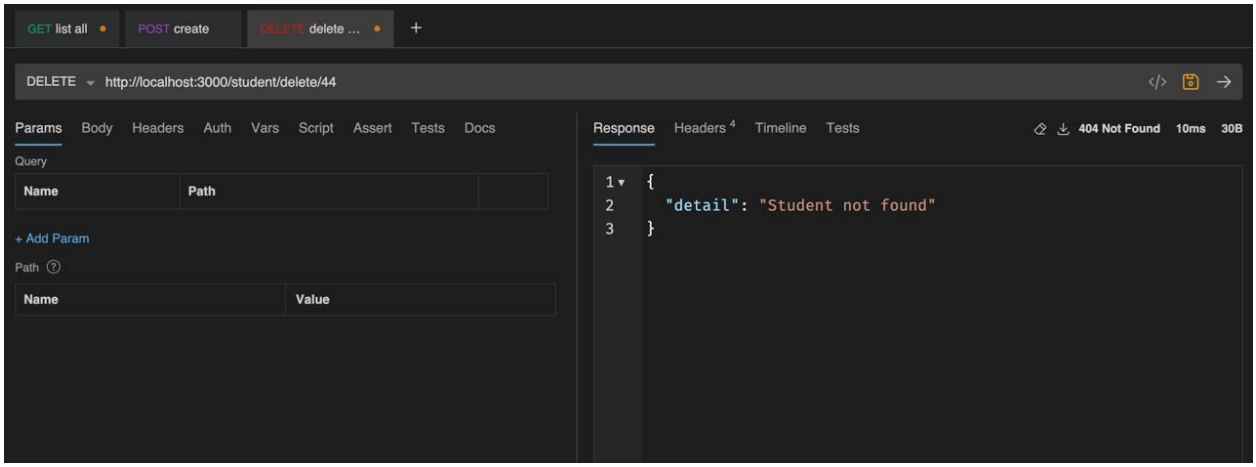


Рисунок 8 – Неуспешный (повторный) DELETE-запрос эндпоинта student/delete

4.4 Эндпоинт student/update

Пример успешной и неуспешной попытки редактирования студента.

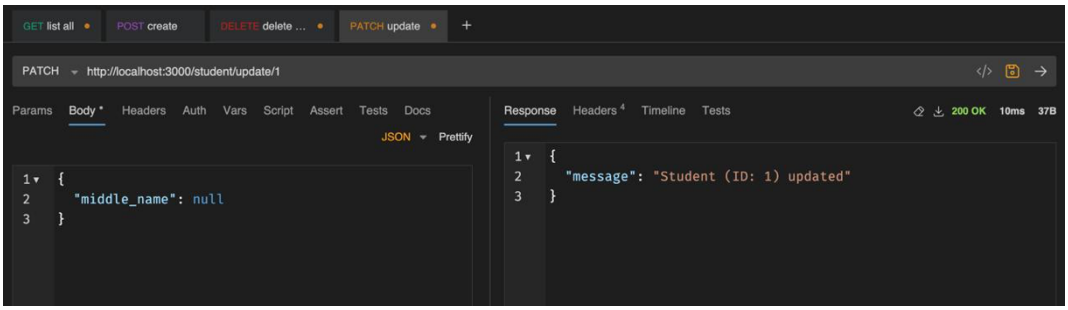


Рисунок 9 – Успешный PATCH-запрос эндпоинта student/update

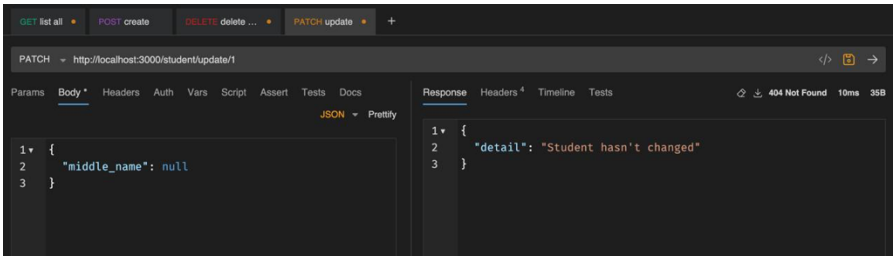


Рисунок 10 – Неуспешный PATCH-запрос эндпоинта student/update

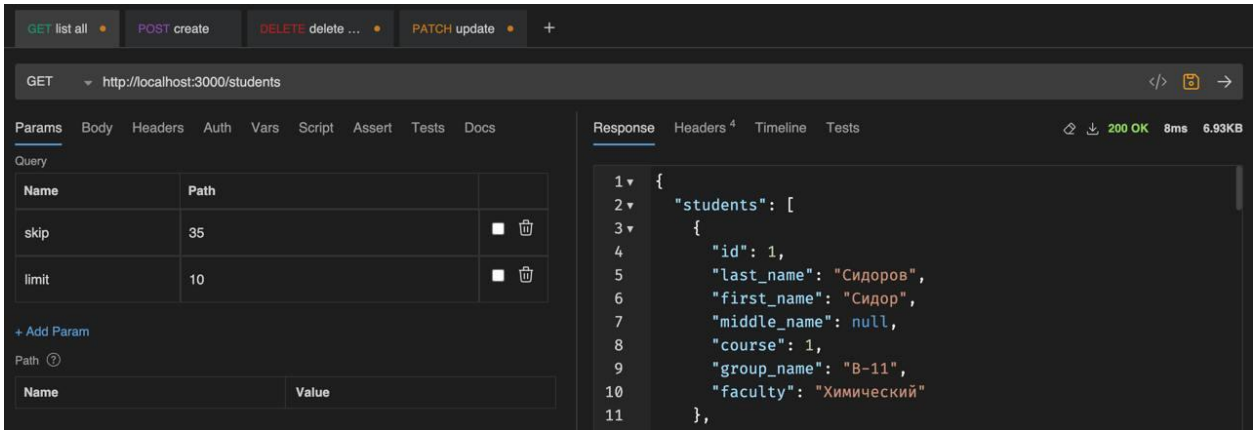


Рисунок 11 – Проверка обновления студента

4.5 Эндпоинт students/search

Пример успешной и неуспешной попытки поиска студентов. Также есть поддержка параметров skip и limit.

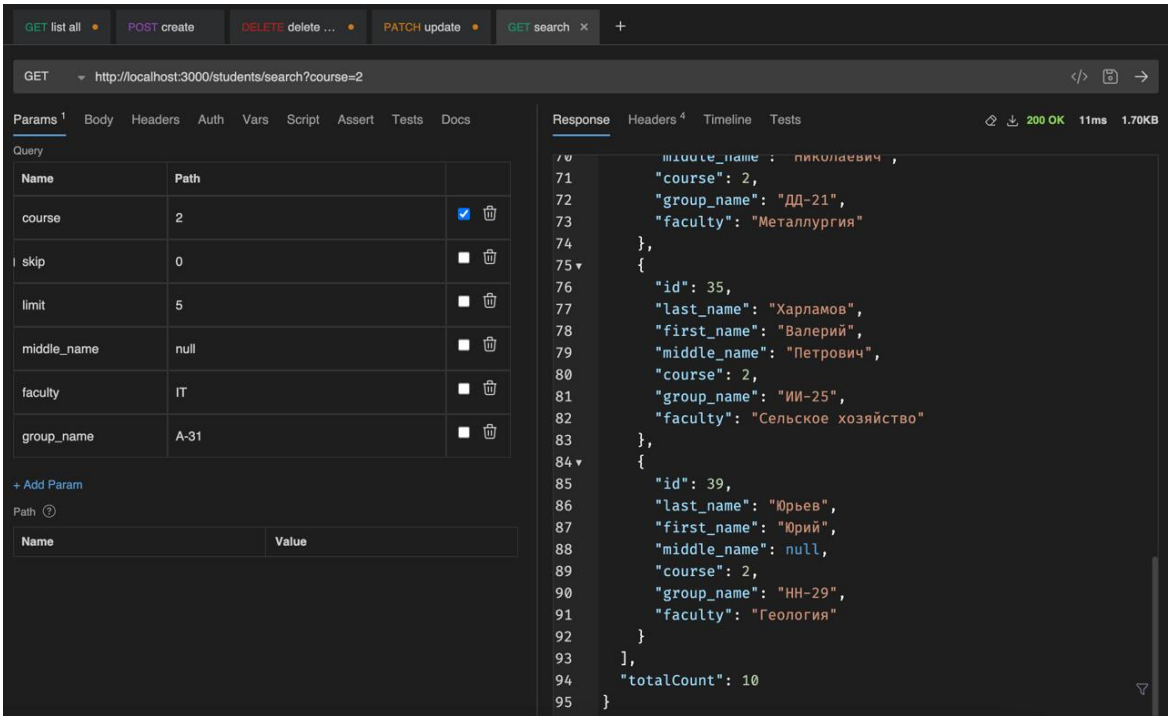


Рисунок 12 – Успешный GET-запрос эндпоинта students/search без дополнительных параметров

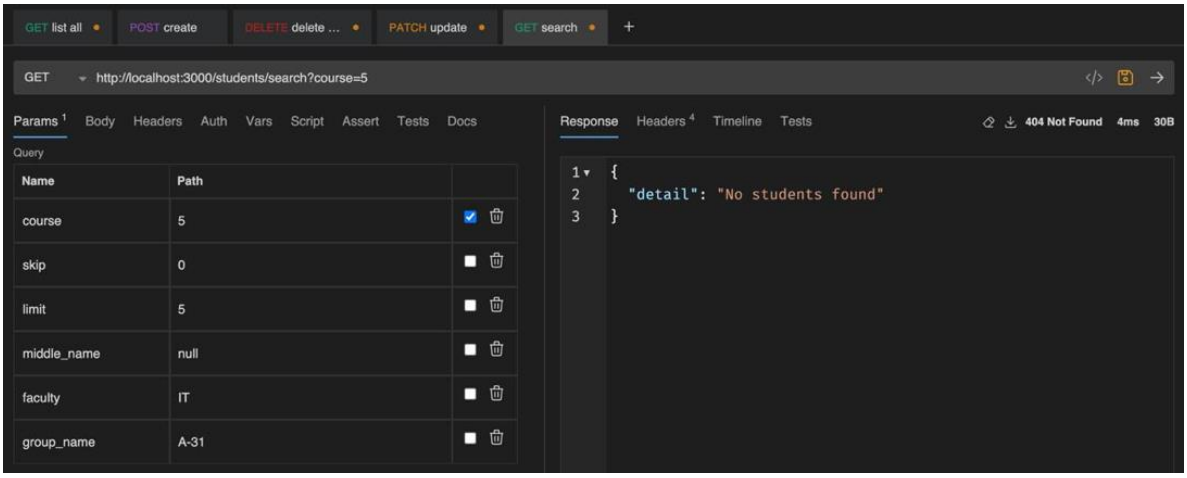


Рисунок 13 – Неуспешный GET-запрос эндпоинта students/search (нет студентов на 5 курсе)

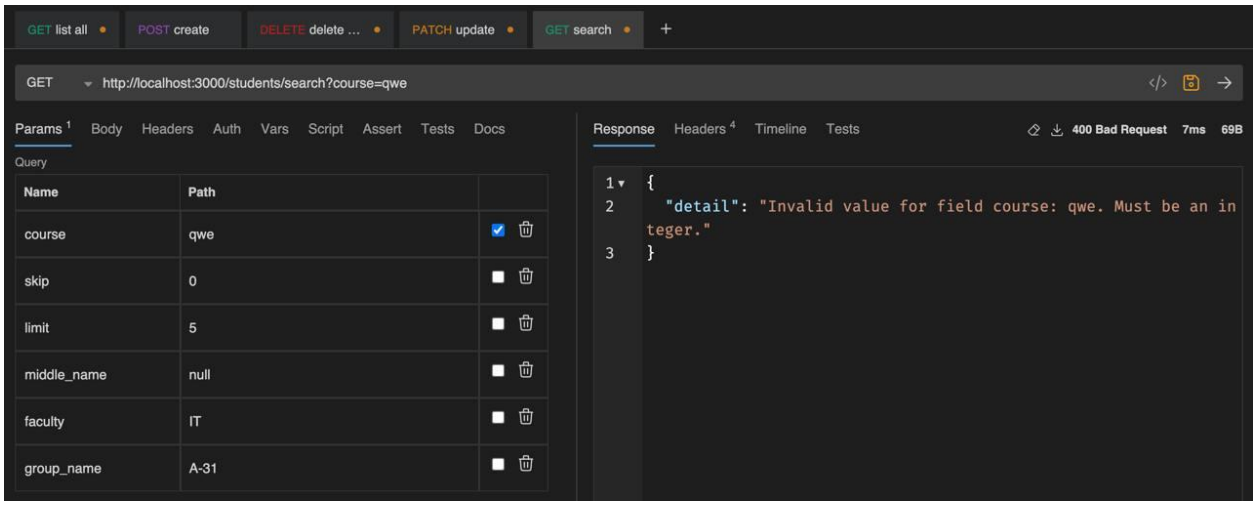


Рисунок 14 – Неуспешный GET-запрос эндпоинта students/search (параметр с поле course – не цифра)

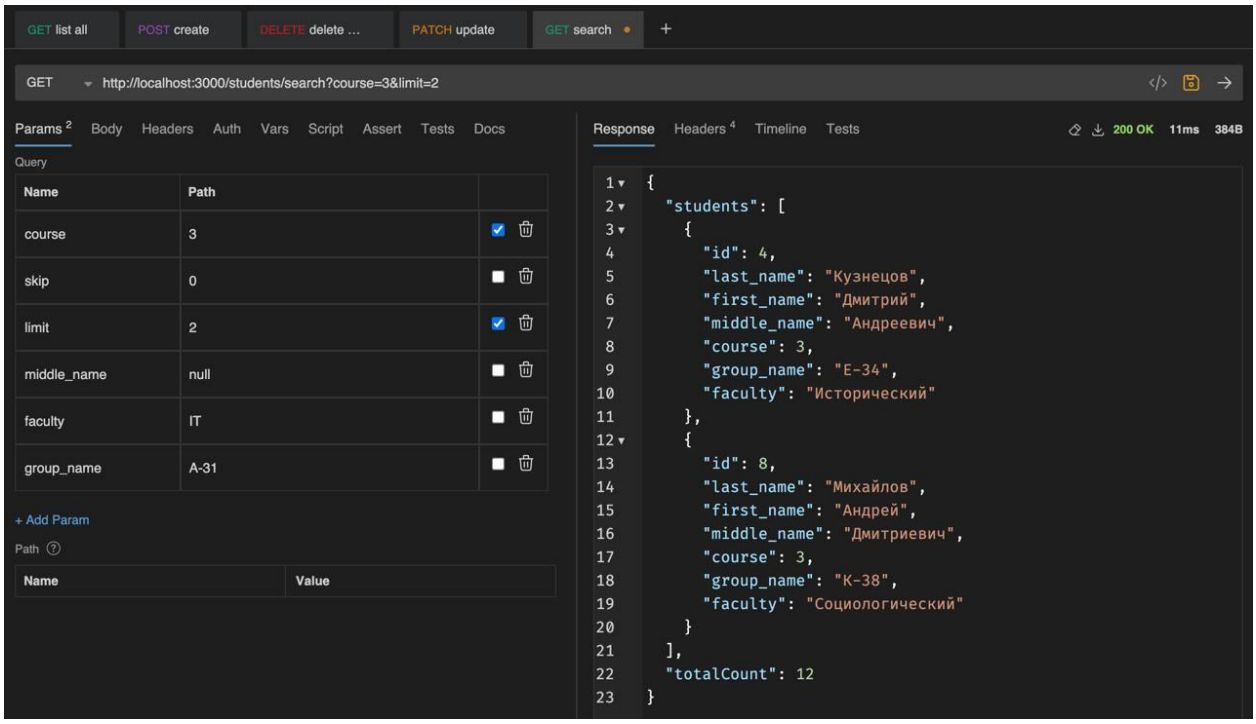


Рисунок 15 – Успешный GET-запрос эндпоинта students/search с дополнительными параметрами

4.6 Пример логов

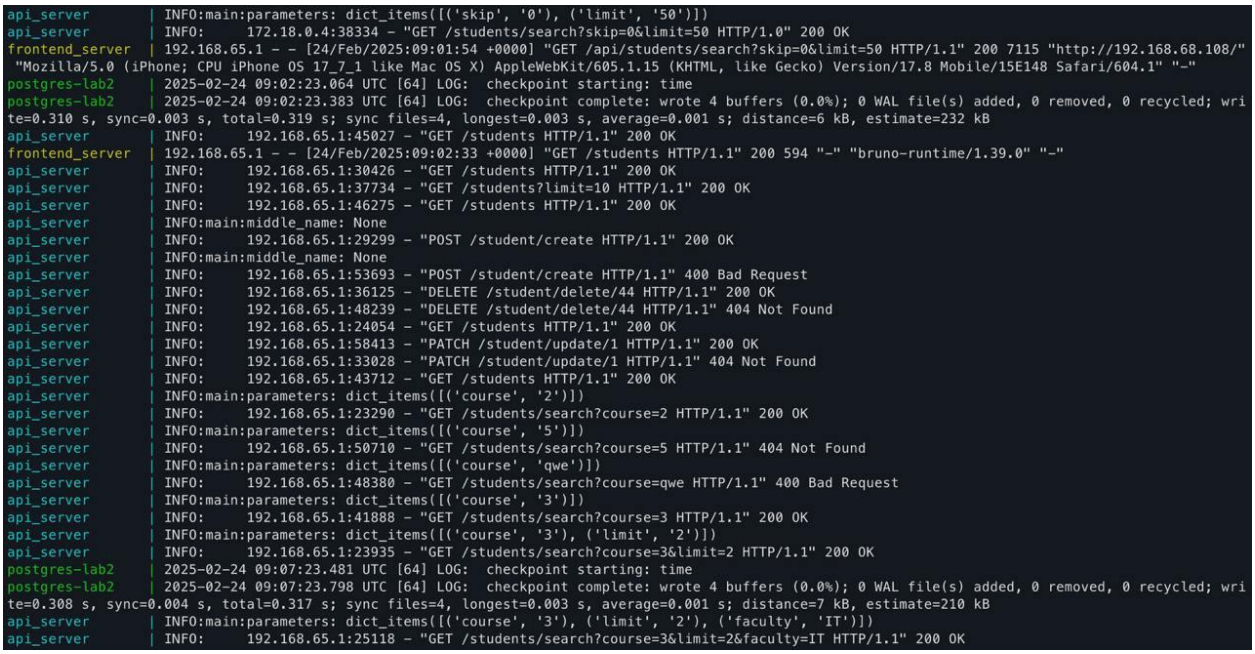


Рисунок 16 – Пример логов после запуска контейнеров

5 ПРИМЕРЫ ФУНКЦИОНАЛА

Рисунок 17 – Пример домашней страницы (5 записей на странице)

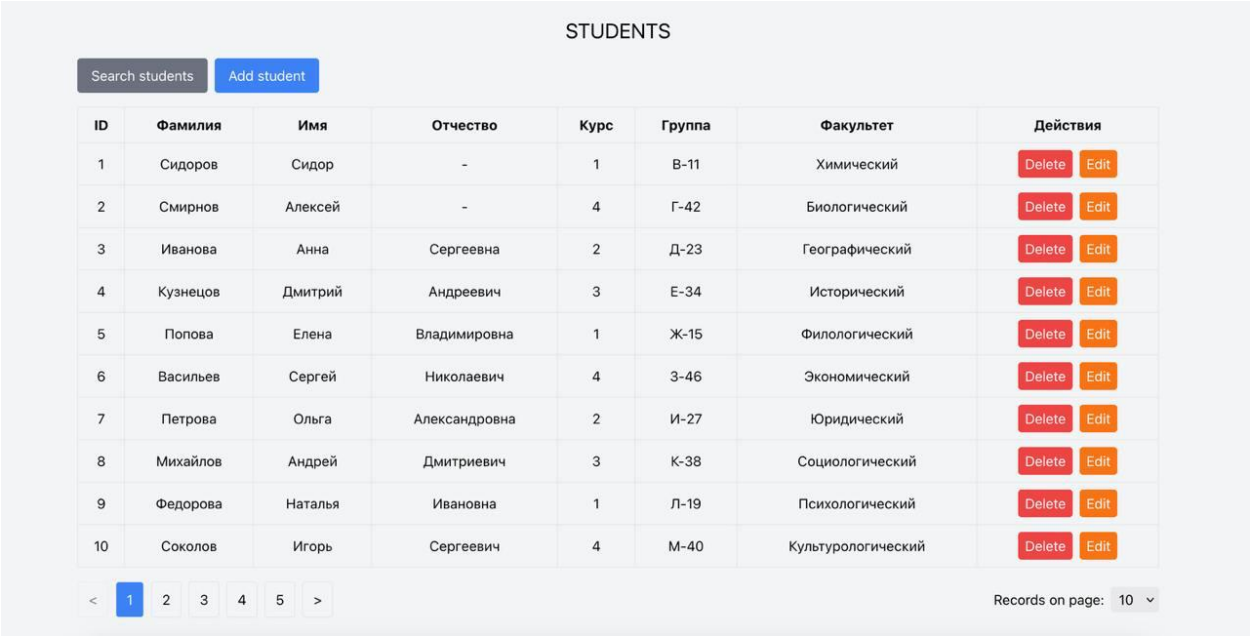


Рисунок 18 – Пример домашней страницы (10 записей на странице)

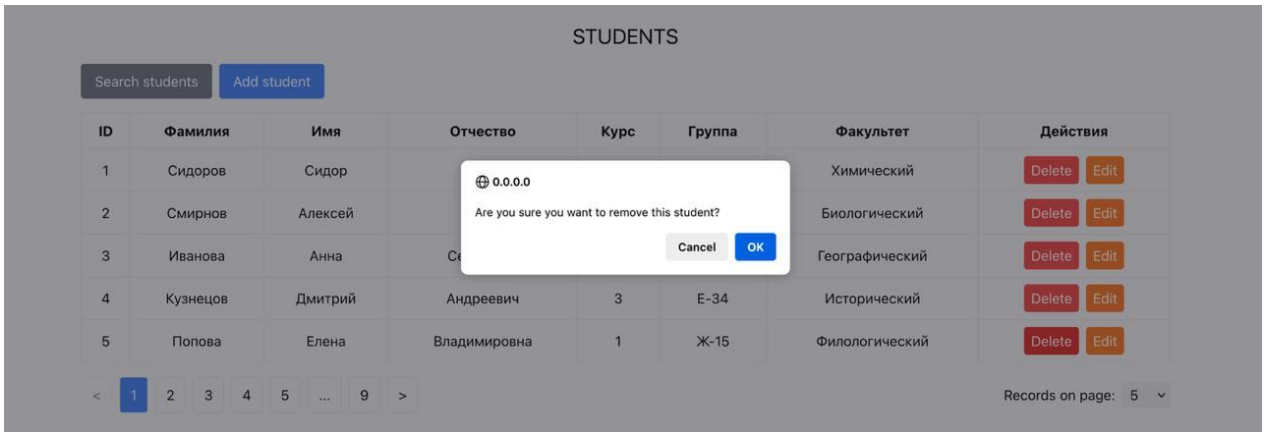


Рисунок 19 – Попытка удаления студента

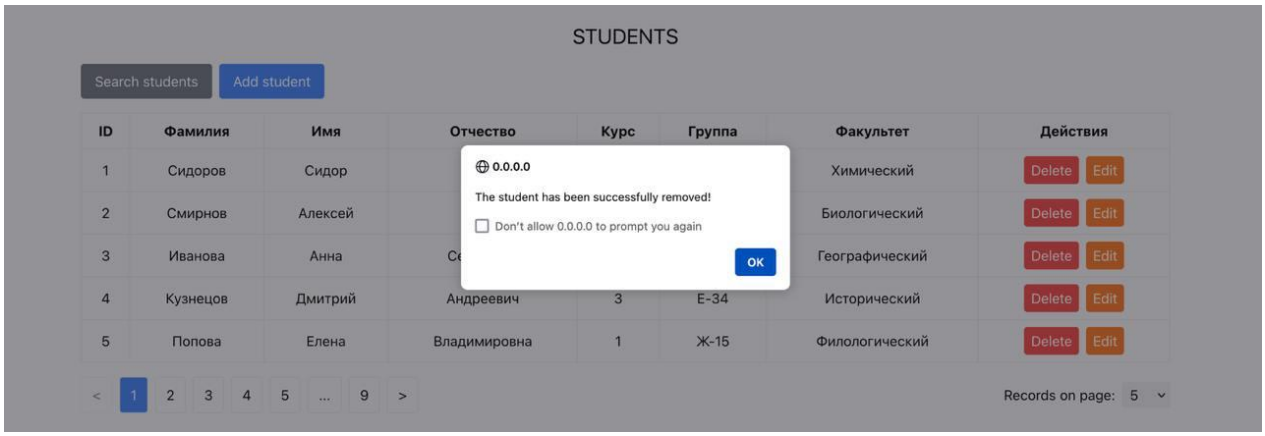


Рисунок 20 – Уведомление об успешном удалении студента

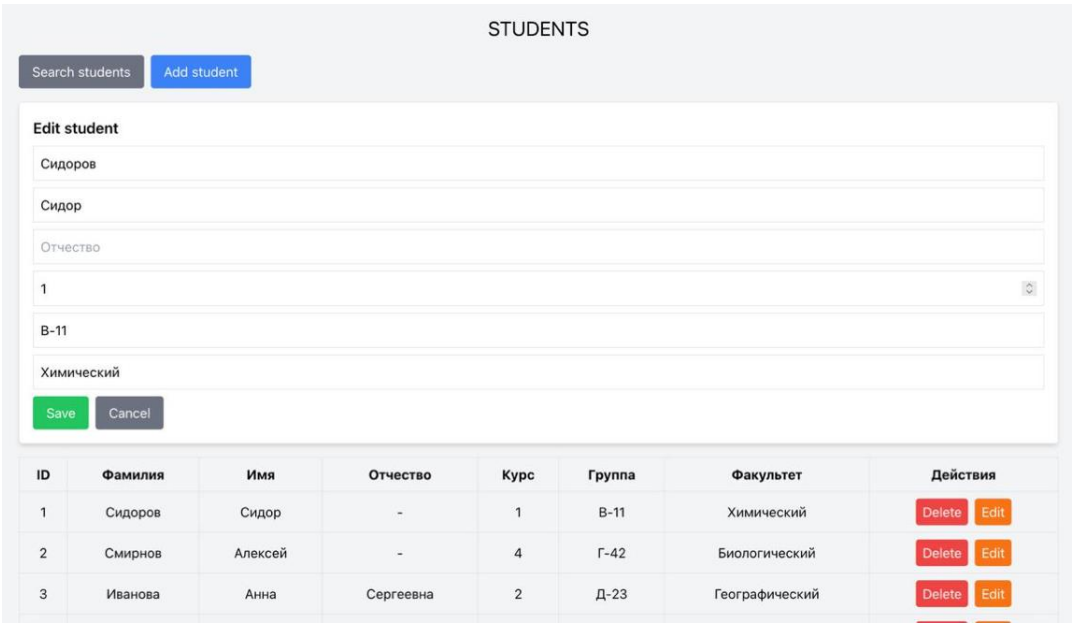


Рисунок 21 – Окно редактирования студента

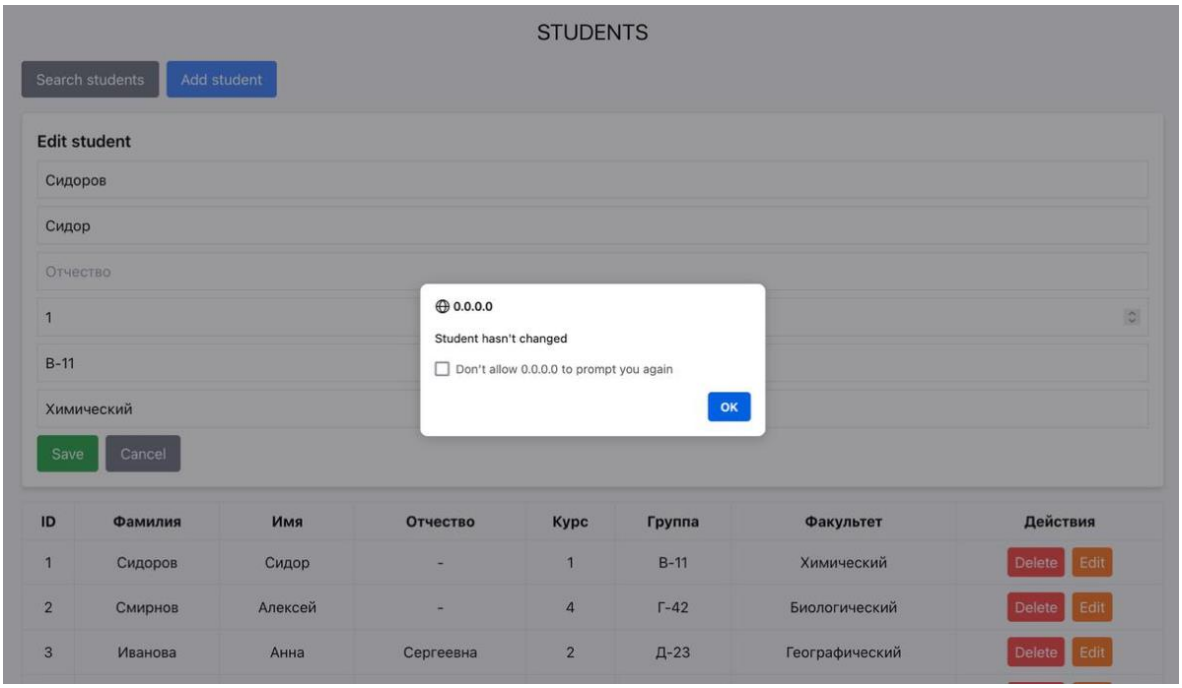


Рисунок 22 – Попытка (неуспешная) сохранить студента, ничего не поменяв

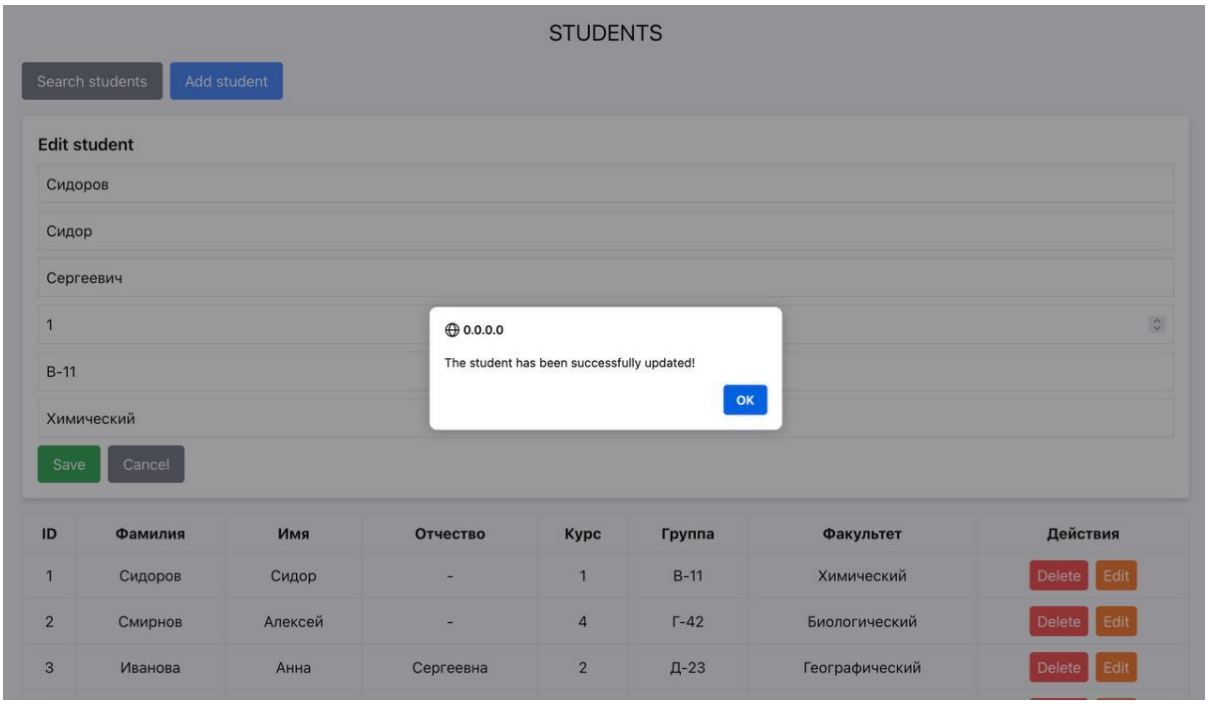


Рисунок 23 – Добавляем отчество – получаем отчество об успешном обновлении студента

Рисунок 24 – Проверка обновления студента (добавили отчество первому студенту)

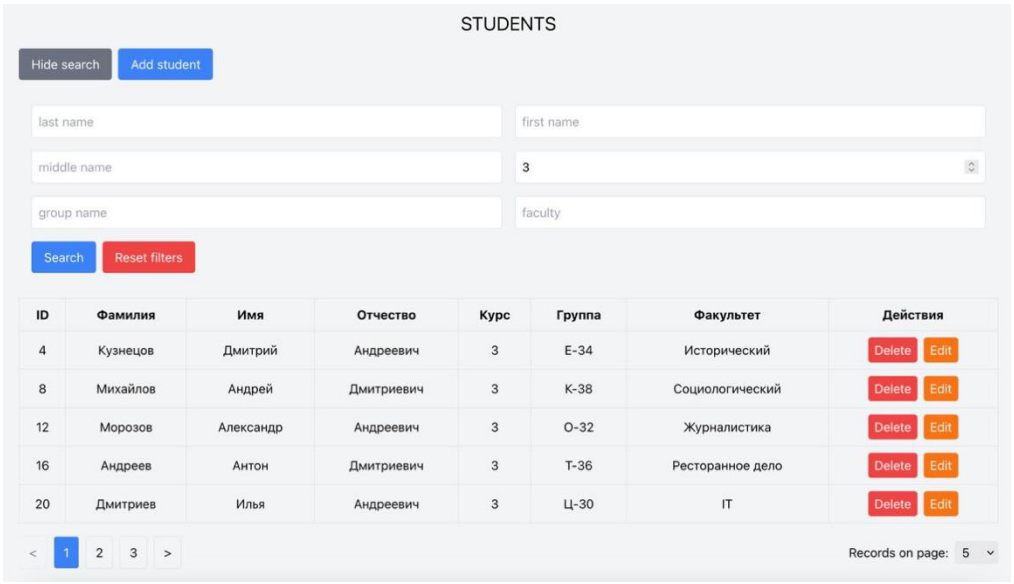


Рисунок 25 – Поиск всех студентов с третьего курса

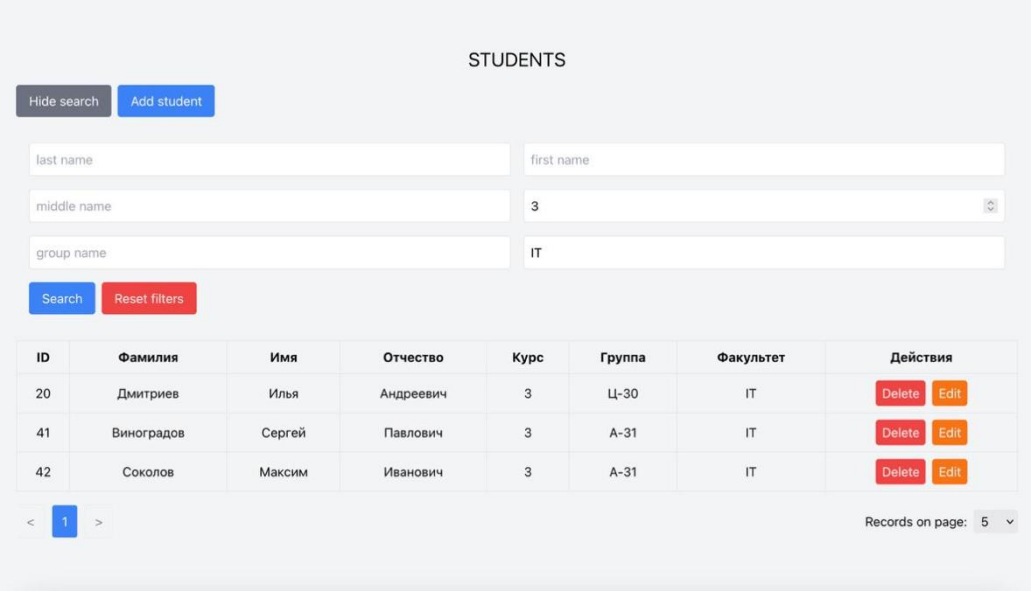


Рисунок 26 – Поиск всех студентов с третьего курса и факультета IT

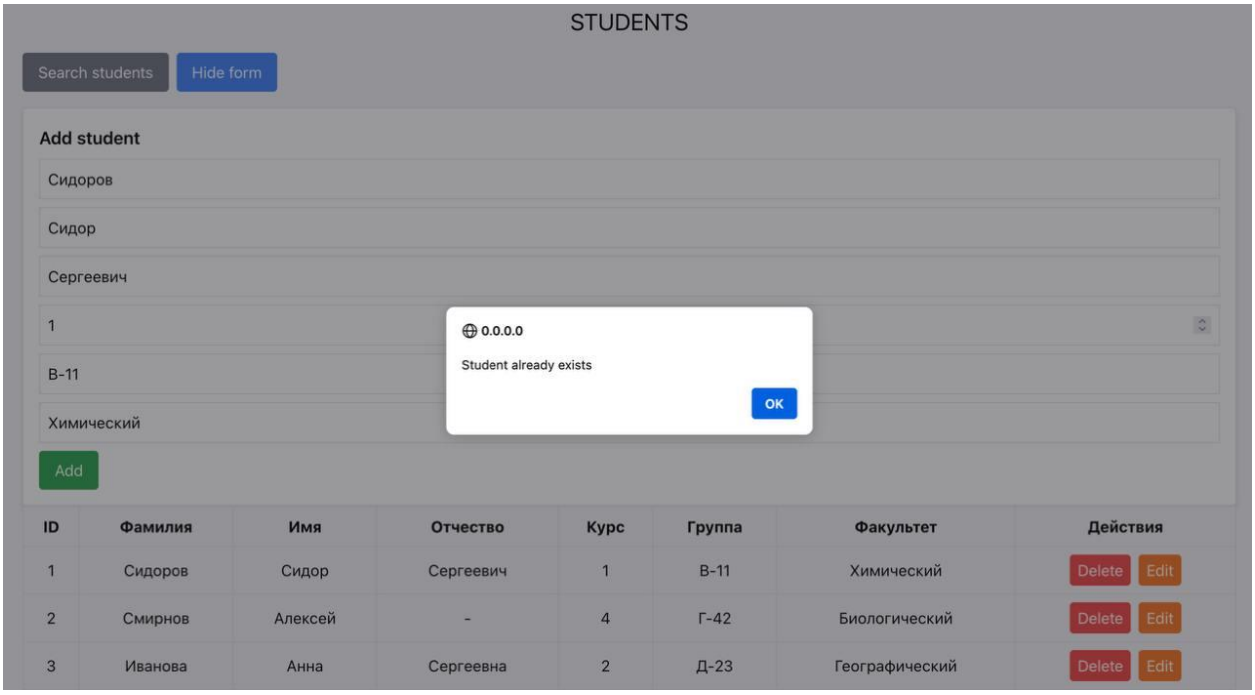


Рисунок 27 – Неуспешная попытка добавить студента (совпадает с первой записью)

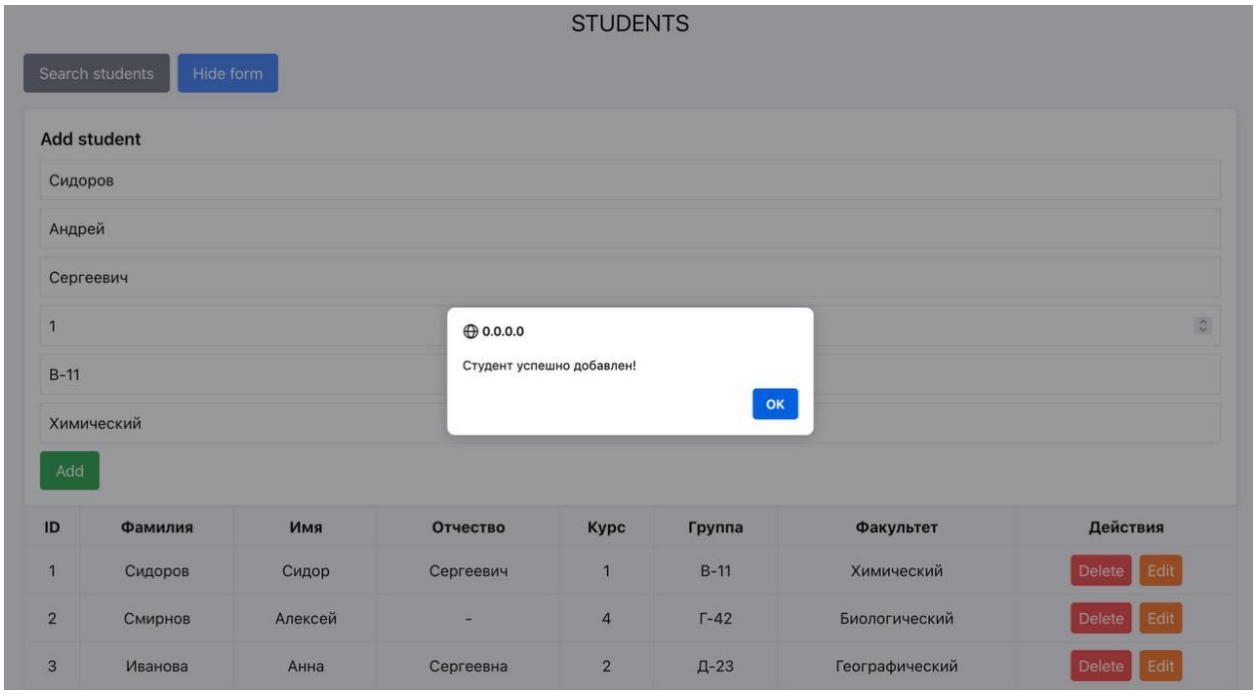


Рисунок 28 – Успешная попытка добавить студента (другое имя)

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены основы создания простых одностраничных веб-приложений с использованием фреймворка FastAPI и библиотеки React. Были освоены механизмы обработки HTTP-запросов, работа с базами данных через SQLAlchemy ORM, а также создание страниц с пагинацией.

СПИСОК ИСТОЧНИКОВ

1. Документация FastAPI [Электронный ресурс]. – URL: https://fastapi.tiangolo.com/ (Дата обращения: 10.02.2025).
2. Документация React [Электронный ресурс]. – URL: https://react.dev/ (Дата обращения: 10.02.2025).
3. Документация Tailwind [Электронный ресурс]. – URL: https://tailwindcss.com/docs/installation/using-vite (Дата обращения: 10.02.2025).
4. Примеры и шаблоны с GitHub [Электронный ресурс]. – URL: https://github.com/cs-itmo/webdev_2024/tree/master/lessons (Дата обращения: 10.02.2025).

ПРИЛОЖЕНИЕ А

Листинг 4 – Программный код main.py (сервис api)

```
from fastapi import FastAPI, Depends, HTTPException, status, Request, Query
from fastapi.security import OAuth2PasswordRequestForm
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from sqlalchemy.orm import Session, joinedload
import logging
from fastapi.middleware.cors import CORSMiddleware

from database import engine, Base, get_db
from models import Student
from typing import List, Optional
from schemas import StudentResponse, StudentCreateOrUpdate, StudentsResponse

# create app
app = FastAPI()

# create database tables
Base.metadata.create_all(bind=engine)

# create logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# ==================================== STUDENT LOGIC
====================================

@app.get("/students", response_model=StudentsResponse)
def read_students(
    db: Session = Depends(get_db),
    skip: int = Query(0, description="Number of records to skip"),
    limit: int = Query(None, description="Maximum number of records to
return")):

    students =
db.query(Student).order_by(Student.id).offset(skip).limit(limit).all()
    total_count = db.query(Student).count()

    return {"students": students, "totalCount": total_count}

@app.post("/student/create")
def create_student(student: StudentCreateOrUpdate, db: Session =
Depends(get_db)):
    middle_name = student.middle_name if student.middle_name else None  #
None or str
    logger.info(f"middle_name: {middle_name}")
    # check if student already exists
    db_student = db.query(Student).filter(
        Student.last_name == student.last_name,
        Student.first_name == student.first_name,
        Student.middle_name == middle_name,
```

```
        Student.course == student.course,
        Student.group_name == student.group_name,
        Student.faculty == student.faculty
    ).first()

    if db_student:
        raise HTTPException(status_code=400, detail="Student already exists")

    new_student = Student(**student.dict())
    db.add(new_student)
    db.commit()
    db.refresh(new_student)
    return {"message": "Student added successfully", "student": new_student}

@app.delete("/student/delete/{student_id}")
def delete_chat_room(student_id: int, db: Session = Depends(get_db)):
    student = db.query(Student).filter(Student.id == student_id).first()
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")

    # delete student from database
    db.delete(student)
    db.commit()
    return {"message": f"Student (ID: {student_id}) deleted"}

@app.patch("/student/update/{student_id}")
def update_student(student_id: int, student_update: StudentCreateOrUpdate,
db: Session = Depends(get_db)):
    student = db.query(Student).filter(Student.id == student_id).first()
    if not student:
        raise HTTPException(status_code=404, detail="Student not found")

    # update fields and check if there is any new filed
    changes_flag = False
    for key, value in student_update.dict(exclude_unset=True).items():
        if getattr(student, key) != value:
            changes_flag = True
            setattr(student, key, value)

    if not changes_flag:
        raise HTTPException(status_code=404, detail="Student hasn't changed")

    db.commit()
    db.refresh(student)
    return {"message": f"Student (ID: {student_id}) updated"}


@app.get("/students/search", response_model=StudentsResponse)
def search_students(
    request: Request,
    db: Session = Depends(get_db)
    ):

    filters = []
    logger.info(f"parameters: {request.query_params.items()}")
    for field, value in request.query_params.items():
        if field in ["skip", "limit"]:
            continue
        if field == "middle_name":
            if value == "null":
                filters.append(Student.middle_name.is_(None))
            else:
                filters.append(getattr(Student, field).ilike(f"%{value}%"))
```

```
        elif hasattr(Student, field):
            column = getattr(Student, field)
            if column.type.python_type == int:
                try:
                    int_value = int(value)
                    filters.append(column == int_value)
                except ValueError:
                    raise HTTPException(status_code=400, detail=f"Invalid
value for field {field}: {value}. Must be an integer.")
            else:
                filters.append(column.ilike(f"%{value}%"))
        else:
            raise HTTPException(status_code=400, detail=f"Invalid search
field: {field}")

    totalCount = db.query(Student).filter(*filters).count() if filters else
db.query(Student).count()
    skip = int(request.query_params.get('skip', 0))
    limit = int(request.query_params.get('limit', totalCount))


    if filters:
        # students = db.query(Student).filter(*filters).all()
        students =
db.query(Student).order_by(Student.id).filter(*filters).offset(skip).limit(li
mit).all()
    else:
        # students = []
        students =
db.query(Student).order_by(Student.id).offset(skip).limit(limit).all()

    if not students:
        raise HTTPException(status_code=404, detail="No students found")

    # return students
    return {"students": students, "totalCount": totalCount}
```

ПРИЛОЖЕНИЕ Б

Листинг 6 – Программный код Home.js (сервис frontend)

```
import React, { useState, useEffect } from "react";
import StudentTable from "../components/StudentTable";
import Pagination from "../components/Pagination";
import StudentForm from "../components/StudentForm";
import StudentEditForm from "../components/StudentEditForm";

const Home = () => {
  //   const apiUrl = process.env.REACT_APP_API_URL || "";
  //   console.log("API URL:", process.env.REACT_APP_API_URL);
  const [students, setStudents] = useState([]);
  const [page, setPage] = useState(1);
  const [pageSize, setPageSize] = useState(5);
  const [totalPages, setTotalPages] = useState(1);
  const [isFormVisible, setIsFormVisible] = useState(false);

  const [editingStudent, setEditingStudent] = useState(null);
  const [isEditFormVisible, setIsEditFormVisible] = useState(false);
  const [isSearchVisible, setIsSearchVisible] = useState(false);

  {
    /* ================================== SEARCH FILTERS
================================== */
  }
  const [filters, setFilters] = useState({
    last_name: "",
    first_name: "",
    middle_name: "",
    course: "",
    group_name: "",
    faculty: "",
  });

  {
    /* ================================== RESET FILTERS
================================== */
  }
  const handleResetFilters = () => {
    setFilters({
      last_name: "",
      first_name: "",
      middle_name: "",
      course: "",
      group_name: "",
      faculty: "",
    });

    // setPage(1);
    // fetchStudents();
  };

  {
    /* ================================== STUDENTS SEARCH
================================== */
  }
  const handleSearch = (e) => {
    e.preventDefault();
    setPage(1);
    fetchStudents();
  };
```

```
  {
    /* ==================================== STUDENTS LOAD
==================================== */
  }
  const fetchStudents = () => {
    const skip = (page - 1) * pageSize;
    const cleanedFilters = Object.fromEntries(
      Object.entries(filters).filter(([_, value]) => value !== "")
    );
    const queryParams = new URLSearchParams({
      skip,
      limit: pageSize,
      ...cleanedFilters,
    });

    fetch(`/api/students/search?${queryParams}`)
      // .then((res) => res.json())
      .then((res) => {
        if (!res.ok) {
          return res.json().then((errorData) => {
            throw new Error(errorData.detail || "Error while data loading");
          });
        }
        return res.json();
      })
      .then((data) => {
        console.log("Received data: ", data);
        setStudents(data.students);
        setTotalPages(Math.ceil(data.totalCount / pageSize));
      })
      .catch((error) => {
        console.error("Error during data loading: ", error);
        alert(error.message);
        setStudents([]);
        setTotalPages(1);
      });
  };

  useEffect(() => {
    fetchStudents();
  }, [page, pageSize]);

  {
    /* ==================================== STUDENTS DELETE
==================================== */
  }
  const handleDelete = (studentId) => {
    if (window.confirm("Are you sure you want to remove this student?")) {
      fetch(`/api/student/delete/${studentId}`, {
        method: "DELETE",
      })
        .then((res) => {
          if (res.ok) {
            setStudents((prevStudents) =>
              prevStudents.filter((student) => student.id !== studentId)
            );
            alert("The student has been successfully removed!");
            window.location.reload();
          } else {
            alert("Error while deleting a student");
          }
        })
```

```
        .catch((error) => console.error("Error while deleting: ", error));
    }
  };

  {
    /* ==================================== STUDENTS UPDATE
==================================== */
  }
  const handleEdit = (student) => {
    setEditingStudent(student);
    setIsEditFormVisible(true);
  };

  const handleUpdate = async (updatedStudent) => {
    try {
      const response = await
fetch(`/api/student/update/${updatedStudent.id}`, {
        method: "PATCH",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(updatedStudent),
      });

      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.detail || "Error while updating a
student");
      }

      alert("The student has been successfully updated!");
      setStudents(
        students.map((student) =>
          student.id === updatedStudent.id ? updatedStudent : student
        )
      );
      setIsEditFormVisible(false);
    } catch (error) {
      alert(error.message);
    }
  };

  return (
    <div className="container mx-auto p-4">
      <div className="flex justify-center">
        <h1 className="text-2xl mb-4">STUDENTS</h1>
      </div>

      {/* ==================================== STUDENTS SEARCH BUTTON
==================================== */}
      <button
        onClick={() => setIsSearchVisible(!isSearchVisible)}
        className="px-4 py-2 bg-gray-500 text-white rounded hover:bg-gray-600
mb-4 mr-2"
      >
        {isSearchVisible ? "Hide search" : "Search students"}
      </button>

      {/* ============================== STUDENT ADD BUTTON
============================== */}
      <button
        onClick={() => setIsFormVisible(!isFormVisible)}
```

```
        className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600
mb-4"
      >
        {isFormVisible ? "Hide form" : "Add student"}
      </button>

      {isFormVisible && (
        <StudentForm
          onStudentAdded={(newStudent) =>
            setStudents([...students, newStudent])
          }
        />
      )}

      {/* ==================================== STUDENTS SEARCH BLOCK
==================================== */}
      {isSearchVisible && (
        <form onSubmit={handleSearch} className="mb-4 bg-gray-100 p-4
rounded">
          <div className="grid grid-cols-2 gap-4">
            {[
              "last_name",
              "first_name",
              "middle_name",
              "course",
              "group_name",
              "faculty",
            ].map((field) => (
              <input
                key={field}
                type={field === "course" ? "number" : "text"}
                name={field}
                placeholder={field.replace("_", " ")}
                value={filters[field]}
                onChange={(e) =>
                  setFilters((prev) => ({ ...prev, [field]: e.target.value
}))
                }
                className="p-2 border rounded w-full"
              />
            ))}
          </div>

          <div className="flex">
            {/* ============================ SEARCH BUTTON
============================ */}
            <button
              type="submit"
              className="mt-4 px-4 py-2 bg-blue-500 text-white rounded
hover:bg-blue-600"
            >
              Search
            </button>
            {/* ============================ RESTORE FILTERS BUTTON
============================ */}
            <button
              type="button"
              onClick={handleResetFilters}
              className="mt-4 px-4 py-2 bg-red-500 text-white rounded
hover:bg-red-600 ml-2"
            >
              Reset filters
            </button>
```

```
          </div>
        </form>
      )}

      {/* ==================================== STUDENT UPDATE BLOCK
==================================== */}
      {isEditFormVisible && editingStudent && (
        <StudentEditForm
          student={editingStudent}
          onUpdate={handleUpdate}
          onCancel={() => setIsEditFormVisible(false)}
        />
      )}

      {/* ==================================== STUDENTS VIEW BLOCK
==================================== */}
      {students.length === 0 ? (
        <p className="text-center text-gray-500">No students found</p>
      ) : (
        <StudentTable
          students={students}
          onDelete={handleDelete}
          onEdit={handleEdit}
        />
      )}

      {/* ==================================== PAGINATION BLOCK
==================================== */}
      <Pagination
        page={page}
        setPage={setPage}
        totalPages={totalPages}
        pageSize={pageSize}
        setPageSize={setPageSize}
      />
    </div>
  );
};

export default Home;
```

ПРИЛОЖЕНИЕ В

Листинг 7 – Настройка tailwind в файле tailwind.config.js (микросервис frontend)

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [],
};
```