**Содержание**

Введение

1. Разработка технического задания

2. Проектирование интерфейса приложения

Интерфейс полностью описывать не надо. Пишем где делаем, для чего нужно, что такое фигма, почему фигму, + и –

Представляет собой приложение предназначенное для управления студентами, курсами и записями об их прохождении. Основной функционал включает работу с тремя связанными между собой сущностями: студентами, учебными курсами и информацией о прохождении курсов. Учебный администратор может добавлять, редактировать и удалять студентов, искать их по фамилии, имени и отчеству, управлять списком курсов с указанием их названий, описаний и преподавателей, а также вести учёт прохождения курсов студентами, фиксируя дату и оценку.

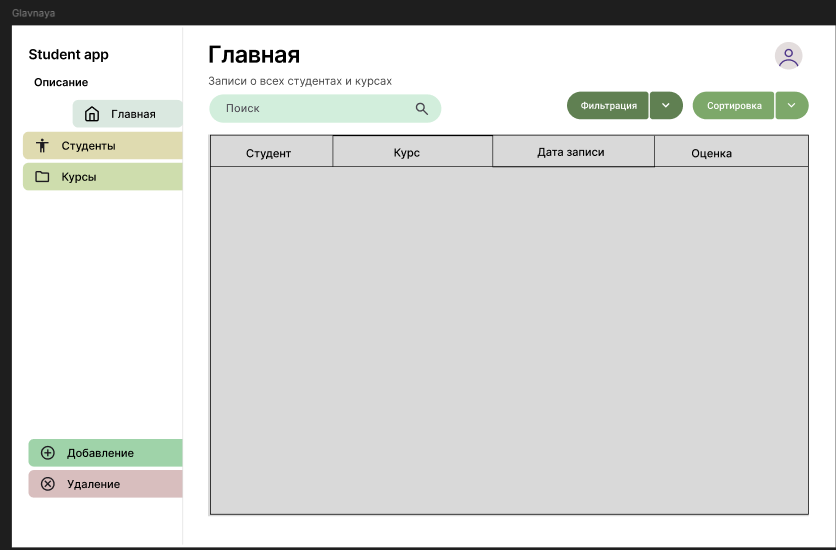


Рис.1 «Главная»

На данной странице Учебный администратор может просматривать записи о прохождении курсов студентами, дату и оценку, полученную студентом.Также имеется возможность произвести поиск,фильтрацию и сортировку.

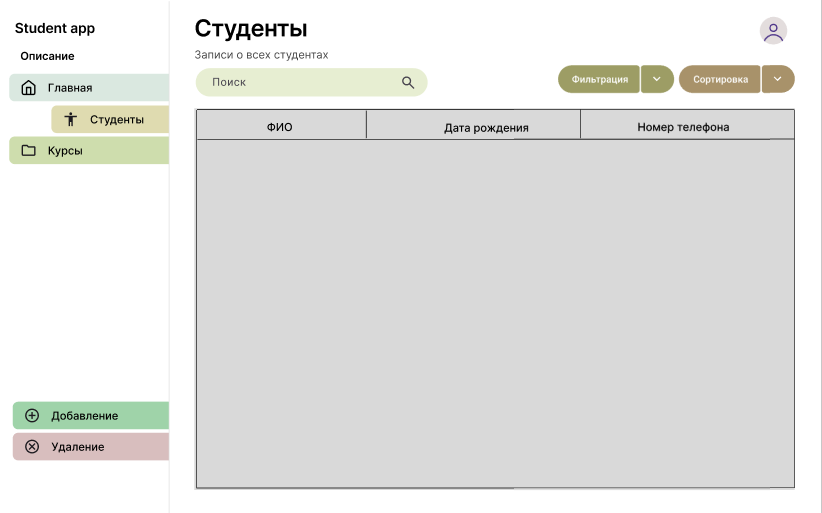


Рис.2 «Студенты»

На данной странице Учебный администратор может просматривать записи о всех студентах. .Также имеется возможность произвести поиск,фильтрацию и сортировку.

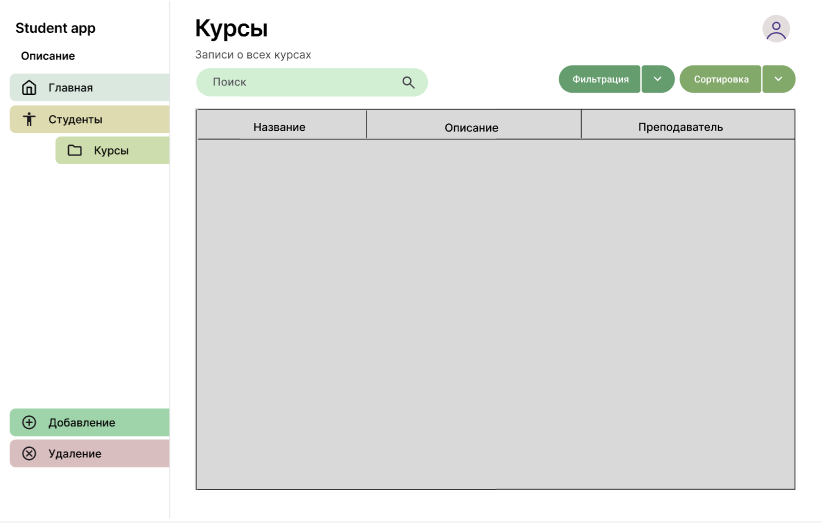


Рис.3 «Курсы»

На данной странице Учебный администратор может просматривать записи о всех курсах. .Также имеется возможность произвести поиск,фильтрацию и сортировку.

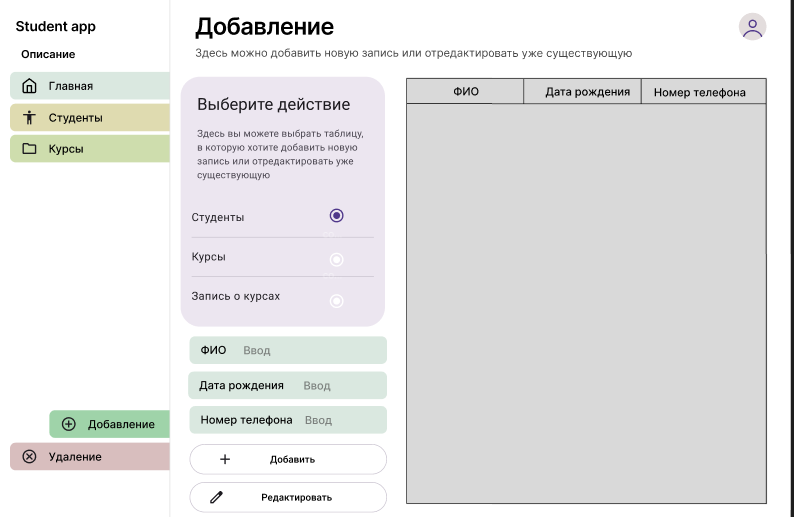


Рис.4 «Добавление»

На данной странице Учебный администратор может добавлять новых студентов,курсы, а также запись о прохождении курса. Кроме того, имеется функция редактирования уже имеющихся записей в таблицах.

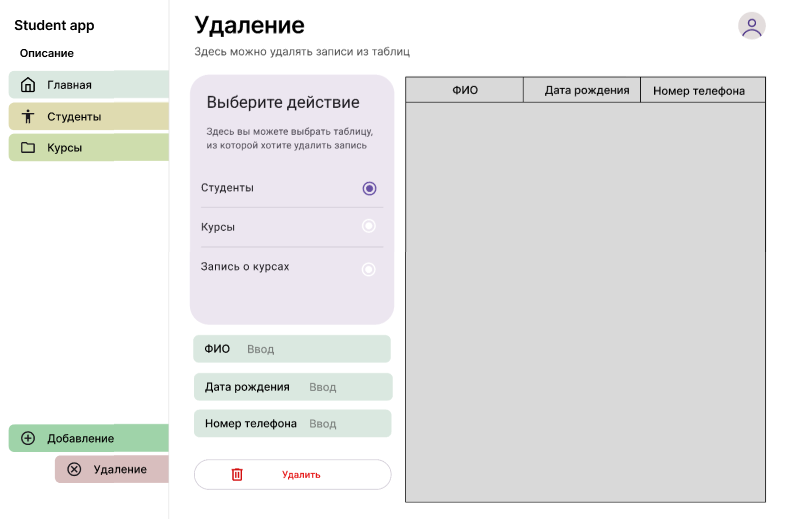


Рис.5 «Удаление»

На данной странице Учебный администратор может удалять записи, из выбранной таблицы.

3. Проектирование и создание базы данных

3.1. Разработка структуры БД

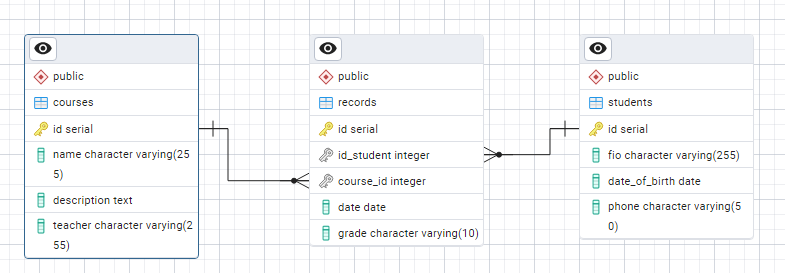


Рис.6 «ER-диаграмма»

Таблица Students нужна для хранения данных о студентах. В ней имеются такие поля:

* id – Первичный ключ в таблице Students. Тип данных integer, not null, primary key, default nextval('students\_id\_seq'::regclass).
* fio – ФИО студента. Тип данных character varying(255), not null
* date\_of\_birth – День рождения студента. Тип данных date, not null
* phone – Номер телефона студента. Тип данных character varying(50)

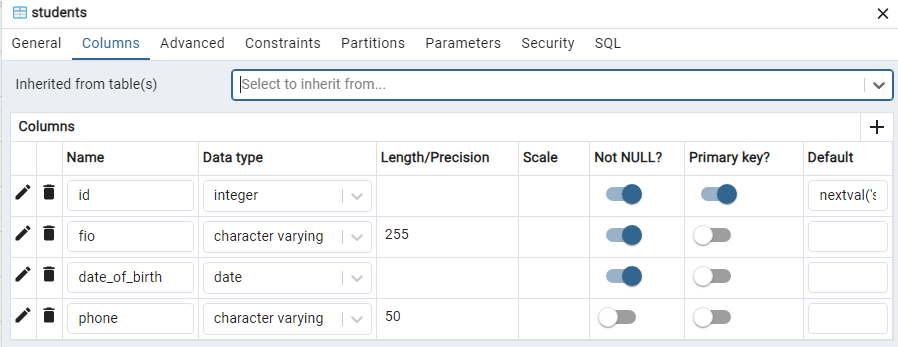


Рис.7 «Таблица Students»

Таблица Courses нужна для хранения данных о курсах. В ней имеются такие поля:

* id - Первичный ключ в таблице Courses. Тип данных integer, not null, primary key, default nextval('courses\_id\_seq'::regclass).
* name – Название курса. Тип данных character varying(255), not null
* description – Описание курса. Тип данных text
* teacher – ФИО преподавателя, который ведёт тот или иной курс. Тип данных character varying(255)

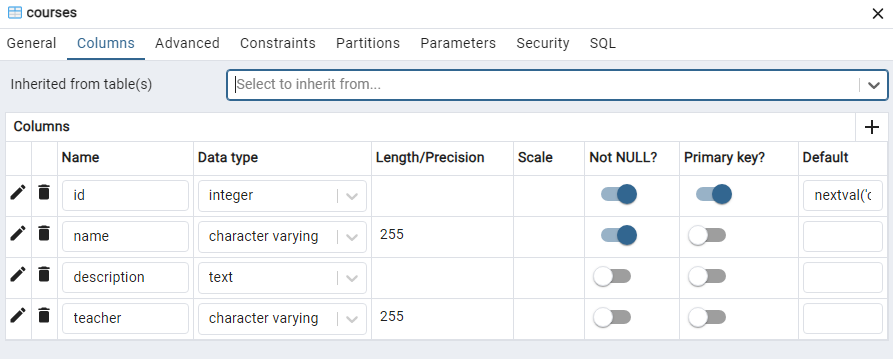


Рис.8 «Таблица Courses»

Таблица Records нужна для хранения данных о прохождении студентами курсов. В ней имеются такие поля:

* id - Первичный ключ в таблице Records. Тип данных integer, not null, primary key, default nextval('records\_id\_seq'::regclass)
* id\_student – Внешний ключ из таблицы Students. Тип данных integer, not null
* course\_id - Внешний ключ из таблицы Courses. Тип данных integer, not null
* Date – Дата проведения курса. Тип данных date, not null
* Grade – Оценка,полученная студентом. Тип данных character varying

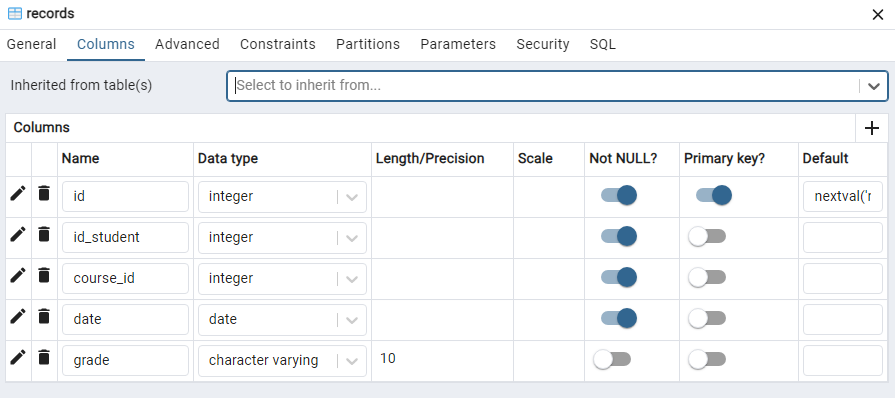


Рис.9 «Таблица Records»

3.2. Создание модели базы данных

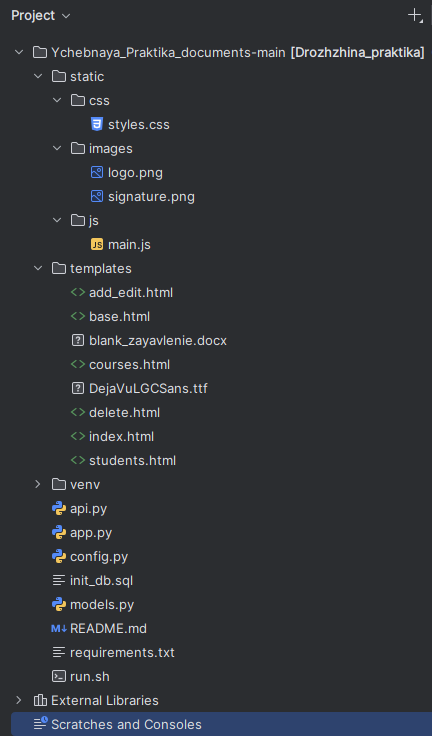


Рис.10 «Иерархия файлов»

- модуль классов

Листинг 1 - класс Студенты

class Student(Base):  
 \_\_tablename\_\_ = 'students'  
 id = Column(Integer, primary\_key=True)  
 fio = Column(String(255), nullable=False)  
 date\_of\_birth = Column(Date, nullable=False)  
 phone = Column(String(50))  
  
 records = relationship("Record", back\_populates="student", cascade="all, delete-orphan")  
  
 def to\_dict(self):  
 return {  
 "id": self.id,  
 "fio": self.fio,  
 "date\_of\_birth": self.date\_of\_birth.isoformat() if self.date\_of\_birth else None,  
 "phone": self.phone  
 }

Листинг 2 - класс Курсы

class Course(Base):  
 \_\_tablename\_\_ = 'courses'  
 id = Column(Integer, primary\_key=True)  
 name = Column(String(255), nullable=False)  
 description = Column(Text)  
 teacher = Column(String(255))  
  
 records = relationship("Record", back\_populates="course", cascade="all, delete-orphan")  
  
 def to\_dict(self):  
 return {  
 "id": self.id,  
 "name": self.name,  
 "description": self.description,  
 "teacher": self.teacher  
 }

Листинг 3 - класс Записи

class Record(Base):  
 \_\_tablename\_\_ = 'records'  
 id = Column(Integer, primary\_key=True)  
 id\_student = Column(Integer, ForeignKey('students.id', ondelete="CASCADE"), nullable=False)  
 course\_id = Column(Integer, ForeignKey('courses.id', ondelete="CASCADE"), nullable=False)  
 date = Column(Date, nullable=False)  
 grade = Column(String(10)) *# '5','4','3','2' or 'Не оценено'* student = relationship("Student", back\_populates="records")  
 course = relationship("Course", back\_populates="records")  
  
 def to\_dict(self):  
 return {  
 "id": self.id,  
 "id\_student": self.id\_student,  
 "student\_fio": self.student.fio if self.student else None,  
 "course\_id": self.course\_id,  
 "course\_name": self.course.name if self.course else None,  
 "date": self.date.isoformat() if self.date else None,  
 "grade": self.grade  
 }

4. Разработка программных модулей

4.1. Создание сервера

- модуль подключения к базе данных

Листинг 4 - Подключение к базе данных

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
DATABASE\_URI = os.getenv("DATABASE\_URL", "postgresql+psycopg://postgres:1234@localhost:5433/course\_records\_db")  
SECRET\_KEY = os.getenv("SECRET\_KEY", "dev-secret-key")

- модуль функций доступа к данным API (GET, POST, PUT, DELETE)

1. Студенты

**1.1 Получение списка студентов с фильтрацией и сортировкой**

Листинг 5 - Получение списка студентов

@api.route('/students', methods=['GET'])

def list\_students():

q = request.args.get('q', '').strip()

sort = request.args.get('sort', 'default')

db = SessionLocal()

query = db.query(Student)

if q:

query = query.filter(Student.fio.ilike(f'%{q}%'))

if sort == 'asc':

query = query.order\_by(asc(Student.date\_of\_birth))

elif sort == 'desc':

query = query.order\_by(desc(Student.date\_of\_birth))

students = [s.to\_dict() for s in query.all()]

db.close()

return jsonify(students)

1. Получаем параметр q из строки запроса, который может быть использован для фильтрации. Если параметр не передан, по умолчанию будет пустая строка. .strip() удаляет лишние пробелы с обеих сторон строки.

2. Получаем параметр sort, который отвечает за сортировку. Если параметр не передан, по умолчанию сортировка будет установлена как 'default'.

3. Создаем подключение к базе данных, используя сессию, которую определяет SessionLocal. Это объект SQLAlchemy, управляющий соединениями с базой данных.

4. Строим запрос к таблице Student. db.query(Student) создаёт базовый объект запроса для работы с моделью Student.

5. Проверяем, был ли передан фильтр для поиска. Если параметр q не пустой, то продолжаем выполнять фильтрацию.

6. Если фильтр существует, добавляем в запрос фильтрацию, которая ищет студентов по полю fio, используя частичное совпадение (аналог SQL LIKE). ilike — это нечувствительный к регистру поиск.

7. Проверяем, какой параметр сортировки передан (в данном случае 'asc' — сортировка по возрастанию).

8. Если сортировка по возрастанию, то добавляем в запрос сортировку по полю date\_of\_birth (дата рождения студентов) в порядке возрастания.

9. Если сортировка по убыванию, то добавляем в запрос сортировку по полю date\_of\_birth в порядке убывания.

10. Выполняем сортировку по убыванию для поля date\_of\_birth.

11. Получаем все результаты из базы данных с помощью query.all(), после чего конвертируем каждый объект студента в словарь с помощью метода to\_dict(). Это необходимо, чтобы вернуть данные в формате JSON.

12. Закрываем сессию с базой данных после выполнения запроса, чтобы освободить ресурсы.

13. Возвращаем результат запроса в формате JSON, используя функцию jsonify из Flask.

Листинг 6 - JavaScript для поиска,фильтрации и сортирорвки студентов

$('#students-search').on('input', function() {  
 const q = $(this).val();  
 const sort = $('#students-sort-asc').hasClass('active') ? 'asc' : ($('#students-sort-desc').hasClass('active') ? 'desc' : 'default');  
 loadStudents({ q, sort });  
});  
$('#students-sort-default').on('click', function(){ $('.btn-group .btn').removeClass('active'); $(this).addClass('active'); $('#students-search').trigger('input'); });  
$('#students-sort-asc').on('click', function(){ $('.btn-group .btn').removeClass('active'); $(this).addClass('active'); $('#students-search').trigger('input'); });  
$('#students-sort-desc').on('click', function(){ $('.btn-group .btn').removeClass('active'); $(this).addClass('active'); $('#students-search').trigger('input'); });

1.Добавляем обработчик события для поля ввода с ID students-search. Когда пользователь что-то вводит в поле, срабатывает событие input.

2.Получаем текущее значение из поля ввода и сохраняем его в переменную q. Это будет использоваться как фильтр в запросе.

3.Определяем, какой параметр сортировки выбран. Если кнопка сортировки по возрастанию (students-sort-asc) имеет класс active, то сортируем по возрастанию. Если кнопка сортировки по убыванию (students-sort-desc) активна, то выбираем сортировку по убыванию. Если ни одна не выбрана, то устанавливаем сортировку по умолчанию.

4.Вызов функции loadStudents с параметрами фильтрации (q) и сортировки (sort). Эта функция отправляет запрос на сервер с нужными параметрами.

5.Обработчики кликов для сортировки по умолчанию, по возрастанию и по убыванию: эти блоки кода отвечают за обновление классов активных кнопок сортировки. Когда пользователь нажимает на одну из кнопок сортировки, она получает класс active, а остальные кнопки — его теряют. После этого срабатывает событие input для поля поиска, чтобы отправить новый запрос на сервер с актуальными параметрами.

Листинг 7 – Загрузка студентов

function loadStudents(params={}) {  
 $.getJSON('/api/students', params).done(function(data) {  
 const tbody = $('#students-table tbody');  
 tbody.empty();  
 data.forEach(s => {  
 const row = `<tr data-id="${s.id}">  
 <td>${s.fio}</td>  
 <td>${formatDate(s.date\_of\_birth)}</td>  
 <td>${s.phone || ''}</td>  
 <td>  
 <a href="/documents/generate-word/${s.id}" class="btn btn-primary btn-sm" target="\_blank">Word</a>  
 <a href="/pdf/generate-pdf/${s.id}" class="btn btn-danger btn-sm" target="\_blank">PDF</a>  
 </td>  
 </tr>`;  
 tbody.append(row);  
 });  
 });  
}

1.Объявление функции loadStudents(params = {}) — она принимает параметры фильтрации и сортировки (например, q и sort) и по ним делает запрос на сервер.

2.$.getJSON('/api/students', params) — отправляет AJAX-запрос к Flask API /api/students, передавая параметры.

3. .done(function(data) {...}) — выполняется, когда сервер успешно возвращает данные.

4. const tbody = $('#students-table tbody'); tbody.empty(); — очищает таблицу перед вставкой новых строк.

5.data.forEach(s => {...}) — перебирает список студентов, возвращённый сервером.

6.Формируется HTML-строка <tr> для каждого студента, включая:

* ФИО (s.fio),
* дату рождения (через formatDate),
* телефон (s.phone),
* кнопки для генерации Word и PDF документов.

7. tbody.append(row) — добавляет каждую строку в таблицу. Таким образом, таблица обновляется динамически без перезагрузки страницы.

**1.2 Создание нового студента**

Листинг 8 – Создание нового студента

@api.route('/students', methods=['POST'])

def create\_student():

"""Создать нового студента"""

data = request.json

db = SessionLocal()

try:

dob = datetime.fromisoformat(data['date\_of\_birth']).date()

except Exception:

return jsonify({"error": "Invalid date\_of\_birth"}), 400

student = Student(fio=data['fio'], date\_of\_birth=dob, phone=data.get('phone'))

db.add(student)

db.commit()

db.refresh(student)

res = student.to\_dict()

db.close()

return jsonify(res), 201

Листинг 9 - JavaScript для создания нового студента

$('#student-add-btn').on('click', function(){

const payload = {

fio: $('#student-fio').val().trim(),

date\_of\_birth: $('#student-dob').val(),

phone: $('#student-phone').val().trim()

};

$.ajax({ url:'/api/students', type:'POST', contentType:'application/json', data: JSON.stringify(payload)})

.done(function(){ alert('Студент добавлен'); loadAddList('students'); loadStudents(); })

.fail(function(){ alert('Ошибка добавления'); });

});

**1.3 Обновление данных студента**

Листинг 10 – Обновление данных студента

@api.route('/students/<int:student\_id>', methods=['PUT'])

def update\_student(student\_id):

"""Обновить данные студента"""

data = request.json

db = SessionLocal()

student = db.query(Student).get(student\_id)

if not student:

db.close()

return jsonify({"error": "Not found"}), 404

student.fio = data.get('fio', student.fio)

if data.get('date\_of\_birth'):

student.date\_of\_birth = datetime.fromisoformat(data['date\_of\_birth']).date()

student.phone = data.get('phone', student.phone)

db.commit()

res = student.to\_dict()

db.close()

return jsonify(res)

Листинг 11 – JavaScript для обновления данных студента

$('#student-edit-btn').on('click', function(){

const id = $(this).data('id');

const payload = { fio: $('#student-fio').val().trim(), date\_of\_birth: $('#student-dob').val(), phone: $('#student-phone').val().trim() };

$.ajax({ url:`/api/students/${id}`, type:'PUT', contentType:'application/json', data: JSON.stringify(payload) })

.done(function(){ alert('Сохранено'); loadAddList('students'); loadStudents(); $('#student-edit-btn').addClass('d-none'); $('#student-add-btn').removeClass('d-none'); })

.fail(function(){ alert('Ошибка сохранения'); });

});

**1.4 Удаление данных о студенте**

Листинг 12 – Удаление студента

@api.route('/students/<int:student\_id>', methods=['DELETE'])

def delete\_student(student\_id):

"""Удалить студента"""

db = SessionLocal()

student = db.query(Student).get(student\_id)

if not student:

db.close()

return jsonify({"error": "Not found"}), 404

db.delete(student)

db.commit()

db.close()

return jsonify({"status": "deleted"})

Листинг 13 – JavaScript для удаления студента

$('#delete-confirm-btn').on('click', function(){

const id = $(this).data('id');

const table = $(this).data('table');

let url = '/api/';

if (table === 'students') url += `students/${id}`;

else if (table === 'courses') url += `courses/${id}`;

else url += `records/${id}`;

$.ajax({ url, type: 'DELETE' }).done(function(){

alert('Удалено');

loadDeleteList(table);

}).fail(function(err){ alert('Ошибка удаления'); });

});

2. Курсы

**2.1 Получение списка курсов**

Листинг 14 – Получение списка курсов

@api.route('/courses', methods=['GET'])

def list\_courses():

"""Список курсов"""

q = request.args.get('q', '').strip()

teacher = request.args.get('teacher', '').strip()

db = SessionLocal()

query = db.query(Course)

if q:

query = query.filter(Course.name.ilike(f'%{q}%'))

if teacher:

query = query.filter(Course.teacher == teacher)

courses = [c.to\_dict() for c in query.all()]

db.close()

return jsonify(courses)

Листинг 15 – JavaScript для получения списка курсов

function loadCourses(params={}) {

$.getJSON('/api/courses', params).done(function(data) {

const tbody = $('#courses-table tbody');

tbody.empty();

data.forEach(c => {

const row = `<tr data-id="${c.id}"><td>${c.name}</td><td>${c.description || ''}</td><td>${c.teacher || ''}</td></tr>`;

tbody.append(row);

});

});

}

**2.2 Создание курса**

Листинг 16– Создание курса

@api.route('/courses', methods=['POST'])

def create\_course():

"""Создать курс"""

data = request.json

db = SessionLocal()

course = Course(name=data['name'], description=data.get('description'), teacher=data.get('teacher'))

db.add(course)

db.commit()

db.refresh(course)

res = course.to\_dict()

db.close()

return jsonify(res), 201

Листинг 17 – JavaScript для создания курса

$('#course-add-btn').on('click', function(){

const payload = {

name: $('#course-name').val().trim(),

description: $('#course-desc').val().trim(),

teacher: $('#course-teacher').val().trim()

};

$.ajax({ url:'/api/courses', type:'POST', contentType:'application/json', data: JSON.stringify(payload)})

.done(function(){ alert('Курс добавлен'); loadAddList('courses'); fillCourseFilter(); fillTeachersFilter(); })

.fail(function(){ alert('Ошибка добавления'); });

});

**2.3 Обновление курса**

Листинг 18 – Обновление курса

@api.route('/courses/<int:course\_id>', methods=['PUT'])

def update\_course(course\_id):

"""Обновить курс"""

data = request.json

db = SessionLocal()

course = db.query(Course).get(course\_id)

if not course:

db.close()

return jsonify({"error": "Not found"}), 404

course.name = data.get('name', course.name)

course.description = data.get('description', course.description)

course.teacher = data.get('teacher', course.teacher)

db.commit()

res = course.to\_dict()

db.close()

return jsonify(res)

Листинг 19 – JavaScript для обновления курса

$('#course-edit-btn').on('click', function(){

const id = $(this).data('id');

const payload = { name: $('#course-name').val().trim(), description: $('#course-desc').val().trim(), teacher: $('#course-teacher').val().trim() };

$.ajax({ url:`/api/courses/${id}`, type:'PUT', contentType:'application/json', data: JSON.stringify(payload) })

.done(function(){ alert('Сохранено'); loadAddList('courses'); fillCourseFilter(); fillTeachersFilter(); $('#course-edit-btn').addClass('d-none'); $('#course-add-btn').removeClass('d-none'); })

.fail(function(){ alert('Ошибка сохранения'); });

});

**2.4 Удаление курса**

Листинг 20 – Удаление курса

@api.route('/courses/<int:course\_id>', methods=['DELETE'])

def delete\_course(course\_id):

"""Удалить курс"""

db = SessionLocal()

course = db.query(Course).get(course\_id)

if not course:

db.close()

return jsonify({"error": "Not found"}), 404

db.delete(course)

db.commit()

db.close()

return jsonify({"status": "deleted"})

Листинг 21 – JavaScript для удаления курса

$('#delete-confirm-btn').on('click', function(){

const id = $(this).data('id');

const table = $(this).data('table');

let url = '/api/';

if (table === 'students') url += `students/${id}`;

else if (table === 'courses') url += `courses/${id}`;

else url += `records/${id}`;

$.ajax({ url, type: 'DELETE' }).done(function(){

alert('Удалено');

loadDeleteList(table);

}).fail(function(err){ alert('Ошибка удаления'); });

});

3.Записи

**3.1 Получение списка записей**

Листинг 22 – Получение списка записей

@api.route('/records', methods=['GET'])

def list\_records():

"""Список записей"""

q = request.args.get('q', '').strip()

course\_id = request.args.get('course\_id', '').strip()

sort = request.args.get('sort', 'default')

db = SessionLocal()

query = db.query(Record).join(Record.student).join(Record.course)

if q:

query = query.filter(or\_(Student.fio.ilike(f'%{q}%'), Course.name.ilike(f'%{q}%')))

if course\_id:

try:

cid = int(course\_id)

query = query.filter(Record.course\_id == cid)

except ValueError:

pass

if sort == 'asc':

query = query.order\_by(asc(Record.date))

elif sort == 'desc':

query = query.order\_by(desc(Record.date))

results = [r.to\_dict() for r in query.all()]

db.close()

return jsonify(results)

Листинг 23 – JavaScript для получения записей

function loadRecords(params={}) {

$.getJSON('/api/records', params).done(function(data) {

const tbody = $('#records-table tbody');

tbody.empty();

data.forEach(r => {

const row = `<tr data-id="${r.id}">

<td>${r.student\_fio || ''}</td>

<td>${r.course\_name || ''}</td>

<td>${formatDate(r.date)}</td>

<td>${renderGrade(r.grade)}</td>

</tr>`;

tbody.append(row);

});

});

}

**3.2 Добавление записи**

Листинг 24 – Добавление записи

@api.route('/records', methods=['POST'])

def create\_record():

"""Создать запись"""

data = request.json

db = SessionLocal()

try:

dt = datetime.fromisoformat(data['date']).date()

except Exception:

return jsonify({"error": "Invalid date"}), 400

rec = Record(id\_student=data['id\_student'], course\_id=data['course\_id'], date=dt, grade=data.get('grade'))

db.add(rec)

db.commit()

db.refresh(rec)

res = rec.to\_dict()

db.close()

return jsonify(res), 201

Листинг 25 – JavaScript

$('#record-add-btn').on('click', function(){

const payload = {

id\_student: parseInt($('#record-student').val()),

course\_id: parseInt($('#record-course').val()),

date: $('#record-date').val(),

grade: $('#record-grade').val()

};

$.ajax({ url:'/api/records', type:'POST', contentType:'application/json', data: JSON.stringify(payload)})

.done(function(){ alert('Запись добавлена'); loadAddList('records'); loadRecords(); })

.fail(function(){ alert('Ошибка добавления'); });

});

**3.3 Обновление записи**

Листинг 26 – Обновление записи

@api.route('/records/<int:rec\_id>', methods=['PUT'])

def update\_record(rec\_id):

"""Обновить запись"""

data = request.json

db = SessionLocal()

rec = db.query(Record).get(rec\_id)

if not rec:

db.close()

return jsonify({"error": "Not found"}), 404

if 'id\_student' in data:

rec.id\_student = data['id\_student']

if 'course\_id' in data:

rec.course\_id = data['course\_id']

if 'date' in data:

rec.date = datetime.fromisoformat(data['date']).date()

if 'grade' in data:

rec.grade = data['grade']

db.commit()

res = rec.to\_dict()

db.close()

return jsonify(res)

Листинг 27 – JavaScript

$('#record-edit-btn').on('click', function(){

const id = $(this).data('id');

const payload = { id\_student: parseInt($('#record-student').val()), course\_id: parseInt($('#record-course').val()), date: $('#record-date').val(), grade: $('#record-grade').val() };

$.ajax({ url:`/api/records/${id}`, type:'PUT', contentType:'application/json', data: JSON.stringify(payload) })

.done(function(){ alert('Сохранено'); loadAddList('records'); loadRecords(); $('#record-edit-btn').addClass('d-none'); $('#record-add-btn').removeClass('d-none'); })

.fail(function(){ alert('Ошибка сохранения'); });

});

**3.4 Удаление записи**

Листинг 28 – Удаление записи

@api.route('/records/<int:rec\_id>', methods=['DELETE'])

def delete\_record(rec\_id):

"""Удалить запись"""

db = SessionLocal()

rec = db.query(Record).get(rec\_id)

if not rec:

db.close()

return jsonify({"error": "Not found"}), 404

db.delete(rec)

db.commit()

db.close()

return jsonify({"status": "deleted"})

Листинг 29 – JavaScript

$('#delete-confirm-btn').on('click', function(){

const id = $(this).data('id');

const table = $(this).data('table');

let url = '/api/';

if (table === 'students') url += `students/${id}`;

else if (table === 'courses') url += `courses/${id}`;

else url += `records/${id}`;

$.ajax({ url, type: 'DELETE' }).done(function(){

alert('Удалено');

loadDeleteList(table);

}).fail(function(err){ alert('Ошибка удаления'); });

});

4.2. Создание клиента

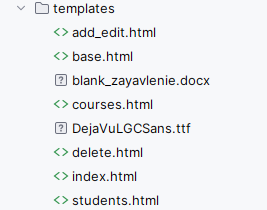


Рис. 11 «Структура папки templates»

Показываем структуру папки паблик какие файлы отвечают за что, весь html не надо.название модулей, за что каждый модуль отвечает, как они взаимодействуют с модулями пай и скрины страниц. Можно вставить листинг для пояснения, но весь код не надо.

4.2.1 add\_edit.html

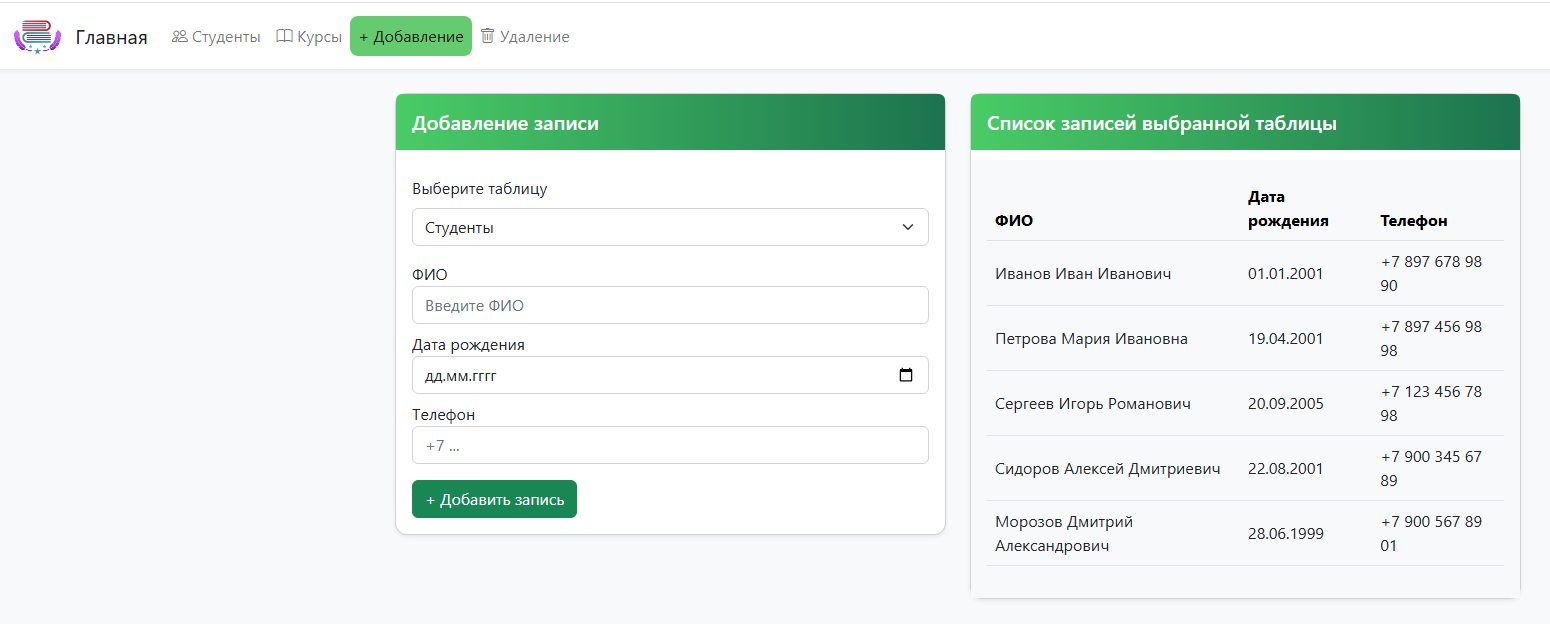


Рис.12 «Добавление»

1.Наследование шаблона. Шаблон использует базовый файл base.html и вставляет свой контент в блок {% block content %}.

2.Структура страницы

Вся страница разделена на два столбца (row):

* Левая часть (col-md-6) – формы для добавления/редактирования записей.
* Правая часть (col-md-6) – таблица для отображения данных выбранной таблицы.

3. Выбор таблицы для добавления. Позволяет выбрать, в какую таблицу будет добавлена запись. На основе выбора показывается соответствующая форма.

Листинг 30 – Выбор таблицы

<select id="add-table-select">

<option value="students">Студенты</option>

<option value="courses">Курсы</option>

<option value="records">Записи</option>

</select>

4.Формы добавления/редактирования

* Студенты – поля: ФИО, дата рождения, телефон.
* Курсы – поля: название, описание, преподаватель.
* Записи – поля: студент (select), курс (select), дата, оценка (select).

Каждая форма содержит две кнопки:

* + Добавить запись – видна по умолчанию.
* Сохранить изменения – скрыта (d-none), появляется при редактировании существующей записи.

5.Правая часть: таблица записей. Здесь динамически отображаются записи выбранной таблицы (студенты, курсы или записи) с помощью JavaScript (main.js).

6. Визуальное оформление. Используются Bootstrap-классы: card, shadow-sm, bg-gradient-header-students, form-control, form-select, btn. Заголовки и формы разделены цветными шапками для наглядности.

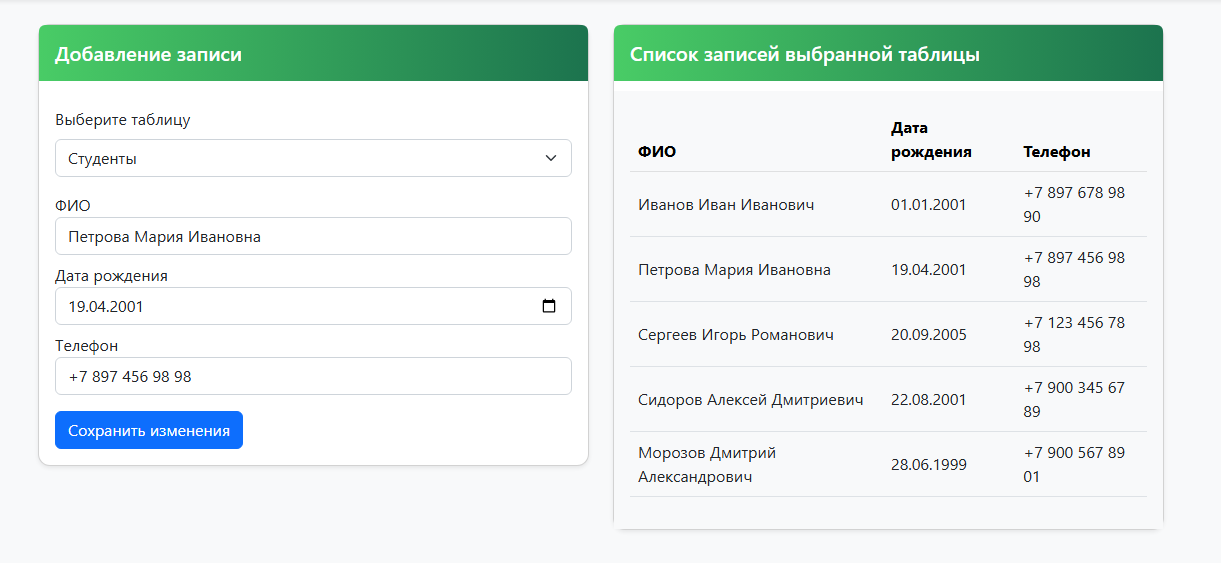


Рис.13 «Редактирование»

4.2.2 base.html

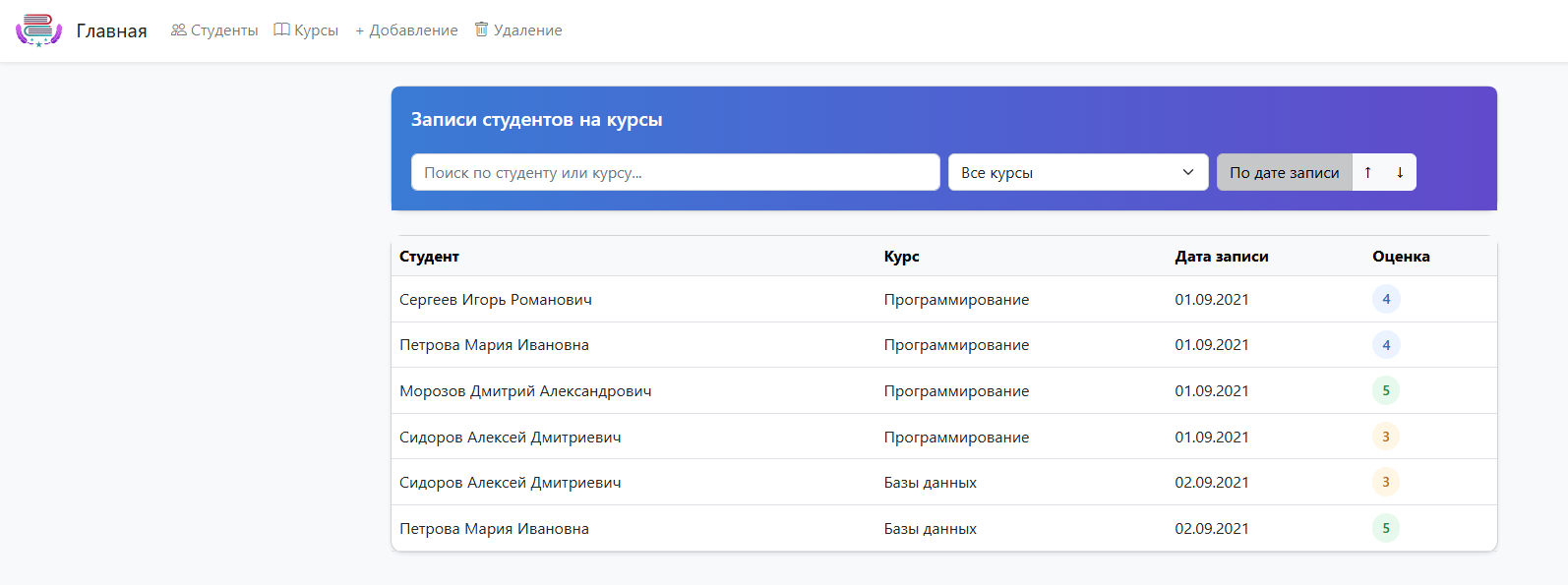


Рис.14 «Главная»

4.2.3 courses.html

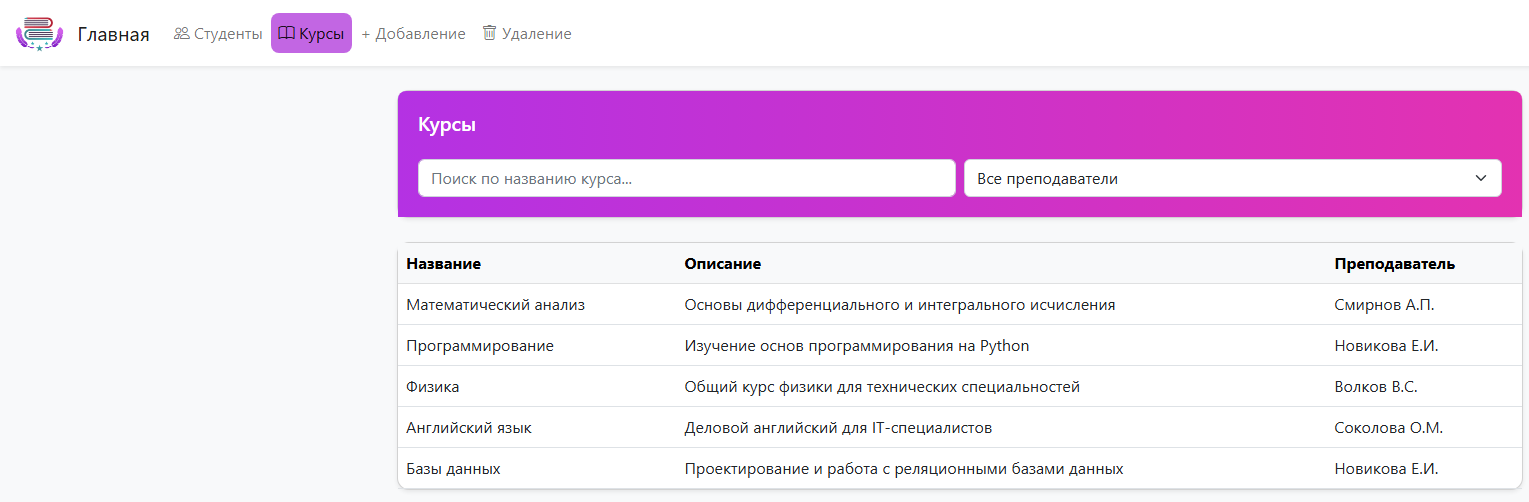


Рис.15 «Курсы»

4.2.4 delete.html

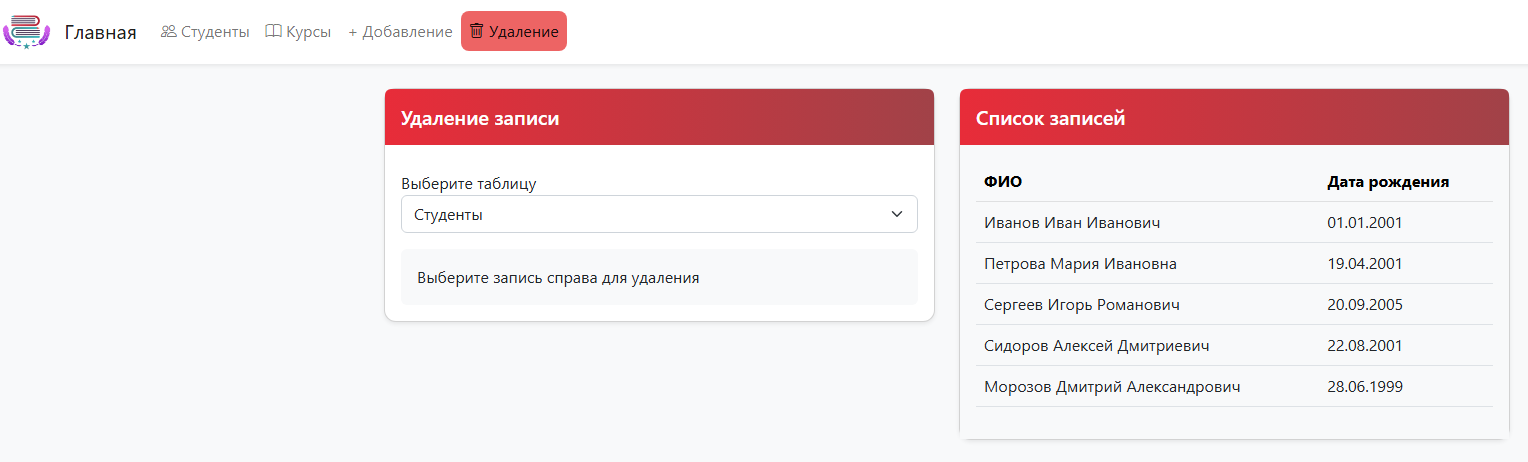


Рис.16 «Удаление»

4.2.5 index.html

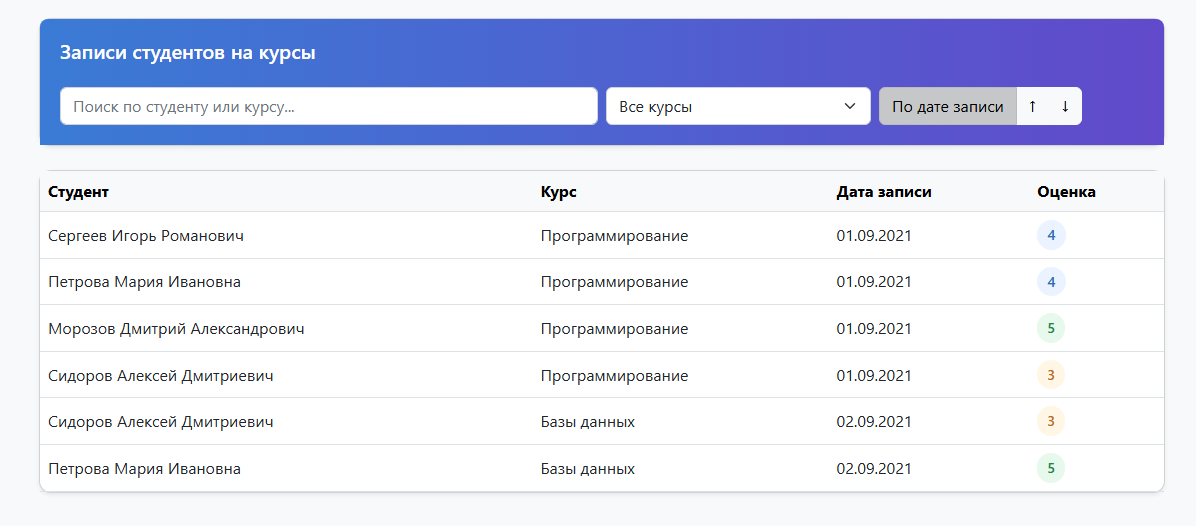


Рис.17 «Записи»

4.2.6 students.html

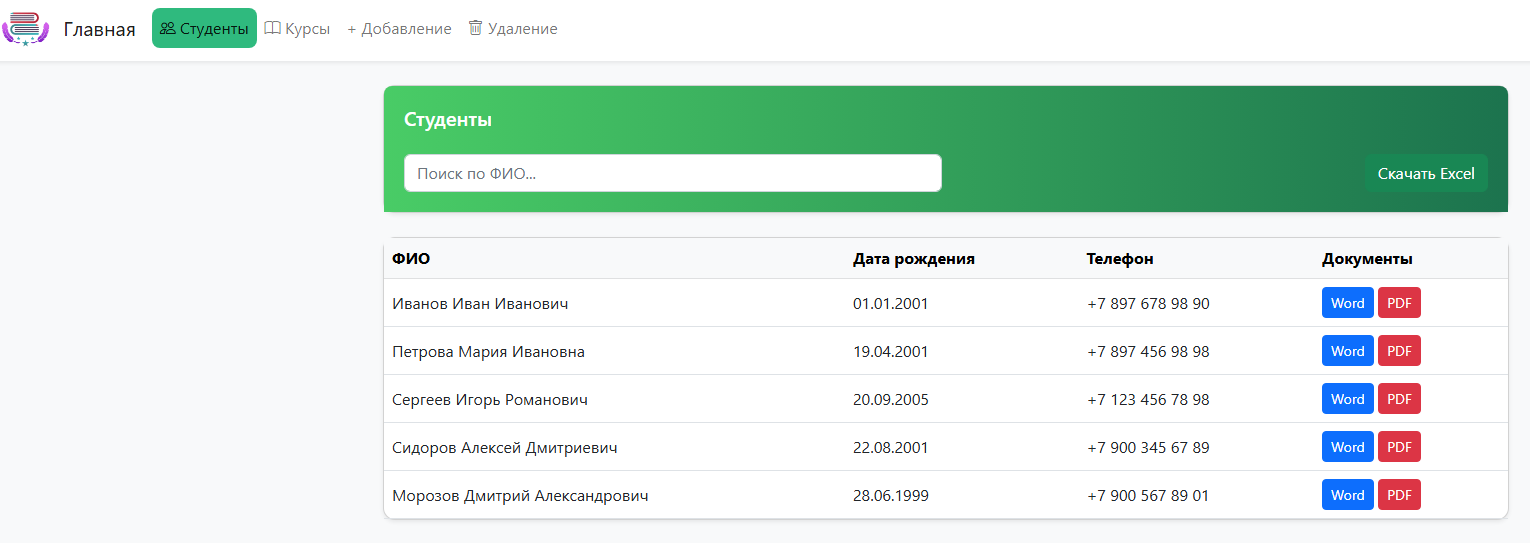


Рис.18 «Студенты»

4.3. Реализация модулей вывода на печать

- вывод в шаблон .docx

Для создания Word-документа используется шаблон:

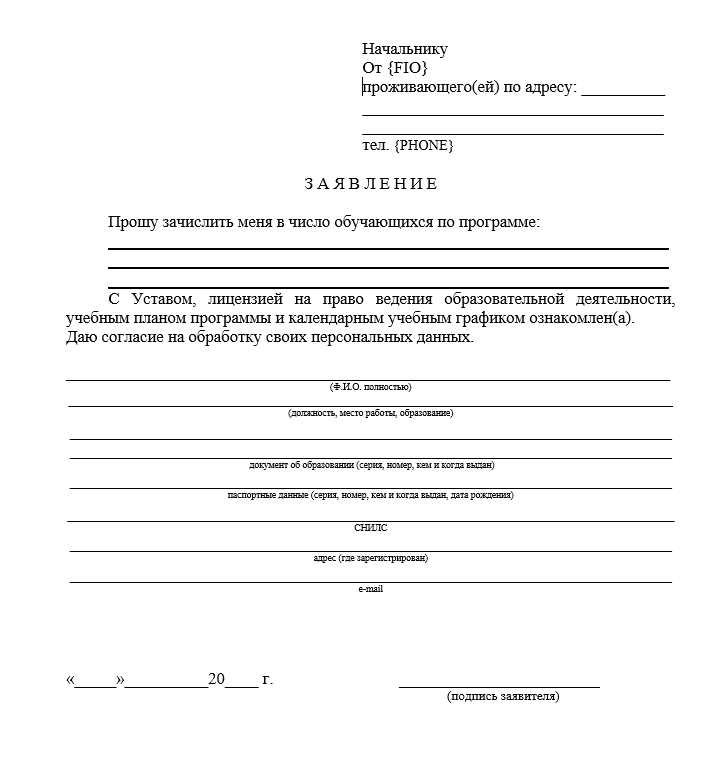


Рис.19 «Шаблон Word-документа»

Листинг 30 – Создание Word-документа

@documents\_bp.route("/generate-word/<int:student\_id>", methods=["GET"])

def generate\_word(student\_id):

"""Генерация Word-документа"""

db = SessionLocal()

student = db.query(Student).filter(Student.id == student\_id).first()

if not student:

db.close()

return jsonify({"error": "Студент не найден"}), 404

doc = Document("templates/blank\_zayavlenie.docx")

for p in doc.paragraphs:

p.text = p.text.replace("{FIO}", student.fio).replace("{PHONE}", student.phone or "")

buffer = BytesIO()

doc.save(buffer)

buffer.seek(0)

db.close()

return send\_file(buffer, as\_attachment=True, download\_name=f"Заявление\_{student.fio}.docx")

Листинг 31 -

- создание документа .xlsx

Листинг 32 -

Листинг 33 -

- создание документа .pdf

Листинг 34 -

Листинг 35 -

4.4. Реализация импорта данных из файла в базу данных

4.5. Загрузка приложения в репозиторий GitHub

5. Тестирование и отладка

5.1. Проведение модульного тестирования

5.2. Отладка программы

6. Разработка мобильной версии

7. Разработка технической документации

7.1. Руководство пользователя

7.2. Руководство программиста

7.3. Руководство системного программиста

7.4. Методика тестирования и испытания программы

Заключение

Список литературы