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

**Факультет безопасности информационных технологий**

**Дисциплина:**

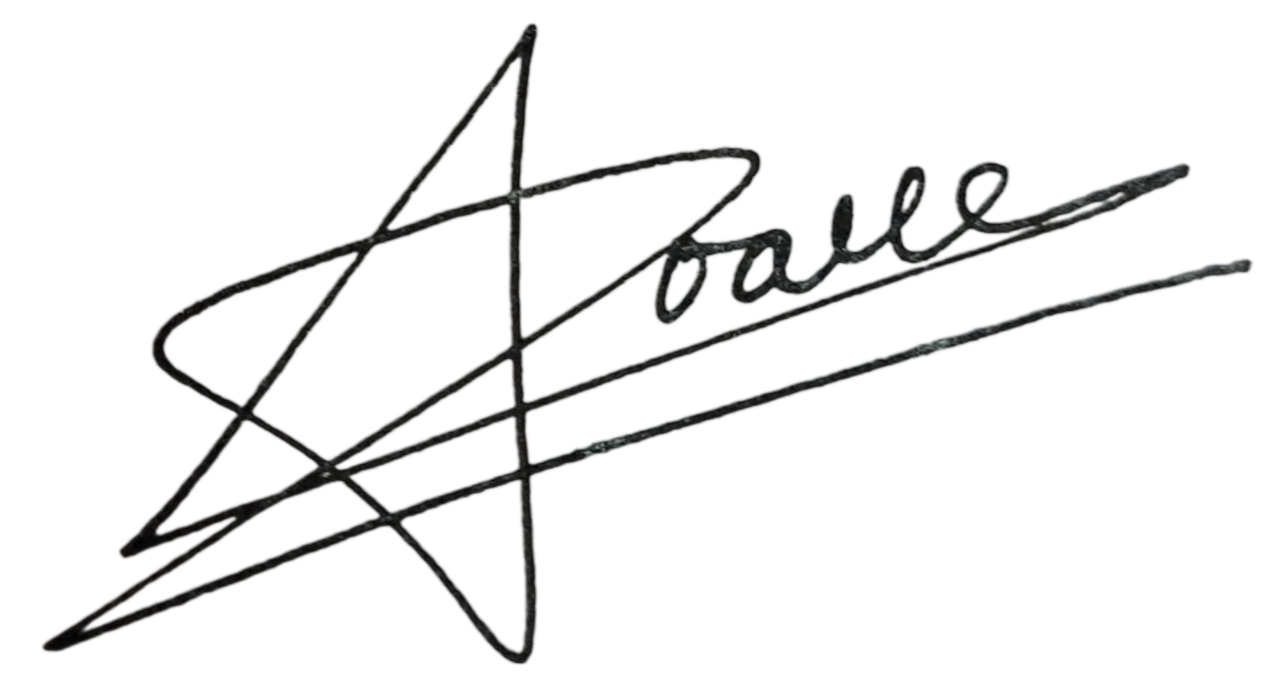
**«Web программирование»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

«Разработка панели администратора»

**Выполнили:**

Чу Ван Доан, студент группы N3347



*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Менщиков Александр Алексеевич

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

# Содержание

[**Содержание 2**](#_heading=h.85nwmm3u2k3p)

[**Введение 4**](#_heading=h.gjdgxs)

[1. Задание 4](#_heading=h.puhy4onlavyt)

[1.1. Сценарий использования 5](#_heading=h.9xmoaavptwf)

[1.2. Техническое задание 5](#_heading=h.fobopicy8jys)

[1.3. Требования в сервису API 5](#_heading=h.lrx0lg4mh06e)

[1.4. Требования к сервису frontend 6](#_heading=h.og8diflz84ww)

[1.5. Требования к базе данных 6](#_heading=h.u88y6l8hhlzf)

[1.6. Дополнительные баллы (по желанию) 6](#_heading=h.nvo06rv8sbbo)

[1.7. Общие требования 6](#_heading=h.o7affqi05o2y)

[1.8. Требования к отчету 7](#_heading=h.vo6co3jcjjyn)

[2. Ход работы 8](#_heading=h.di7u4ulvin57)

[2.1. Описание архитектуры ПО 8](#_heading=h.ulbutra9hjmj)

[2.1.1. Общая структура 8](#_heading=h.uatzeoqjcde3)

[2.1.2. Компоненты системы 9](#_heading=h.31jxi8to4dlz)

[2.1.3. Взаимодействие компонентов 10](#_heading=h.vkzjsutxmjl)

[2.1.4. Особенности и расширяемость 10](#_heading=h.vxdvazfhdimh)

[2.2. Описание структуры базы данных 10](#_heading=h.dtvr894ullp)

[2.3. Описание протокола и форматов передачи данных; 11](#_heading=h.oodvtnt24zs2)

[2.3.1. Протокол передачи данных 11](#_heading=h.f4fwj6v5tu8p)

[2.3.2. Форматы передачи данных 12](#_heading=h.5cxoopr6nf7h)

[2.3.3. Основные маршруты и параметры 12](#_heading=h.f4fwj6v5tu8p)

[2.3.4. Middleware для обработки CORS 12](#_heading=h.f4fwj6v5tu8p)

[2.3.5. Логика обработки данных 13](#_heading=h.f4fwj6v5tu8p)

[2.3.6. Преимущества подхода 13](#_heading=h.f4fwj6v5tu8p)

[2.4. Описание API 13](#_heading=h.8v72j8x9bjzq)

[2.4.1. Получение списка студентов 13](#_heading=h.2lkal12rpxlt)

[2.4.2. Общие аспекты API 14](#_heading=h.2lkal12rpxlt)

[2.4.3. Возможности расширения API 15](#_heading=h.2lkal12rpxlt)

[2.5. Скриншоты функционала 16](#_heading=h.pwm54tmys8di)

[2.6. Исходный код 24](#_heading=h.smtmh52b72ur)

[2.6.1. Код для необходимых файлов 24](#_heading=h.x5fiwcbpkf5j)

[2.6.2. API 26](#_heading=h.y7cvip7hbnoj)

[2.6.3. Frontend 29](#_heading=h.ecnt8cu5xbgh)

[**Заключение 39**](#_heading=h.tyjcwt)

# Введение

Цель работы - Разработать приложение для администрирования базы данных студентов, включающее основные компоненты: сервис API, интерфейс управления и базу данных. Приложение должно обеспечивать эффективное управление данными, поддержку пагинации и предоставлять удобный и интуитивно понятный интерфейс.

## Задание

Необходимо разработать приложение-админку для управления базой данных студентов. Для этого вам нужно связать между собой 3 сервиса.

* Первый – простейший JSON REST API на Python FastAPI, который позволяет управлять записями.
* Второй – собранный статический сайт на React.
* Третий – база данных, с которой взаимодействует API.

Хранение данных должно быть реализовано в PostgreSQL либо в MongoDB. Аутентификацию делать не требуется. Сервисы обязательно должны иметь поддержку пагинации.

### Сценарий использования

Пользователь заходит на главную страницу сайта и видит таблицу с текущими данными студентов. Внизу он видит переключатель страниц (пагинация с изменением размера страницы). API должен поддерживать эту пагинацию и выдавать ровно столько данных, сколько запрашивается с фронтенда.

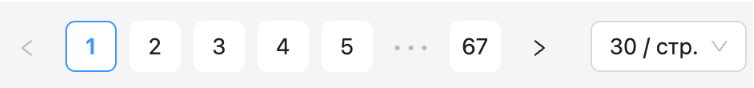


Рисунок 1 - Пример виджета пагинации

В базовом варианте достаточно реализовать только просмотр данных. Если вы хотите усложнений (либо добрать доп. баллы), см. раздел «Дополнительные баллы».

### Техническое задание

* Микросервис API – функционал REST JSON API;
* Микросервис frontend – веб-сервер (например, на базе Nginx), который выдает собранные файлы JS,CSS,HTML на основе React;
* База данных.

### Требования в сервису API

* Должен быть разработан на языке Python с использованием фреймворка FastAPI; По согласованию можете выбрать любую другую технологию.
* Выдача данных только в JSON;
* Поддержка пагинации с произвольным размером страницы;
* Хранение и извлечение данных должно быть в БД PostgreSQL либо MongoDB по выбору;
* Доступ к БД только с использованием ORM библиотек;
* При выполнении усложненных версий лабораторной работы необходимо использовать REST подход и правильные HTTP методы: GET, POST, DELETE, PATCH;
* Данные в БД можете заполнить тестовыми сведениями (~20 записей).

### Требования к сервису frontend

* Должен быть разработан на языке JavaScript;
* Должен использоваться React;
* Должен запрашивать данные с использованием REST API;
* Необходимо реализовать переключатель страниц (пагинация с изменением размера страницы);
* Итоговый бандл должен быть собран в виде статических файлов. Раздавать их можно с помощью любого веб-сервера (Например, <https://www.npmjs.com/package/http-server> или Nginx).

### Требования к базе данных

* Должна быть выбрана БД PostgreSQL или MongoDB;
* 1 таблица со следующими полями: Фамилия, Имя, Отчество, Курс, Группа, Факультет.

### Дополнительные баллы (по желанию)

* Добавить функцию создания новых записей (POST);
* Добавить удаление записи (DELETE);
* Добавить редактирование любого поля (обновление записи, PATCH);
* Добавить поиск и фильтрацию по полям;
* Добавить docker-compose.yml и завернуть все микросервисы в Docker с использованием Dockerfile.

### Общие требования

* Весь проект должен запускаться одной командой. Это может быть bash скрипт, docker-compose скрипт или NPM команда.
* Должна поддерживаться следующая структура директории:
* В корне расположен README.md (и docker-compose.yml если вы выбрали усложнение);
* Каждый микросервис, его код, package.json и прочие файлы расположены в своих директориях (api, frontend, db).
* Наружу должны быть доступны только 2 HTTP порта – 80 для frontend и 3000 для API;
* База данных должна обеспечивать persistence в локальную директорию dump;
* Должны быть настроены файлы .gitignore и .dockerignore.

### Требования к отчету

* Отчет в формате Docx в соответствии с приложенным шаблоном;
* Заполнение каждого из следующих разделов:
* Описание архитектуры ПО;
* Описание структуры базы данных;
* Описание протокола и форматов передачи данных;
* Описание API;
* Скриншоты функционала (каждая страница и кейс из технического задания).
* Приложите ключевые файлы с кодом (api, frontend) в главы приложений отчета;
* Приложите код в виде ZIP архива к отчету.
* Каждый файл должен быть не более 5 Мб.

## Ход работы

### Описание архитектуры ПО

project/

├── .gitignore

├── docker-compose.yml

├── package-lock.json

├── package.json

├── frontend/

│ ├── Dockerfile

│ ├── package.json

│ ├── public/

│ │ └── index.html

│ ├── src/

│ │ ├── App.js

│ │ ├── FilterForm.js

│ │ ├── Pagination.js

│ │ ├── index.js

│ │ ├── services/

│ │ │ └── api.js

│ │ └── styles/

│ │ └── main.css

├── api/

│ ├── Dockerfile

│ ├── requirements.txt

│ ├── app/

│ ├── \_\_init\_\_.py

│ ├── crud.py

│ ├── database.py

│ ├── main.py

│ ├── models.py

│ └── schemas.py

#### Общая структура

Архитектура ПО представляет собой распределенную систему, включающую два основных компонента:

* Frontend: пользовательский интерфейс, реализованный на основе React.
* Backend: серверная часть API, реализованная с использованием Python и FastAPI.
* Инфраструктура: поддерживается Docker и orchestration с помощью Docker Compose.

#### Компоненты системы

**a) Frontend**

* Расположен в папке frontend.
* Основной стек технологий:
* React: библиотека для создания UI.
* CSS: используется для стилизации через файл styles/main.css.
* API-запросы обрабатываются через модуль services/api.js.
* Основные модули:
* index.js: точка входа приложения.
* App.js: главная страница, управляющая рендерингом.
* Pagination.js и FilterForm.js: компоненты для фильтрации и пагинации данных.
* Dockerfile: настройка контейнера для запуска React-приложения.

**b) Backend**

* Расположен в папке api.
* Основной стек технологий:
* FastAPI: фреймворк для разработки REST API.
* SQLAlchemy: ORM для взаимодействия с базой данных.
* Основные файлы:
* main.py: точка входа backend-приложения, регистрация маршрутов.
* database.py: подключение к базе данных.
* models.py: описания ORM-моделей.
* schemas.py: валидация входных/выходных данных через Pydantic.
* crud.py: операции Create, Read, Update, Delete.
* Dockerfile: настройка контейнера для API.

**c) Инфраструктура**

* docker-compose.yml: orchestration компонентов (Frontend, Backend).
* База данных конфигурируется в Docker Compose, но её подробности не указаны в файлах.

#### Взаимодействие компонентов

* Frontend отправляет запросы на API Backend через определенные маршруты (например, /api/...).
* Backend обрабатывает запросы, взаимодействует с базой данных, возвращает данные или ошибки.
* Docker Compose обеспечивает запуск всей системы в изолированных контейнерах.

#### Особенности и расширяемость

* Использование FastAPI и React обеспечивает высокую производительность и простоту масштабирования.
* Чёткое разделение компонентов позволяет легко изменять или заменять любую часть системы.

### Описание структуры базы данных

База данных состоит из одной основной таблицы students, описанной в файле models.py. Таблица содержит информацию о студентах, включая их личные данные, академические характеристики и принадлежность к факультету.

Таблица: students

* id (Primary Key, Integer):
* Уникальный идентификатор студента.
* Используется как первичный ключ.
* Фамилия (String):
* Фамилия студента.
* Индексируемое поле для ускорения поиска.
* Имя (String):
* Имя студента.
* Индексируемое поле.
* Отчество (String):
* Отчество студента.
* Индексируемое поле.
* Курс (String):
* Курс, на котором обучается студент.
* Индексируемое поле.
* Группа (String):
* Учебная группа студента.
* Индексируемое поле.
* Факультет (String):
* Название факультета, к которому относится студент.
* Индексируемое поле.
* Код:

CREATE TABLE students (

id SERIAL PRIMARY KEY,

Фамилия VARCHAR(255),

Имя VARCHAR(255),

Отчество VARCHAR(255),

Курс VARCHAR(255),

Группа VARCHAR(255),

Факультет VARCHAR(255)

);

* Особенности:
* Все поля, кроме id, индексируются (index=True) для обеспечения быстрого доступа к данным при выполнении запросов.
* Таблица может быть расширена для добавления дополнительных атрибутов студентов или связей с другими таблицами.
* Использование:
* Таблица предназначена для хранения и управления данными о студентах.
* Реализована с помощью SQLAlchemy, что позволяет удобно взаимодействовать с базой данных через ORM.

### Описание протокола и форматов передачи данных;

#### Протокол передачи данных

* Система использует HTTP/HTTPS как протокол для передачи данных между клиентом (Frontend) и сервером (Backend). API разработан на основе FastAPI, что обеспечивает поддержку стандартов REST и возможности асинхронного взаимодействия.

#### Форматы передачи данных

Для передачи данных между клиентом и сервером используется формат JSON:

* JSON используется как для входных данных (запросов), так и для выходных (ответов).
* Пакет Pydantic используется для сериализации и десериализации данных, что обеспечивает строгую проверку их соответствия схемам.

#### Основные маршруты и параметры

* Маршрут: /students/
* Метод: GET
* Формат запроса:
* URL-параметры:
* skip (int): Смещение для пагинации (по умолчанию 0).
* limit (int): Количество возвращаемых записей (по умолчанию 10).
* Фамилия, Имя, Отчество, Курс, Группа, Факультет (str, опционально): Фильтры для поиска студентов.
* Формат ответа:

[

{

"id": 1,

"Фамилия": "Иванов",

"Имя": "Иван",

"Отчество": "Иванович",

"Курс": "2",

"Группа": "A1",

"Факультет": "Информатика"

},

...

]

#### Middleware для обработки CORS

* Для поддержки междоменных запросов используется CORS Middleware.
* Настройки:
* allow\_origins=["\*"]: Разрешены запросы с любых доменов.
* allow\_methods=["\*"]: Разрешены все методы HTTP.
* allow\_headers=["\*"]: Разрешены все заголовки.

#### Логика обработки данных

* Клиент отправляет запрос на маршрут /students/ с указанными фильтрами и параметрами пагинации.
* Сервер принимает запрос и передаёт его в слой CRUD для выполнения фильтрации:
* Словарь filters заполняется параметрами, переданными через Query.
* Фильтры применяются к запросу к базе данных через SQLAlchemy.
* Сервер возвращает результат в формате JSON.

#### Преимущества подхода

* Использование FastAPI обеспечивает автоматическую генерацию документации API (например, Swagger).
* Формат JSON удобен для передачи структурированных данных и взаимодействия с фронтендом.
* Гибкая система фильтрации позволяет клиенту получать только необходимые данные.

### Описание API

API разработан с использованием FastAPI, предоставляющего RESTful интерфейсы для работы с данными студентов.

#### Получение списка студентов

* Маршрут: /students/
* Метод: GET
* Описание: Возвращает список студентов с возможностью фильтрации и пагинации.
* Параметры запроса:
* skip (int, опционально, по умолчанию: 0): Количество записей для пропуска (начало с skip).
* limit (int, опционально, по умолчанию: 10): Количество записей, которые нужно вернуть.
* Фильтры (все опциональны):
* Фамилия (str): Фильтр по фамилии студента.
* Имя (str): Фильтр по имени студента.
* Отчество (str): Фильтр по отчеству студента.
* Курс (str): Фильтр по курсу студента.
* Группа (str): Фильтр по группе студента.
* Факультет (str): Фильтр по факультету студента.
* Пример запроса:

GET /students/?skip=0&limit=5&Фамилия=Иванов

* Пример ответа:

[

{

"id": 1,

"Фамилия": "Иванов",

"Имя": "Иван",

"Отчество": "Иванович",

"Курс": "2",

"Группа": "A1",

"Факультет": "Информатика"

},

{

"id": 2,

"Фамилия": "Петров",

"Имя": "Петр",

"Отчество": "Петрович",

"Курс": "1",

"Группа": "B2",

"Факультет": "Математика"

}

]

#### Общие аспекты API

* FastAPI:
* Обеспечивает высокую производительность благодаря асинхронности.
* Предоставляет автоматическую генерацию документации API (Swagger и ReDoc), доступную по адресу /docs и /redoc.
* SQLAlchemy:
* ORM для работы с базой данных.
* Обеспечивает гибкую фильтрацию данных с использованием словаря filters.
* Pydantic:
* Используется для определения схем данных (schemas.py).
* Автоматически валидирует входящие запросы и формирует структурированные ответы.

#### Возможности расширения API

* Добавление новых маршрутов:
* Например, маршрут для создания, обновления и удаления студентов.
* Пример POST-запроса:

POST /students/

Content-Type: application/json

{

"Фамилия": "Иванов",

"Имя": "Иван",

"Отчество": "Иванович",

"Курс": "3",

"Группа": "A2",

"Факультет": "Физика"

}

* Аутентификация: Добавить JWT или OAuth2 для ограничения доступа к API.
* Масштабируемость:
* Использовать более сложные фильтры (например, диапазоны значений, сортировку).
* Реализовать пагинацию с указанием общего количества записей.

### Скриншоты функционала



Рисунок 2: Выполнение программы

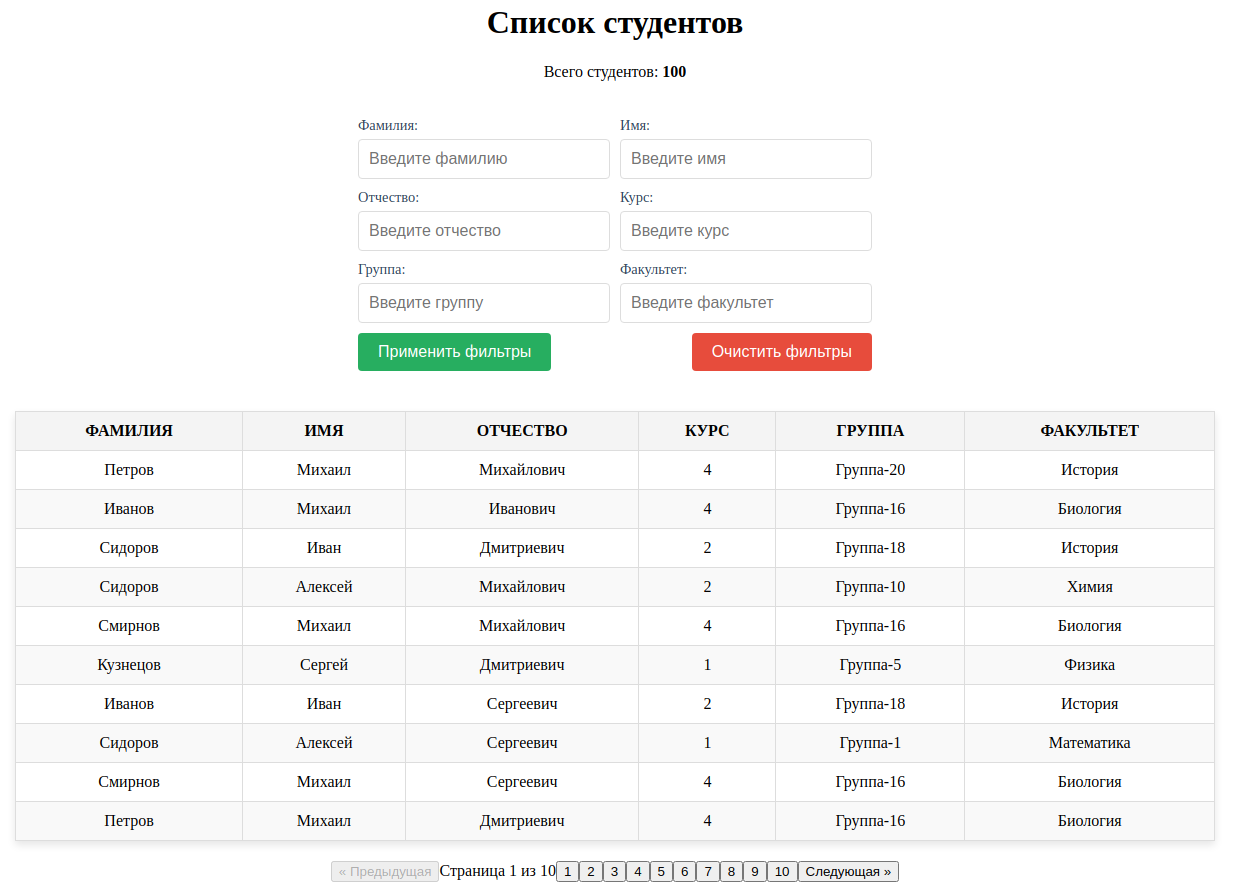


Рисунок 3: Главная страница веб-сайта

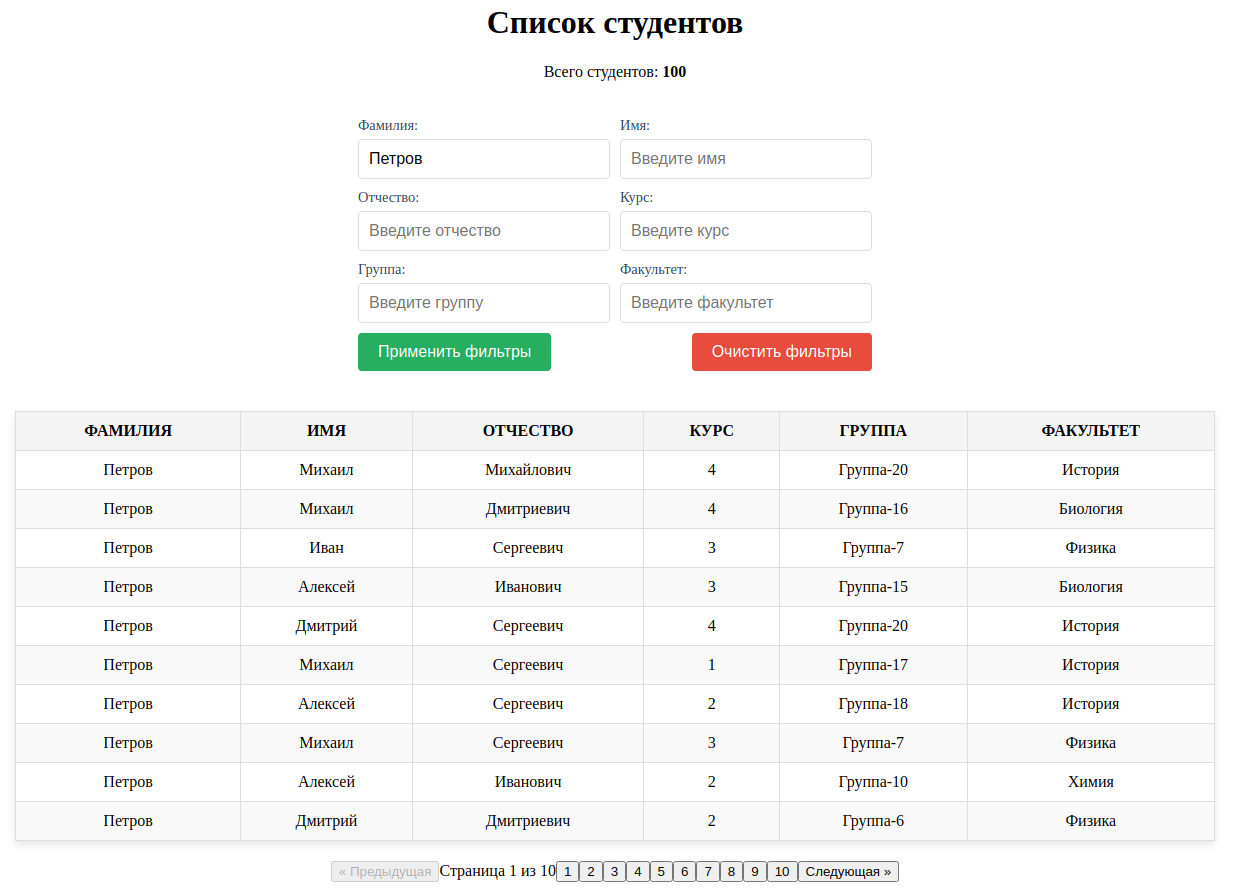


Рисунок 4: Пример фильтрации студентов по Фамилия

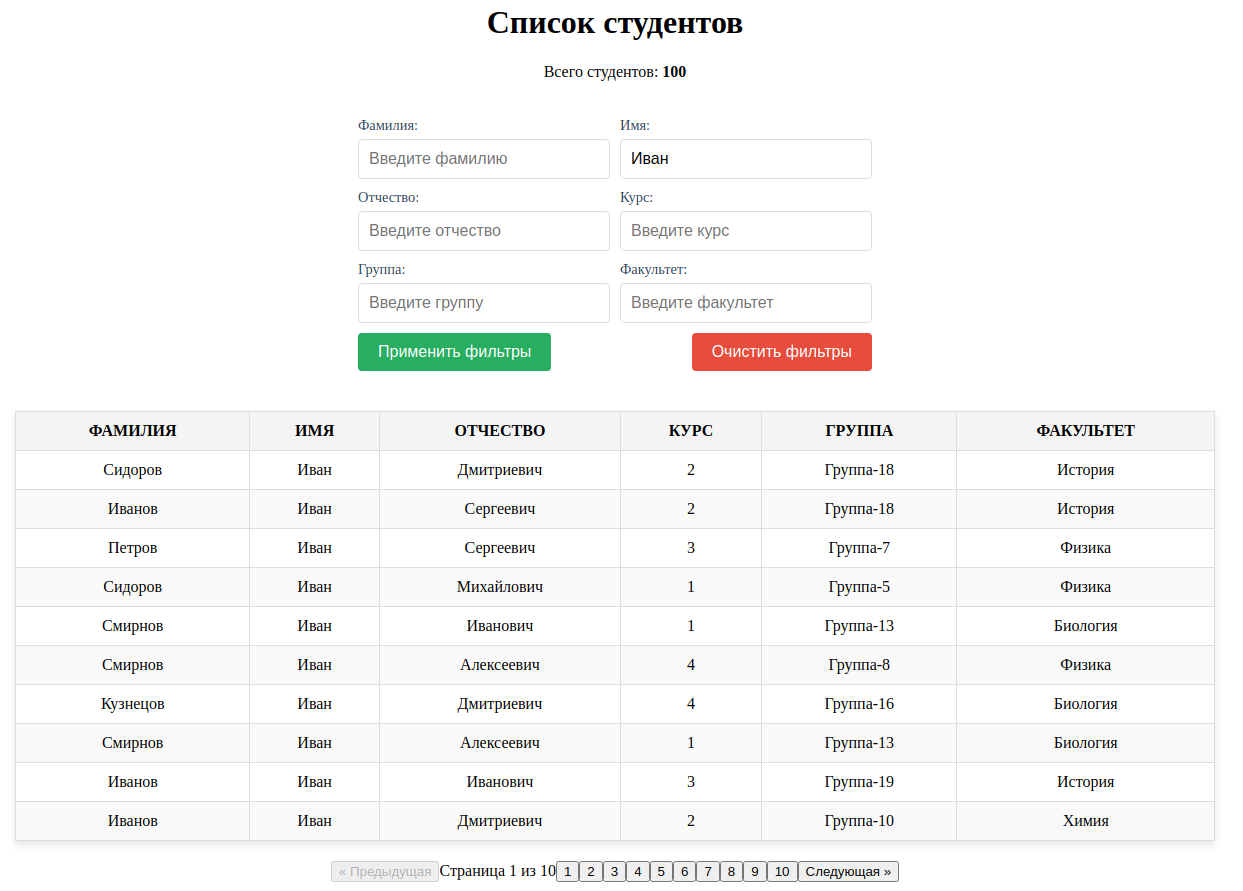


Рисунок 5: Пример фильтрации студентов по Имени

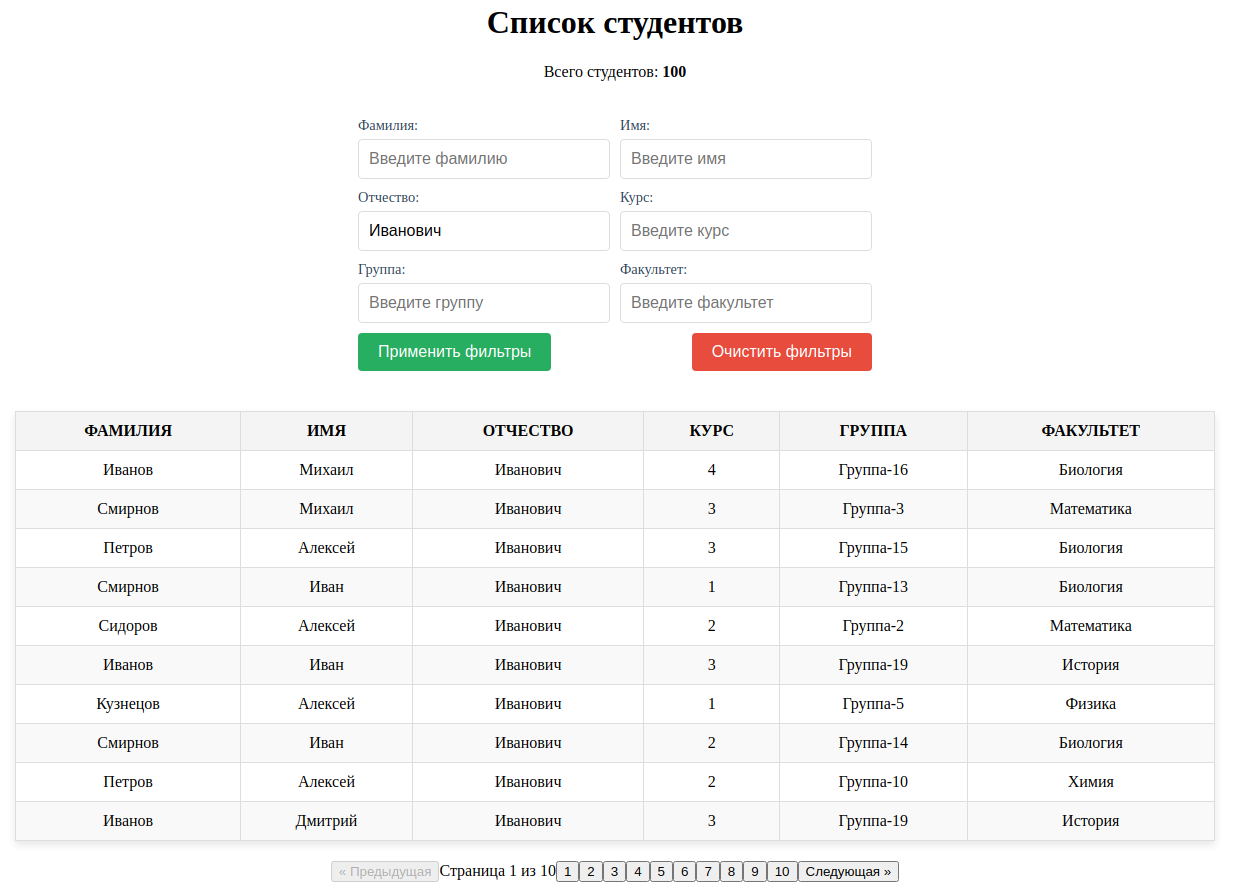


Рисунок 6: Пример фильтрации студентов по Отчеству

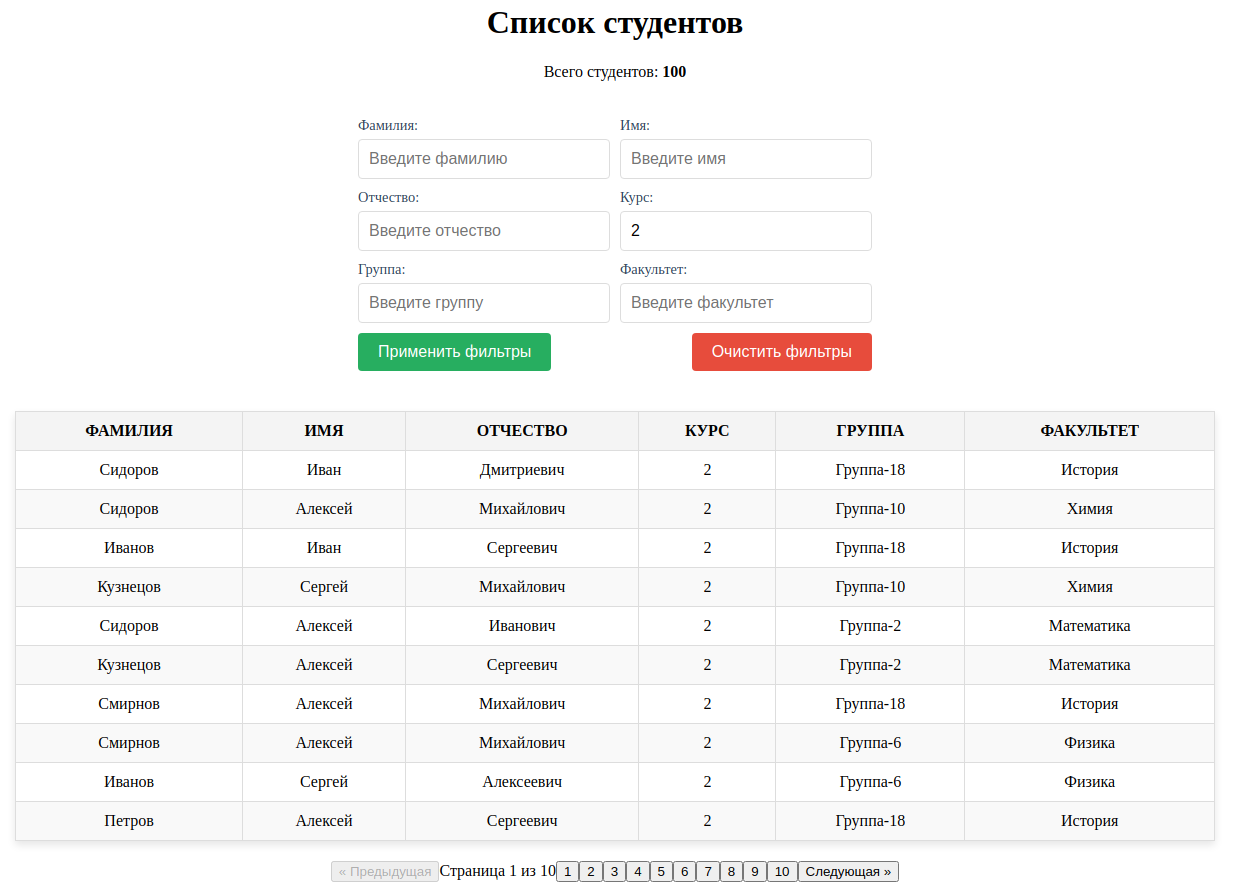


Рисунок 7:Пример фильтрации студентов по Курсу.

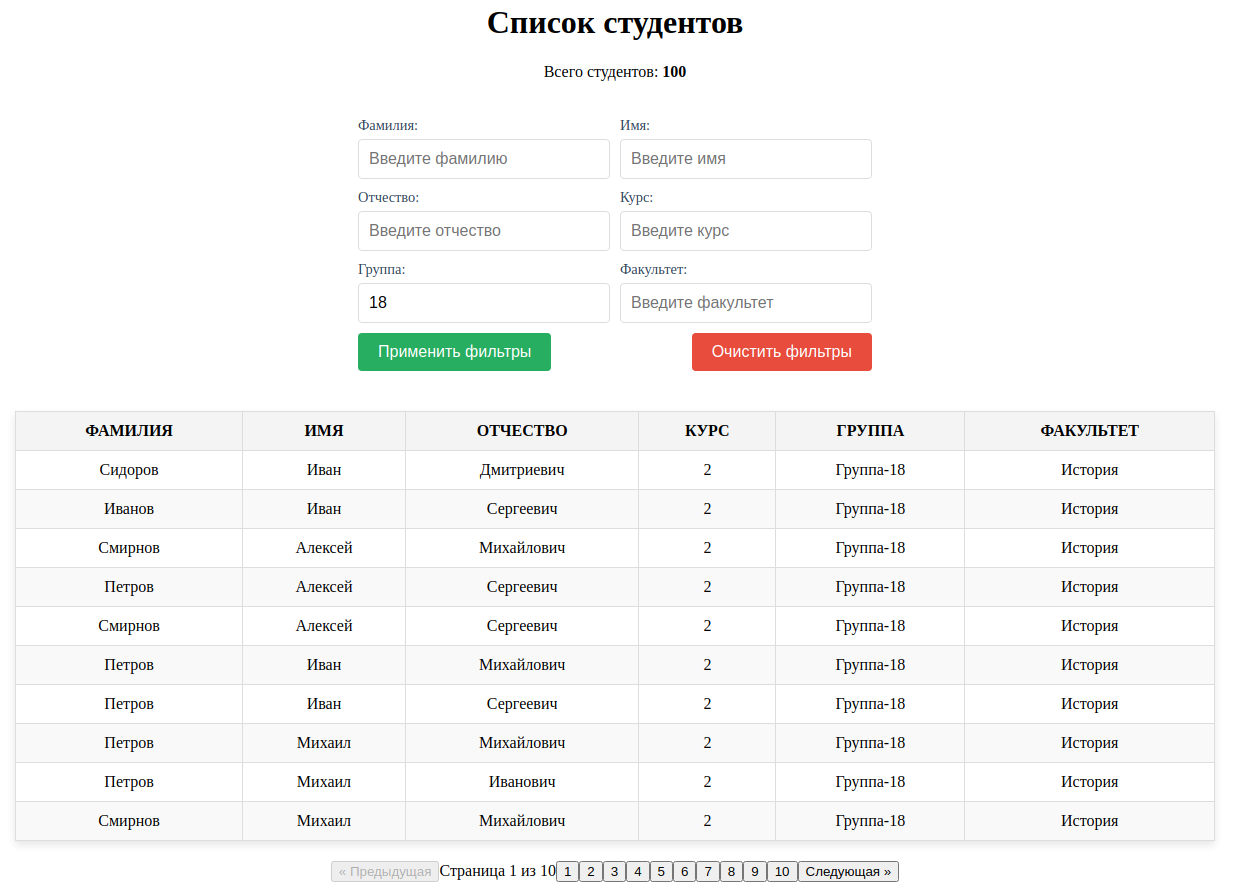


Рисунок 8: Пример фильтрации студентов по Группе.

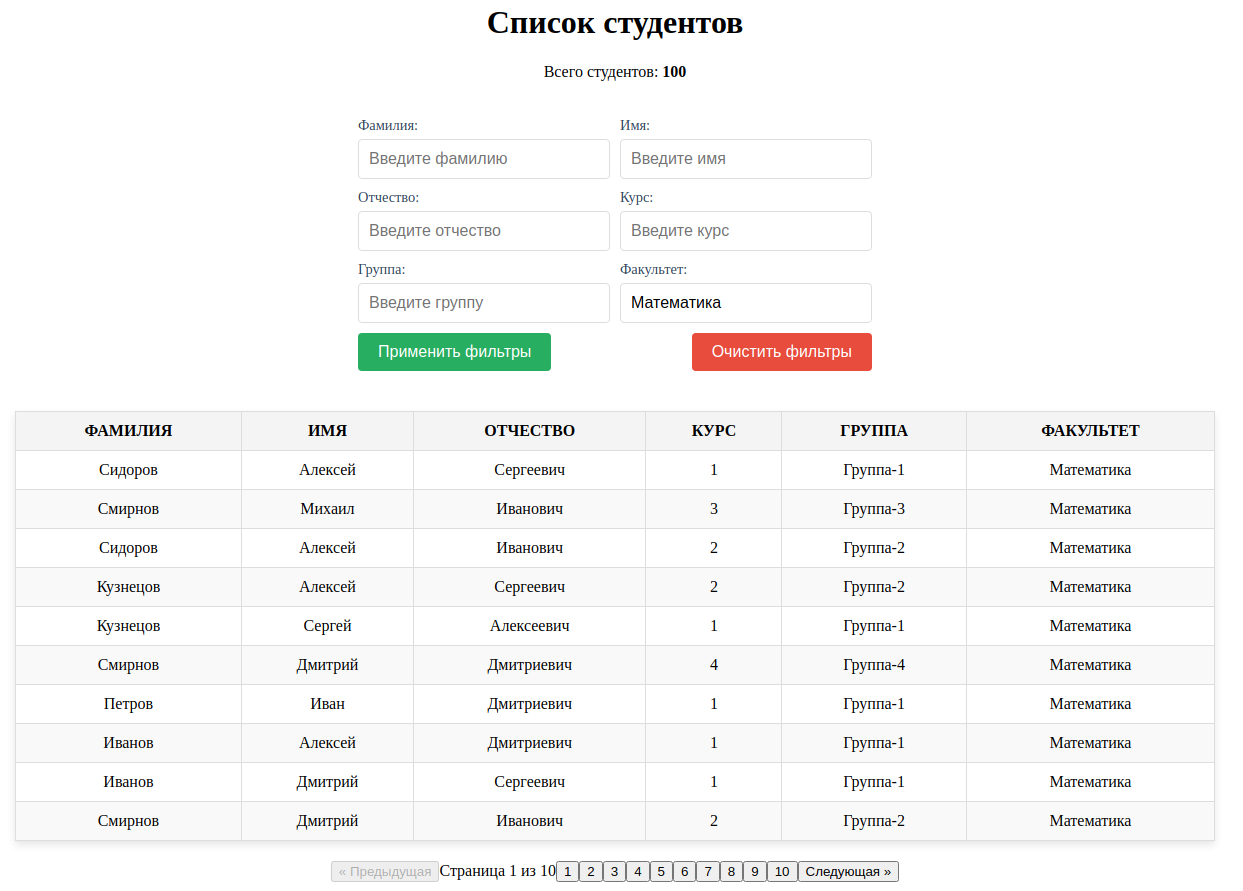


Рисунок 9: Пример фильтрации студентов по Факультету

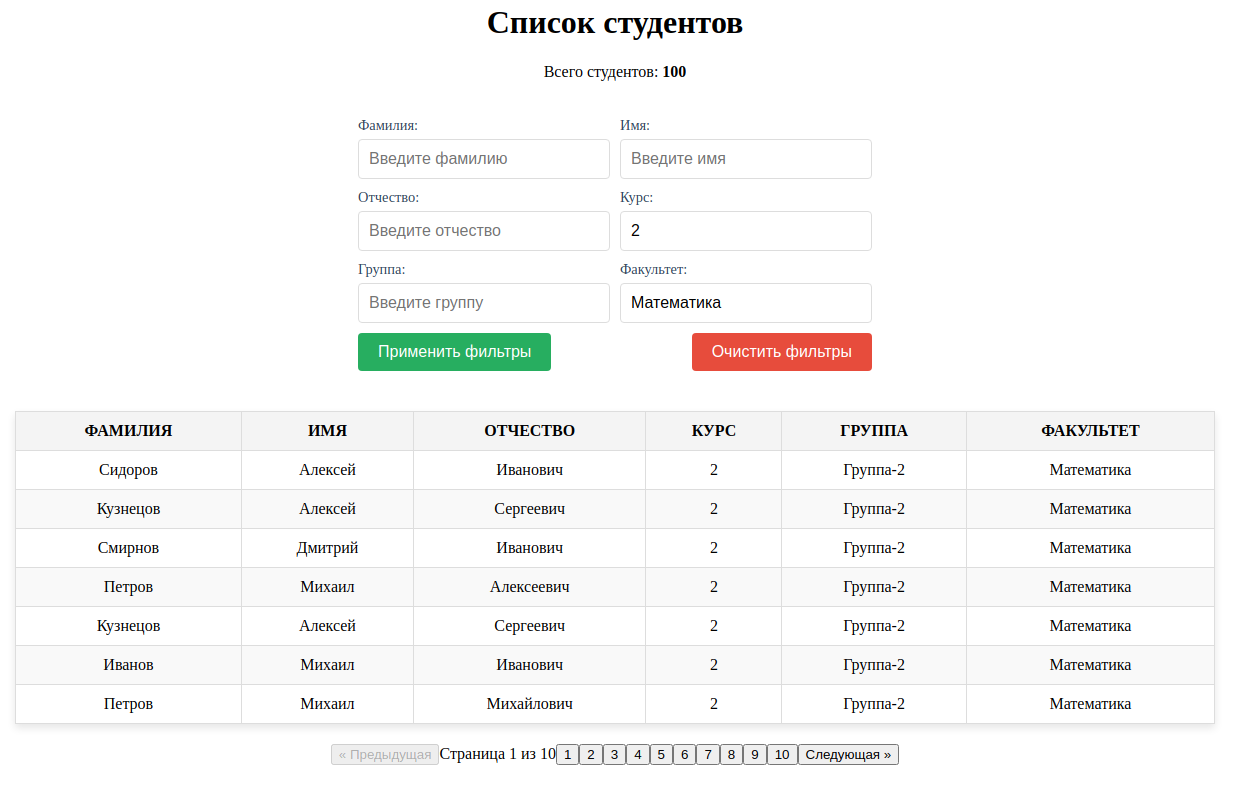


Рисунок 10: Пример фильтрации студентов по Факультету и по Группе

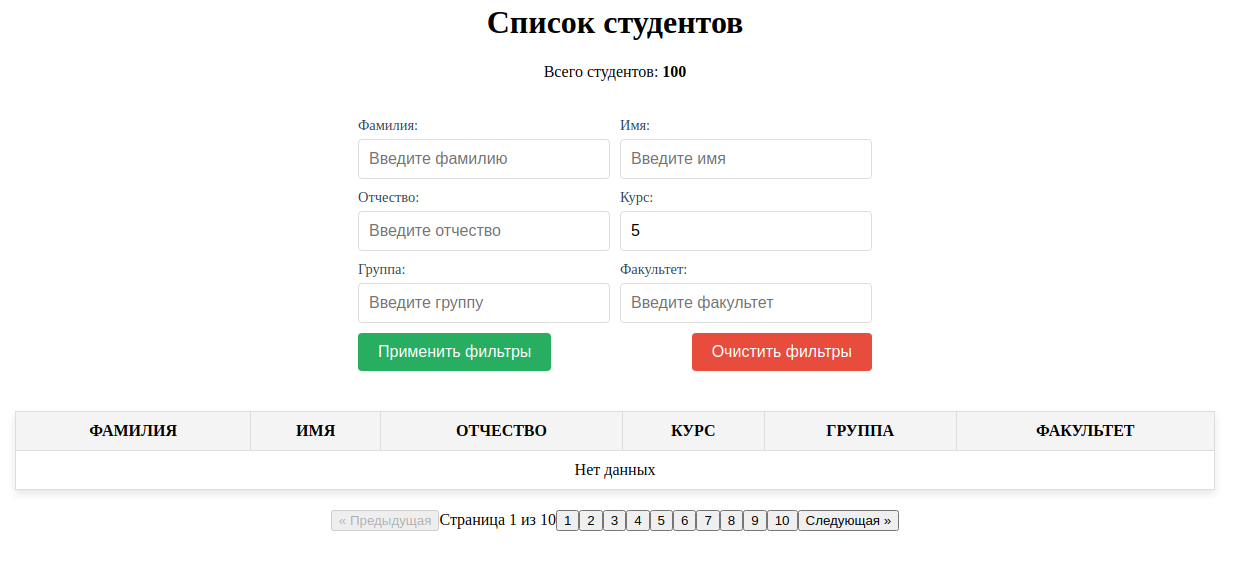


Рисунок 11: Пример фильтрации студентов, отсутствующих в данных

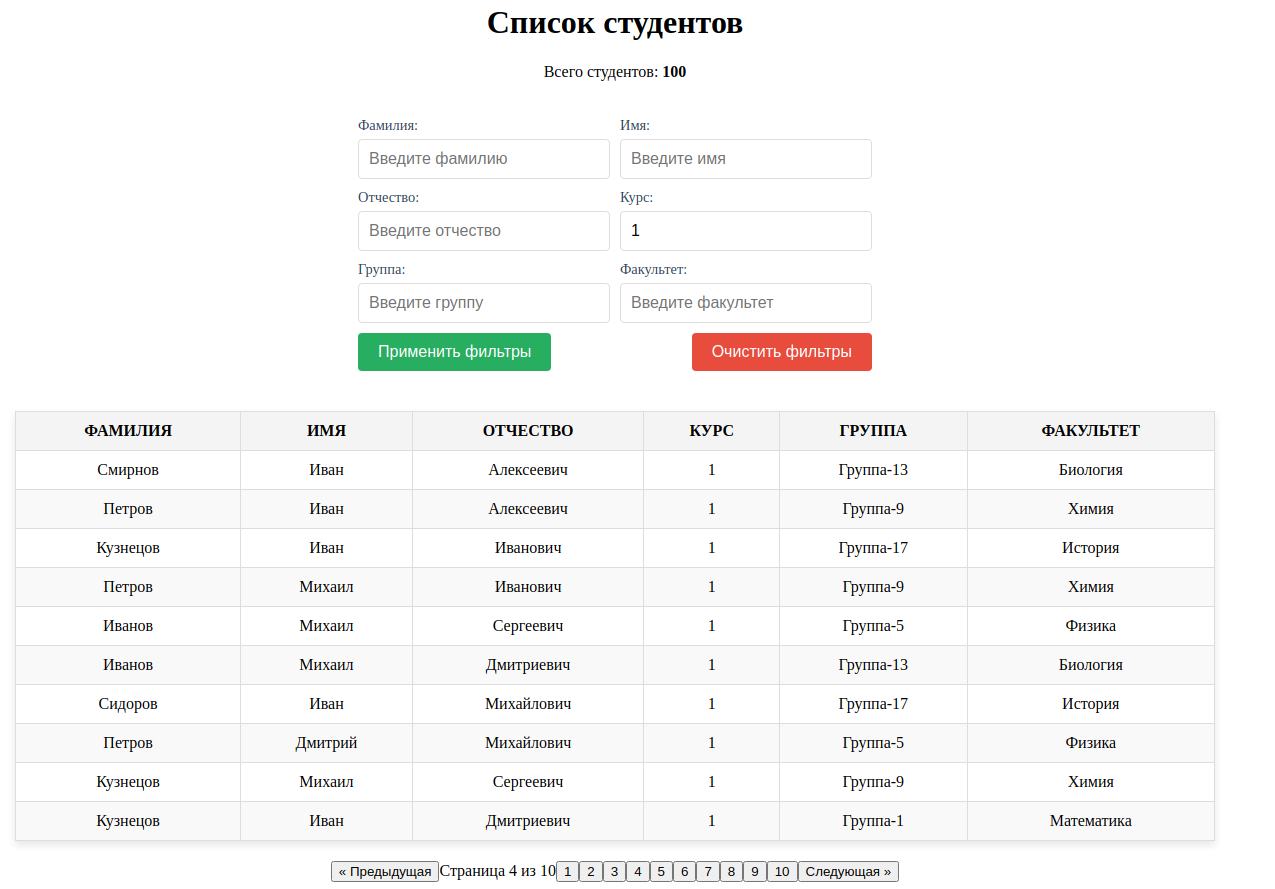


Рисунок 12: Пример перехода между страницами (находясь на странице 4)

### Исходный код

#### Код для необходимых файлов

* .**gitignore**

# Python

\*.pyc

\_\_pycache\_\_/

env/

venv/

# Node.js

node\_modules/

build/

# Docker

\*.log

\*.pid

* **docker-compose.yml**

version: '3.9'

services:

db:

image: postgres:13

container\_name: postgres

environment:

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: chuvandoan

POSTGRES\_DB: student\_db

ports:

- "5433:5432"

volumes:

- db\_data:/var/lib/postgresql/data

api:

build: ./api

container\_name: api

depends\_on:

- db

environment:

DATABASE\_URL: postgresql://postgres:chuvandoan@db:5432/student\_db

ports:

- "3000:8000"

frontend:

build: ./frontend

container\_name: frontend

depends\_on:

- api

ports:

- "80:3000"

volumes:

db\_data:

* **package-lock.json**

{

"name": "project",

"lockfileVersion": 3,

"requires": true,

"packages": {

"": {

"dependencies": {

"process": "^0.11.10"

}

},

"node\_modules/process": {

"version": "0.11.10",

"resolved": "https://registry.npmjs.org/process/-/process-0.11.10.tgz",

"integrity": "sha512-cdGef/drWFoydD1JsMzuFf8100nZl+GT+yacc2bEced5f9Rjk4z+WtFUTBu9PhOi9j/jfmBPu0mMEY4wIdAF8A==",

"engines": {

"node": ">= 0.6.0"

}

}

}

}

* **package.json**

{

"dependencies": {

"process": "^0.11.10"

}

}

#### API

* **api/Dockerfile**

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY app/ ./app/

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

* **api/requirements.txt**

fastapi

uvicorn

sqlalchemy

psycopg2-binary

pydantic

* **api/app/crud.py**

from sqlalchemy.orm import Session

from sqlalchemy import or\_

from . import models

def get\_students(db: Session, skip: int, limit: int, filters: dict):

query = db.query(models.Student)

for field, value in filters.items():

if value:

query = query.filter(getattr(models.Student, field).ilike(f"%{value}%"))

return query.offset(skip).limit(limit).all()

* **api/app/database.py**

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

DATABASE\_URL = "postgresql://postgres:chuvandoan@db:5432/student\_db"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

* **api/app/main.py**

from fastapi import FastAPI, Depends, Query

from sqlalchemy.orm import Session

from . import models, schemas, crud

from .database import SessionLocal, engine

from fastapi.middleware.cors import CORSMiddleware

models.Base.metadata.create\_all(bind=engine)

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

@app.get("/students/", response\_model=list[schemas.Student])

def read\_students(

skip: int = 0,

limit: int = 10,

Фамилия: str = Query(None),

Имя: str = Query(None),

Отчество: str = Query(None),

Курс: str = Query(None),

Группа: str = Query(None),

Факультет: str = Query(None),

db: Session = Depends(get\_db),

):

filters = {

"Фамилия": Фамилия,

"Имя": Имя,

"Отчество": Отчество,

"Курс": Курс,

"Группа": Группа,

"Факультет": Факультет,

}

return crud.get\_students(db, skip=skip, limit=limit, filters=filters)

* **api/app/models.py**

from sqlalchemy import Column, Integer, String

from .database import Base

class Student(Base):

\_\_tablename\_\_ = "students"

id = Column(Integer, primary\_key=True, index=True)

Фамилия = Column(String, index=True)

Имя = Column(String, index=True)

Отчество = Column(String, index=True)

Курс = Column(String, index=True)

Группа = Column(String, index=True)

Факультет = Column(String, index=True)

* **api/app/schemas.py**

from pydantic import BaseModel

class StudentBase(BaseModel):

Фамилия: str

Имя: str

Отчество: str

Курс: str

Группа: str

Факультет: str

class StudentCreate(StudentBase):

pass

class Student(StudentBase):

id: int

class Config:

orm\_mode = True

#### Frontend

* **frontend/Dockerfile**

FROM node:14

WORKDIR /app

COPY package.json .

RUN npm install

COPY public ./public

COPY src ./src

CMD ["npm", "start"]

* **package.json**

{

"name": "frontend",

"version": "1.0.0",

"dependencies": {

"react": "^17.0.2",

"react-dom": "^17.0.2",

"react-scripts": "^4.0.3"

},

"scripts": {

"start": "react-scripts start"

}

}

* **frontend/public/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Student Management</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

* **frontend/src/services/api.js**

export async function fetchStudents(page, filters) {

const queryParams = {

...filters,

skip: (page - 1) \* 10,

limit: 10,

};

const queryString = new URLSearchParams(queryParams).toString();

try {

console.log("Query String:", queryString);

const response = await fetch(`http://localhost:3000/students/?${queryString}`);

console.log("API Response Status:", response.status);

if (!response.ok) {

throw new Error(`API error: ${response.status}`);

}

const data = await response.json();

console.log("Parsed API Response:", data);

return { students: data, total: 100 };

} catch (error) {

console.error("Error fetching students:", error.message);

return { students: [], total: 0 };

}

}

* **frontend/src/App.js**

import React, { useState, useEffect } from "react";

import { fetchStudents } from "./services/api";

import FilterForm from "./FilterForm";

import Pagination from "./Pagination";

import "./styles/main.css";

function App() {

const [students, setStudents] = useState([]);

const [filters, setFilters] = useState({});

const [page, setPage] = useState(1);

const [totalStudents, setTotalStudents] = useState(0);

const [loading, setLoading] = useState(false);

const [error, setError] = useState(null);

useEffect(() => {

const fetchData = async () => {

setLoading(true);

setError(null);

try {

const response = await fetchStudents(page, filters);

if (response && Array.isArray(response.students)) {

setStudents(response.students);

setTotalStudents(response.total);

} else {

throw new Error("Неверные данные");

}

} catch (error) {

setError("Не удалось загрузить данные. Попробуйте позже.");

} finally {

setLoading(false);

}

};

fetchData();

}, [page, filters]);

const totalPages = Math.ceil(totalStudents / 10);

return (

<div className="container">

<header className="header">

<h1>Список студентов</h1>

<p>

Всего студентов: <strong>{totalStudents}</strong>

</p>

</header>

<FilterForm setFilters={setFilters} />

{loading ? (

<p className="loading">Загрузка данных...</p>

) : error ? (

<p className="error">{error}</p>

) : (

<table className="student-table">

<thead>

<tr>

<th>Фамилия</th>

<th>Имя</th>

<th>Отчество</th>

<th>Курс</th>

<th>Группа</th>

<th>Факультет</th>

</tr>

</thead>

<tbody>

{Array.isArray(students) && students.length > 0 ? (

students.map((student) => (

<tr key={student.id}>

<td>{student.Фамилия}</td>

<td>{student.Имя}</td>

<td>{student.Отчество}</td>

<td>{student.Курс}</td>

<td>{student.Группа}</td>

<td>{student.Факультет}</td>

</tr>

))

) : (

<tr>

<td colSpan="6">Нет данных</td>

</tr>

)}

</tbody>

</table>

)}

<Pagination currentPage={page} setPage={setPage} totalPages={totalPages} />

</div>

);

}

export default App;

* **frontend/src/FilterForm.js**

import React, { useState } from "react";

import "./styles/main.css";

function FilterForm({ setFilters }) {

const [localFilters, setLocalFilters] = useState({});

const handleChange = (e) => {

setLocalFilters({ ...localFilters, [e.target.name]: e.target.value });

};

const handleSubmit = (e) => {

e.preventDefault();

const validFilters = Object.fromEntries(

Object.entries(localFilters).filter(([key, value]) => value.trim() !== "")

);

setFilters(validFilters);

};

const handleClear = () => {

setLocalFilters({});

setFilters({});

};

return (

<form className="filter-form" onSubmit={handleSubmit}>

<div className="form-group">

<label htmlFor="Фамилия">Фамилия:</label>

<input

id="Фамилия"

name="Фамилия"

placeholder="Введите фамилию"

value={localFilters["Фамилия"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-group">

<label htmlFor="Имя">Имя:</label>

<input

id="Имя"

name="Имя"

placeholder="Введите имя"

value={localFilters["Имя"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-group">

<label htmlFor="Отчество">Отчество:</label>

<input

id="Отчество"

name="Отчество"

placeholder="Введите отчество"

value={localFilters["Отчество"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-group">

<label htmlFor="Курс">Курс:</label>

<input

id="Курс"

name="Курс"

placeholder="Введите курс"

value={localFilters["Курс"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-group">

<label htmlFor="Группа">Группа:</label>

<input

id="Группа"

name="Группа"

placeholder="Введите группу"

value={localFilters["Группа"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-group">

<label htmlFor="Факультет">Факультет:</label>

<input

id="Факультет"

name="Факультет"

placeholder="Введите факультет"

value={localFilters["Факультет"] || ""}

onChange={handleChange}

/>

</div>

<div className="form-actions">

<button className="btn btn-submit" type="submit">

Применить фильтры

</button>

<button

className="btn btn-clear"

type="button"

onClick={handleClear}

>

Очистить фильтры

</button>

</div>

</form>

);

}

export default FilterForm;

* **frontend/src/index.js**

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));

* **frontend/src/Pagination.js**

import React from "react";

import "./styles/main.css";

function Pagination({ currentPage, setPage, totalPages }) {

const handlePrevious = () => {

if (currentPage > 1) {

setPage(currentPage - 1);

}

};

const handleNext = () => {

if (currentPage < totalPages) {

setPage(currentPage + 1);

}

};

return (

<div className="pagination">

<button

className="pagination-button"

disabled={currentPage === 1}

onClick={handlePrevious}

>

&laquo; Предыдущая

</button>

<span className="pagination-info">

Страница {currentPage} из {totalPages}

</span>

{Array.from({ length: totalPages }, (\_, i) => (

<button

key={i}

className={`pagination-button ${

currentPage === i + 1 ? "active" : ""

}`}

onClick={() => setPage(i + 1)}

>

{i + 1}

</button>

))}

<button

className="pagination-button"

disabled={currentPage === totalPages}

onClick={handleNext}

>

Следующая &raquo;

</button>

</div>

);

}

export default Pagination;

* **frontend/src/styles/main.css**

.filter-form {

display: grid;

grid-template-columns: repeat(2, 1fr);

gap: 10px;

margin-bottom: 20px;

}

.form-group {

display: flex;

flex-direction: column;

gap: 5px;

}

.form-group label {

font-size: 0.9rem;

color: #34495e;

}

.filter-form input {

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 1rem;

}

.filter-form input:focus {

border-color: #3498db;

outline: none;

}

.form-actions {

grid-column: span 2;

display: flex;

justify-content: space-between;

}

.btn {

padding: 10px 20px;

border: none;

border-radius: 4px;

font-size: 1rem;

cursor: pointer;

transition: background 0.3s;

}

.btn-submit {

background: #27ae60;

color: white;

}

.btn-submit:hover {

background: #2ecc71;

}

.btn-clear {

background: #e74c3c;

color: white;

}

.btn-clear:hover {

background: #c0392b;

}

.student-table {

width: 100%;

max-width: 1200px;

margin: 20px auto;

border-collapse: collapse;

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);

font-size: 1rem;

text-align: center;

}

.student-table th,

.student-table td {

border: 1px solid #ddd;

padding: 10px;

}

.student-table th {

background-color: #f4f4f4;

font-weight: bold;

text-transform: uppercase;

}

.student-table tbody tr:nth-child(even) {

background-color: #f9f9f9;

}

.student-table tbody tr:hover {

background-color: #eaf3fa;

}

.container {

display: flex;

flex-direction: column;

align-items: center;

}

.header {

text-align: center;

margin-bottom: 20px;

}

.loading,

.error {

text-align: center;

font-size: 1.2rem;

color: #333;

}

# Заключение

Проект представляет собой модульную систему с Frontend на React и Backend на FastAPI. Сильные стороны включают RESTful API с фильтрацией, поддержку CORS и контейнеризацию через Docker. Система масштабируема, легко расширяется и готова к внедрению.

**Список использованных источников**

* FastAPI
  + Документация FastAPI<https://fastapi.tiangolo.com/>
  + FastAPI Tutorial <https://realpython.com/fastapi-python-web-apis/>
* PostgreSQL
  + Официальная документация PostgreSQL <https://www.postgresql.org/docs/>
  + SQLAlchemy ORM <https://docs.sqlalchemy.org/>
* Docker
  + Документация Docker <https://docs.docker.com/>
  + Docker Compose Guide <https://docs.docker.com/compose/>
* Nginx
  + Официальная документация Nginx <https://nginx.org/ru/docs/>
  + Nginx и настройка обратного прокси <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>
* JWT JSON Web Tokens
  + Документация JWT <https://jwt.io/introduction/>
  + Использование JWT в FastAPI <https://github.com/fastapi/fastapi>
* WebSocket
  + Введение в WebSocket <https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API>
  + WebSocket в FastAPI <https://fastapi.tiangolo.com/advanced/websockets/>
* Аутентификация и авторизация
  + OAuth 2.0 и OpenID Connect <https://auth0.com/docs/get-started> – рекомендации по протоколам безопасности
  + Обеспечение безопасности API в FastAPI <https://fastapi.tiangolo.com/tutorial/security/>
* Микросервисная архитектура
  + Обзор микросервисной архитектуры <https://martinfowler.com/microservices/>
  + Микросервисы и Docker <https://docs.docker.com/get-started/overview/>
* Проектирование и нормализация баз данных
  + Принципы нормализации базы данных <https://www.guru99.com/database-normalization.html>
  + ER диаграммы для проектирования базы данных <https://vertabelo.com/blog/what-is-an-entity-relationship-diagram/>
* Шаблоны и генерация HTML на сервере
  + Использование Jinja2 в FastAPI <https://fastapi.tiangolo.com/advanced/templates/>
* Работа с асинхронными функциями
  + Асинхронное программирование в Python <https://docs.python.org/3/library/asyncio.html>
  + Асинхронный ввод/вывод в FastAPI <https://fastapi.tiangolo.com/async/>