Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Пояснительная записка к курсовой работе**

**по дисциплине «Современные технологии программирования»**

**на тему:**

**«Экзаменационная система»**

Выполнил:

студент группы ПИ19-1в

Заваруев Иван Сергеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Научный руководитель:

Варнавский Александр Николаевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Москва 2021

ОГЛАВЛЕНИЕ

[1ВВЕДЕНИЕ 3](#_Toc72420428)

[2 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc72420429)

[3 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ 6](#_Toc72420430)

[3 ОПИСАНИЕ ПРОГРАММЫ 7](#_Toc72420431)

[3.1 База данных 7](#_Toc72420432)

[3.2 Сервер 8](#_Toc72420433)

[3.3 Интерфейс 11](#_Toc72420434)

[4 ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ 13](#_Toc72420435)

[4.1 Главная страница 13](#_Toc72420436)

[4.2 Результаты студента 13](#_Toc72420437)

[4.3 Добавление нового студента 14](#_Toc72420438)

[4. 4 Редактирование информации о студенте 14](#_Toc72420439)

[4.5 Список результатов экзаменов 15](#_Toc72420440)

[5 НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ 16](#_Toc72420441)

[6 ЗАКЛЮЧЕНИЕ 18](#_Toc72420442)

[7 СПИСОК ЛИТЕРАТУРЫ 19](#_Toc72420443)

1ВВЕДЕНИЕ

Цифровизация шагает по планете, не отстает от современных тенденций и Финансовый Университет, в качестве курсовой работы предлагающий студентам второго курса разработать целый ворох информационно-справочных систем, призванных заменить собой традиционные аналоговые методы записи, хранения, передачи и предоставления информации. Такой выбор выглядит как нельзя лучше в нынешних условиях стабильно растущей потребности мировой экономики в квалифицированных кадрах IT-профиля, готовых трудиться на благо человечества в целом и себя лично.

Выбор автора этого труда пал на тему, возглавлявшую сводный список своих сестер – “Экзаменационная система”. В свете вышеизложенных соображений не поддается переоценке в шумном сплетении человеческих занятий благородная роль просвещения, обучения и образования. Испокон веков стремились люди распространять и преумножать в себе навыки и умения лучших своих представителей, жажда знаний толкала человека к открытиям и изобретениям, а ремесло учителя за сотни веков пропиталось не только кровавым потом ежедневного труда тела и разума, но и радостью воспитания, новых поколений, с гордостью несущих неугасимый свет просвещения сквозь страницы истории.

Именно высоким целям образования и призвана служить описываемая на страницах этой записки цифровая система, занимая скромное, но вместе с тем почетное место информационного ресурса, обеспечивающего своевременное информирование обучающихся о результатах проходимых ими промежуточных и итоговых испытаний.

2 ПОСТАНОВКА ЗАДАЧИ

Разработка программного обеспечения неизбежно начинается с постановки задачи, определения функционала будущего продукта, его свойств и особенностей, распределения ресурсов и выработки стратегии последующей работы.

Ввиду выявленной путем многочисленных попыток изучения оного необъятности объема человеческого знания, прямо или косвенно необходимого для построения системы, одновременно отвечающей требованиям и рекомендациям образовательного учреждения, инициировавшего процесс появления на свет описываемого объекта интеллектуального труда, и удовлетворяющей потребности автора если и не сохранять абсолютный контроль над своим детищем, то по крайней мере свободно ориентироваться в его внутреннем устройстве, было принято решение урезать функциональную и визуальную составляющие до уровня, позволяющего с минимально возможными затратами неумолимо истощающихся в процессе взаимодействия с упомянутым объемом запасов времени и нервов автора преодолеть необходимый порог признания работы достойной занять свое скромное место на полках архива в ряду себе подобных и не послужить причиной выдворения его создателя из стен обучающего его учреждения.

В результате проведенного анализа и принятия во внимание вышеперечисленные обстоятельства, был выработан следующий список задач и требований, определивший итоговый вид разрабатываемой системы:

* Программа должна представлять собой информационную систему, главная задача которой состоит в предоставлении прошедшему процедуру экзамена обучающемуся сведений о результатах испытания.
* Архитектурно система будет состоять из трех сообщающихся слоёв:
  + база данных, хранящая и предоставляющая информацию,
  + слой интерфейса, обеспечивающий взаимодействие с системой пользователей,
  + управляющий слой, обеспечивающий корректное взаимодействие слоёв предыдущих.
* Система должна обеспечивать пользователю возможность оперативно добавить в себя новую информацию, получить редактировать, а при необходимости и удалить уже имеющуюся.

3 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ

Каждый, кому приходилось сдавать важный, возможно определяющий ход дальнейшей жизни экзамен, помнит то волнение, с которым приходится дожидаться объявления результатов пройденного испытания. Ни малейших сомнений не вызывает необходимость вовлечения в процесс оповещения цифровой информационной системы, способной в полной мере, круглосуточно и неустанно обслуживать информационные потребности проэкзаменованных граждан.

Информационная система, задействуя поражающие воображение вычислительные мощности современных электронных вычислительных машин способна заменять собой неисчислимые множества операторов телефонных линий, сотрудников приемных комиссий и просто преподавателей, позволяя им пустить сэкономленное время в русло творческого труда или заслуженного отдыха.

Беспристрастная машина, хотя и не будучи в состоянии с чуткостью живого человека подбодрить провалившегося студента или разделить радость с его более трудолюбивым или удачливым однокашником, с другой стороны не может быть подвергнута моральному давлению или физическому воздействию, что несомненно благотворно сказывается на поддержании и развитии здорового климата честных и непредвзятых отношений преподающих и обучающихся в образовательной среде.

3 ОПИСАНИЕ ПРОГРАММЫ

3.1 База данных

В качестве СУБД задействована реляционная БД PostgreSQL, которая хранит данные в виде набора таблиц. Доступ к ней напрямую будет осуществляться лишь сервером (см. Рисунок 1).

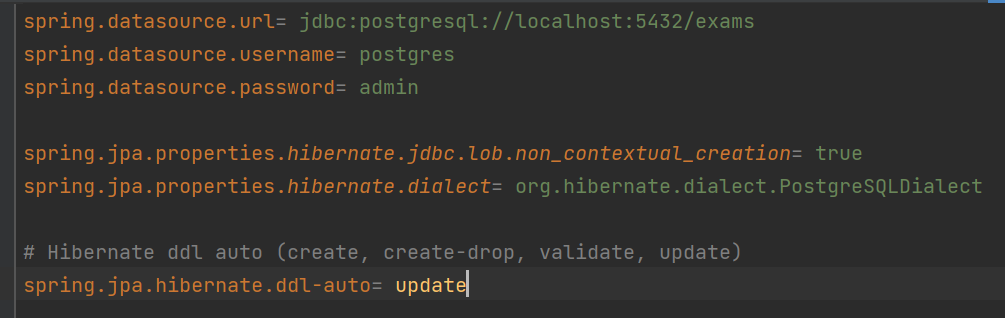


Рисунок 1 - Конфигурация подключения к базе данных

На основе пользовательских классов (моделей) в рамках предметной области объявлены следующие сущности с соответствующими им полями:

1. Students – таблица всех студентов (см. Рисунок 2):
   * id – уникальный идентификатор студента;
   * firstname – имя;
   * lastname – фамилия;

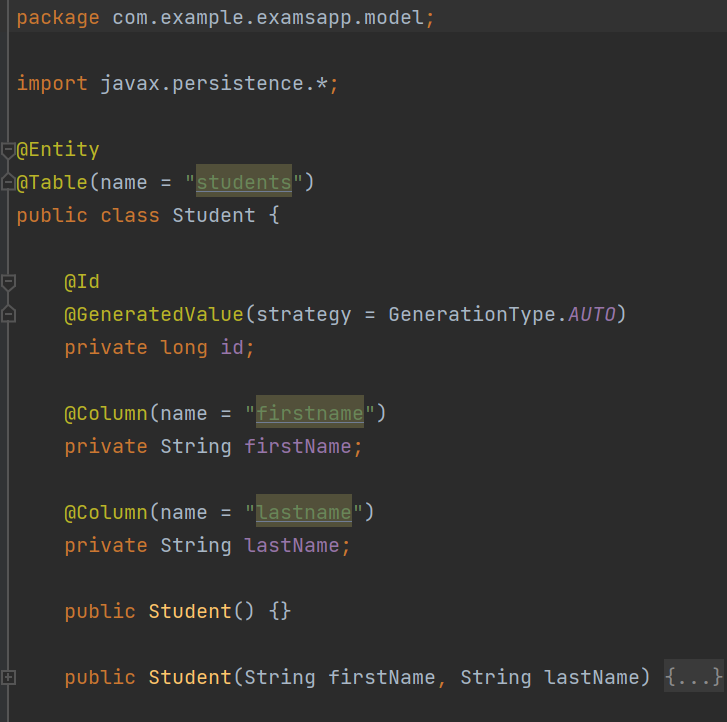


Рисунок 2 - Класс Student

1. Exams – таблица всех результатов экзаменов (см. Рисунок 3):
   * id – уникальный идентификатор результата экзамена;
   * student\_id – уникальный идентификатор студента, экзамен сдававшего.
   * subject – дисциплина экзамена
   * score – оценка студента за экзамен

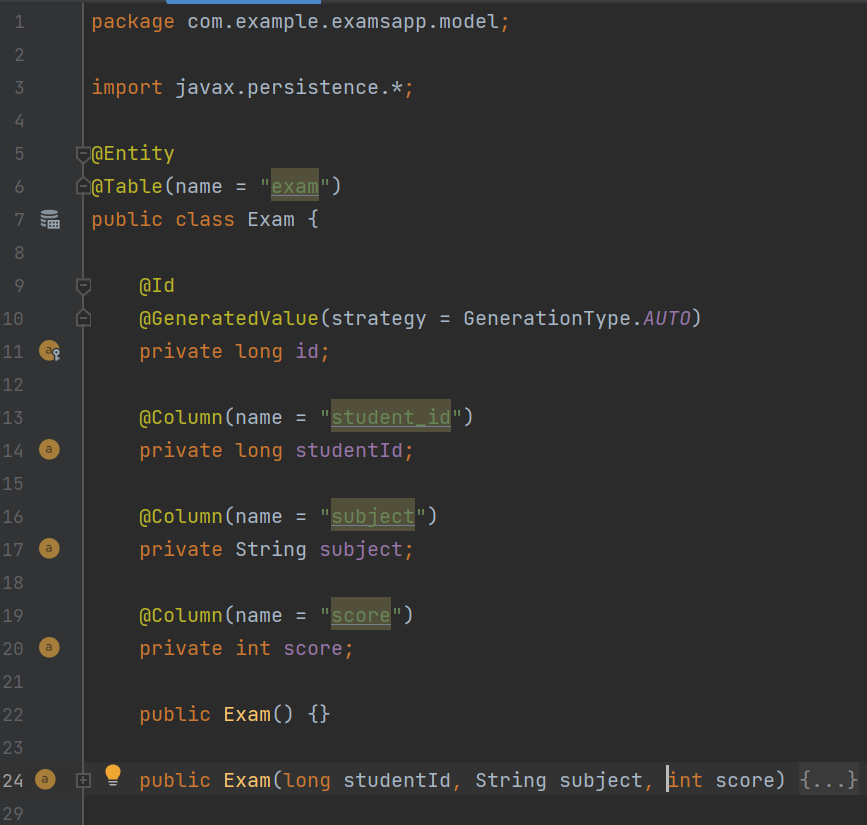


Рисунок 3 - Класс Exam

3.2 Сервер

Серверная сторона приложения реализована с использованием библиотеки языка Java под названием Spring Boot. Для создания простого web-приложения с учетом использования базы данных PostgreSQL, были использованы следующие основные технологии:

1. Spring web – ядро для разрабатываемого web-приложения, в своей основе подключающий локальный сервер Apache Tomcat;
2. Spring boot maven plugin – плагин для сборки проекта;
3. Spring Data JPA – технология, предоставляющая интерфейс для управления, сохранения и получения сущностей при работе с БД, и превращения описанных Java классов в объектно-реляционный вид;

Приложение состоит из обычных Java классов с использованием специфических аннотаций для реализации архитектуры Модель-Представление-Контроллер, но с добавлением слоя так называемых Репозиториев, упрощающих взаимодействие с базой данных. Схема взаимодействия всех компонентов приложения представлена на Рисунке 4.

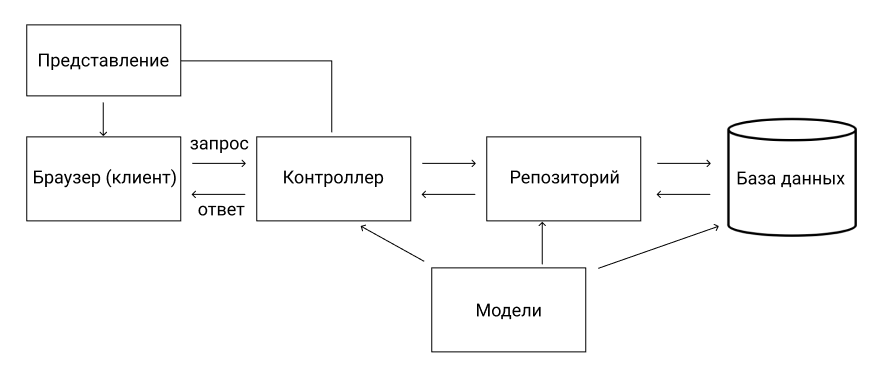


Рисунок 4 - Схема компонентов

Стоит подробнее рассмотреть задачи каждого функционального блока.

1. Контроллер отвечает за логику обработки HTTP запросов – GET, POST, PUT, DELETE соответственно. Когда клиент отправляет запрос, он в первую очередь попадает в контроллер. В зависимости от того, какой HTTP запрос должен прийти в этот метод контроллера, отрабатывает тот или иной mapping, связывающий соответствующий метод с URL-адресом (см. Рисунок 5).



Рисунок 5 - Фрагмент класса StudentController

1. Затем контроллер обращается к репозиторию – интерфейсу, расширяющему JpaRepository, и предоставляющего в пользование удобные методы для основных операций с данными в БД – создание, чтение, обновление, удаление (см. Рисунок 6).

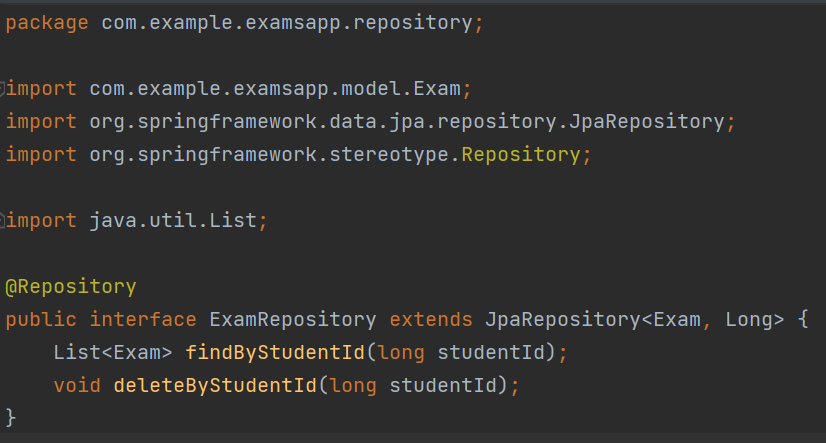


Рисунок 6 - ­Интерфейс ExamRepository

1. Каждой модели соответствует своя таблица в БД. Технология Hibernate, в качестве реализации интерфейса JPA, позволяет не писать запросы к БД вручную, а автоматически создавать таблицы в ней на основе наших описанных Java классов. Модель обменивается с репозиторием данными, а тот отдает их обратно контроллеру.
2. Далее контроллер отдает полученные данные блоку представления. Данные переходят клиенту (в случае веб-страницы – браузеру), и тот на их основе показывает актуальные данные в пользовательском интерфейсе.

3.3 Интерфейс

Слой интерфейса внешний – он обеспечивает взаимодействие системы с пользователем. Здесь с помощью механизма Java-шаблонов Thymeleaf генерируются HTML-страницы (см. Рисунок 7), которые пользователю и показываются.

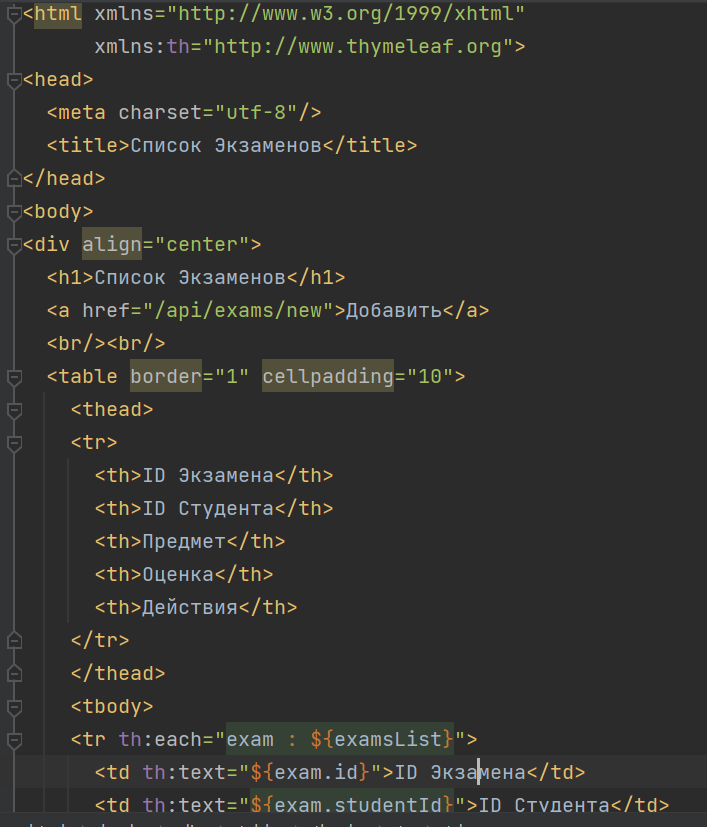


Рисунок 7 - Фрагмент файла страницы списка экзаменов

Клиентская часть описывает логику взаимодействия и отображения следующих страниц (см. Таблица 1).

Таблица 1 - Список HTML-страниц

|  |  |  |
| --- | --- | --- |
| Название страницы | Адрес | Функции |
| Студенты | /api/students | Отображает таблицу всех студентов |
| Студент | /api/students/{id} | Отображает таблицу результатов экзаменов выбранного студента |
| Редактировать Студента | /api/students/edit/{id} | Отображает форму для редактирования данных выбранного студента |
| Добавить Студента | /api/students/new | Отображает форму добавления в базу данных нового студента |
| Список Экзаменов | /api/exams | Отображает таблицу всех результатов экзаменов |
| Редактировать Экзамен | /api/exams/edit/{id} | Отображает форму для редактирования данных выбранного экзамена |
| Добавить Экзамен | /api/exams/new | Отображает форму добавления в базу нового результата экзамена |

4 ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ

Интерфейс программы представлен семью страницами:

## 4.1 Главная страница

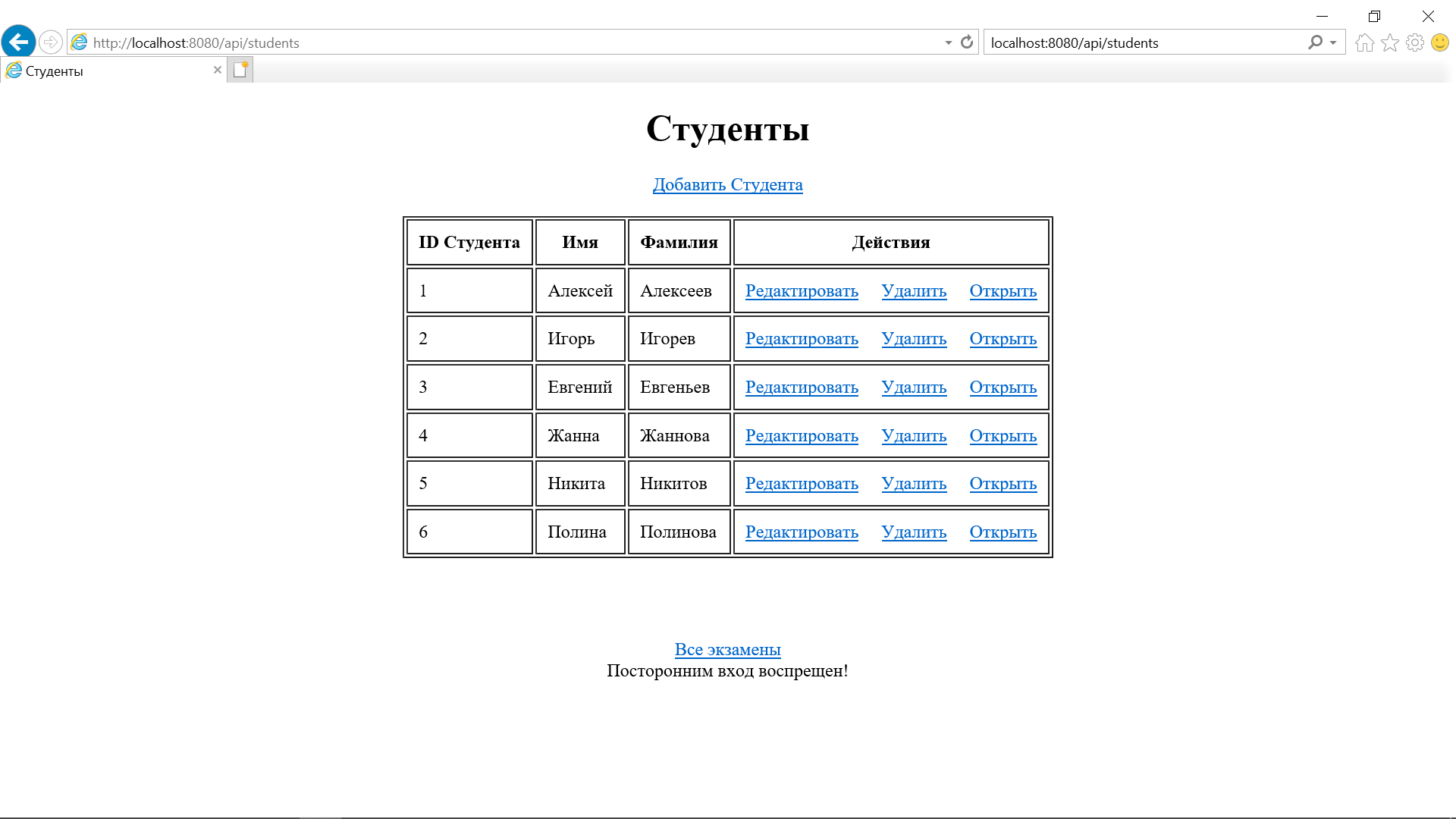


Рисунок 8 - Главная страница

Главная страница (см. Рисунок 8) представляет из себя список всех имеющихся в базе данных студентов. Будучи минималистичной до аскетизма, она, тем не менее, содержит интуитивно понятные элементы управления, позволяющие редактировать, добавлять и удалять данные о студентах. В нижней части страницы нашла свое место ссылка на список результатов экзаменов. Чрезвычайно высокий средний уровень внутренней культуры пользователя системы позволил реализовать механизм защиты списка и распределения ролей на морально-этических принципах сознательности и ответственности.

## 4.2 Результаты студента

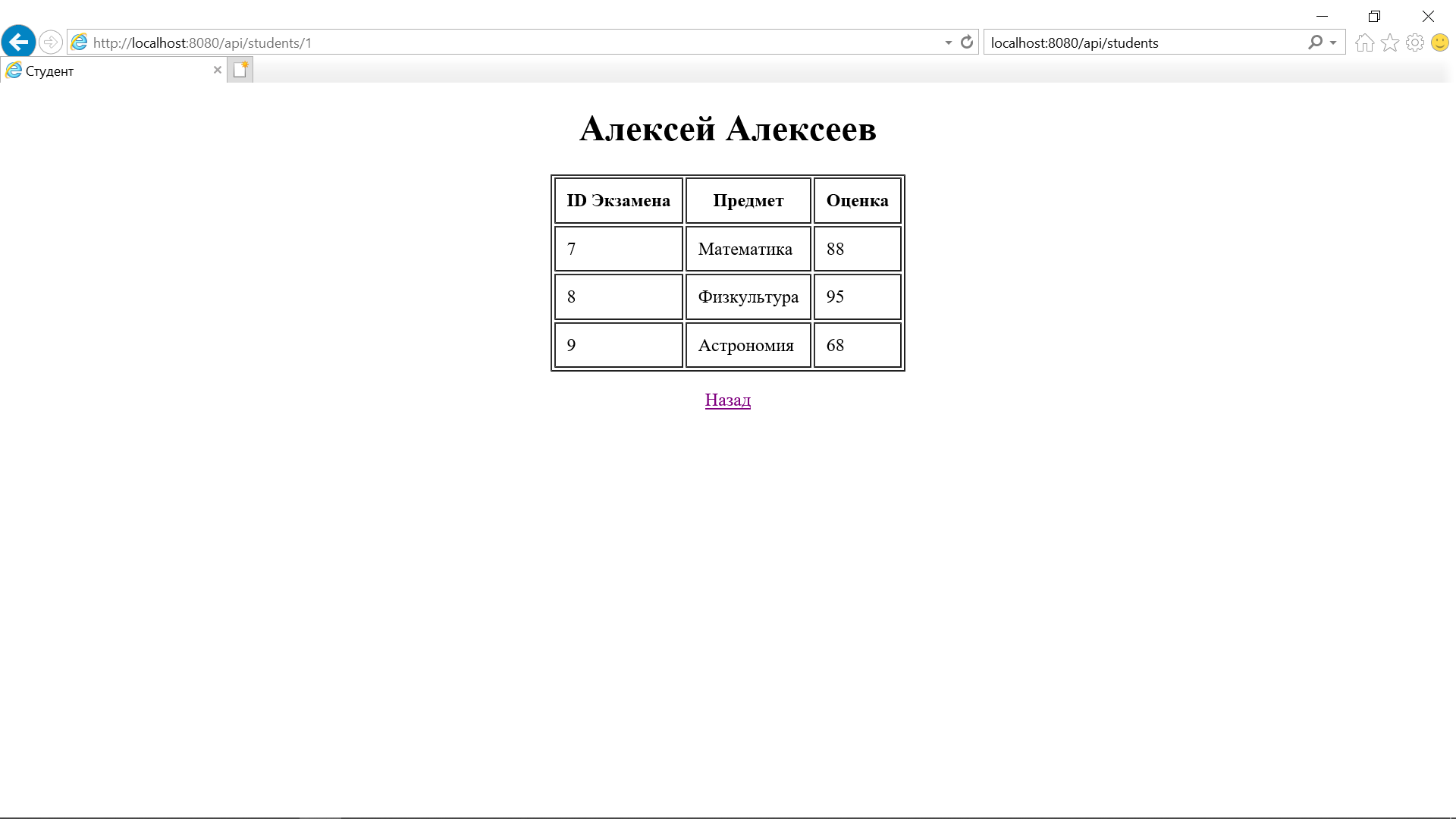


Рисунок 9 - Результаты студента

При нажатии на слово «Открыть» напротив нужного студента пользователь попадет на страницу, выводящую все результаты экзаменов, относящиеся к этому студенту (см. Рисунок 9).

## 4.3 Добавление нового студента

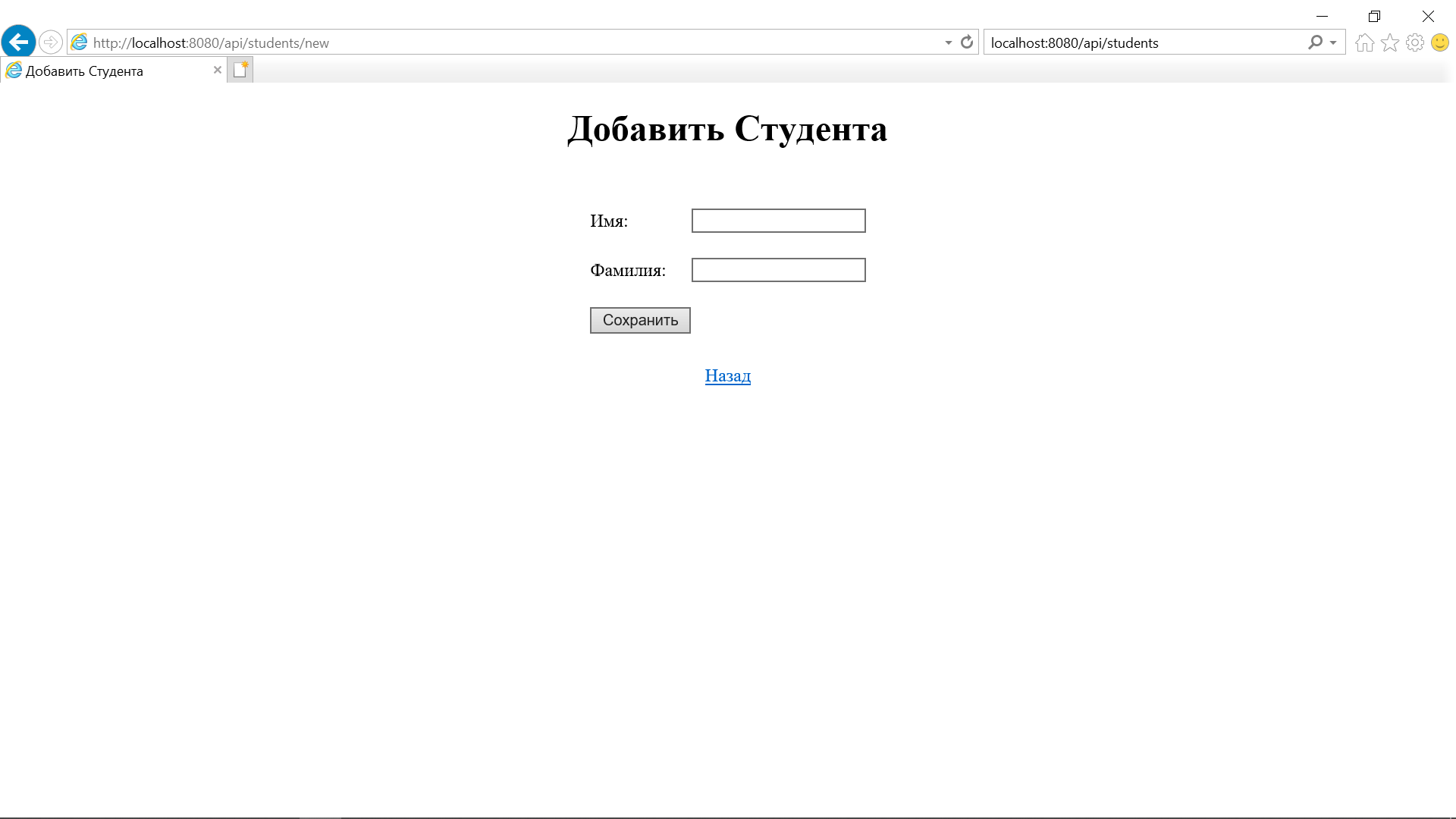
­­­

Рисунок 10 - Добавление нового студента

Нажатие на ссылку «Добавить Студента» перенаправляет на не менее изящную страницу добавления, где нехитрым путем ввода имени и фамилии можно создать в базе данных запись о новом студенте. Поле уникального идентификатора сгенерируется автоматически (см. Рисунок 10).

## 4. 4 Редактирование информации о студенте

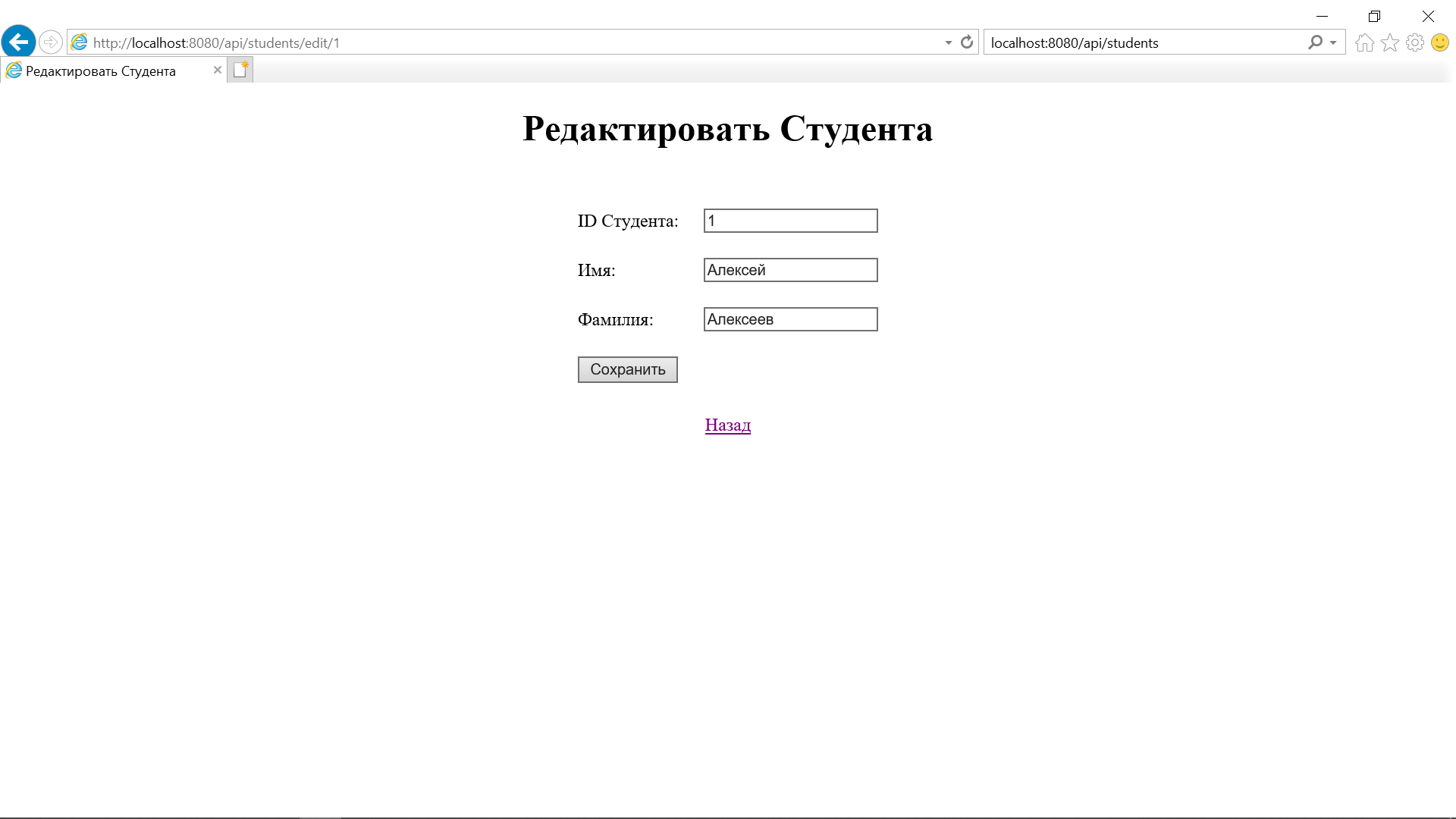


Рисунок 11 - Редактирование информации о студенте

На это странице можно поменять студенту имя и/или фамилию в базе данных. Поле «ID Студента» неактивное (см. Рисунок 11).

## 4.5 Список результатов экзаменов

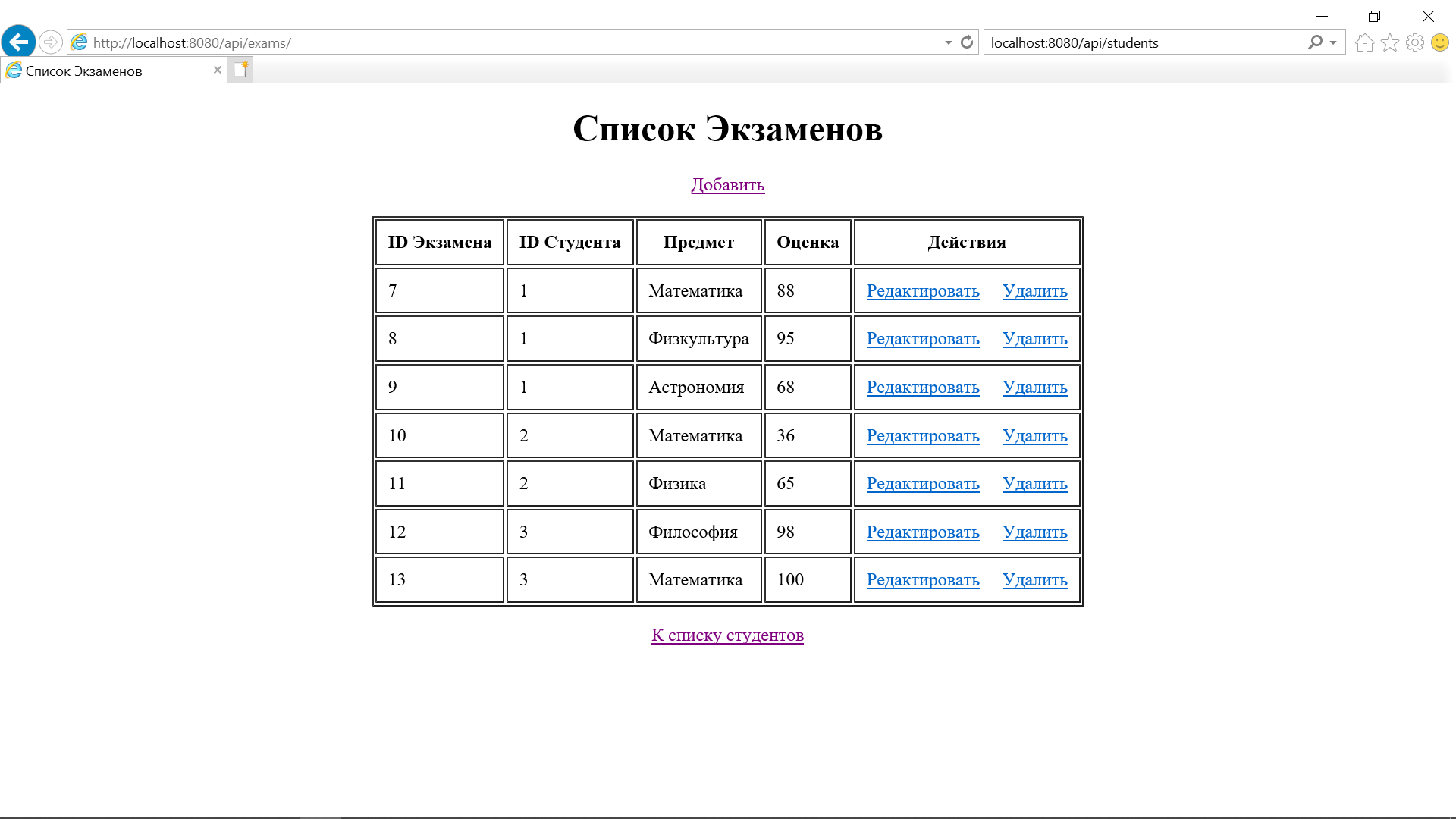


Рисунок 12 - Список результатов экзаменов

Страница выводит на экран список всех известных базе данных результатов экзаменов. На то, что обычный посетитель забрел слишком далеко указывает даже поле «ID Студента» вместо имени и фамилии. Экзамены можно добавлять, удалять и редактировать так же, как и студентов (см. Рисунок 12).

5 НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

Для корректной работы приложения были реализованы следующие функциональные блоки классов:

* Контроллеры (2 шт.) – классы для обработки клиентских запросов;
* Модели (2 шт.) – классы, описывающие сущности предметной области и их атрибуты;
* Репозитории (2 шт.) – интерфейсы, предоставляющие методы для основных операций с базой данных;

Назначение каждого класса программы следующее:

1. StudentController – отвечает за запросы, касающиеся отображения и других операций с таблицей студентов.
2. ExamController – отвечает за запросы, касающиеся отображения и других операций с таблицей экзаменов.
3. StudentRepository – интерфейс JpaRepository для упрощения операций с таблицей студентов.
4. ExamRepository – интерфейс JpaRepository для упрощения операций с таблицей экзаменов.
5. Student – сущность, представляющая таблицу студентов.
6. Exam – сущность, представляющая таблицу экзаменов.

Связь классов между собой показана на Рисунке 13.

Graphical user interface

Description automatically generated

Рисунок 13 - Взаимные отношения классов

6 ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана информационно-справочная экзаменационная система, позволяющая экзаменатору эффективно и оперативно предоставлять экзаменованным информацию о результатах их экзаменов.

Трехслойная архитектура проекта, состоящая из базы данных, управляющего и интерфейсного блоков, удобна в разработке и поддержке, открыта к поистине бесконечному совершенствованию. Пользователь, в свою очередь, получает простой и понятный интерфейс, удовлетворяющий его потребности в получении интересующей информации. Таким образом, спроектированное решение удовлетворяет поставленным требованиям и сформулированной задаче курсовой работы.

7 СПИСОК ЛИТЕРАТУРЫ

1. Webb P. et al. Spring boot reference guide //Part IV. Spring Boot features. – 2013. – Т. 24.
2. Gómez O. S., Rosero R. H., Cortés-Verdín K. CRUDyLeaf: a DSL for generating spring boot REST APIs from entity CRUD operations //Cybernetics and Information Technologies. – 2020. – Т. 20. – №. 3. – С. 3-14..
3. Dewailly L. Building a RESTful web service with spring. – Packt Publishing Ltd, 2015.

# ПРИЛОЖЕНИЕ А

## Исходный код программы

pom.xml

*<?xml version="1.0" encoding="UTF-8"?>*

*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">*

*<modelVersion>4.0.0</modelVersion>*

*<parent>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-parent</artifactId>*

*<version>2.4.5</version>*

*<relativePath/> <!-- lookup parent from repository -->*

*</parent>*

*<groupId>com.example</groupId>*

*<artifactId>exams-app</artifactId>*

*<version>0.0.1-SNAPSHOT</version>*

*<name>exams-app</name>*

*<description>Demo project for Spring Boot</description>*

*<properties>*

*<java.version>11</java.version>*

*</properties>*

*<dependencies>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-data-jpa</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-web</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>org.postgresql</groupId>*

*<artifactId>postgresql</artifactId>*

*<scope>runtime</scope>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-test</artifactId>*

*<scope>test</scope>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-thymeleaf</artifactId>*

*</dependency>*

*</dependencies>*

*<build>*

*<plugins>*

*<plugin>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-maven-plugin</artifactId>*

*</plugin>*

*</plugins>*

*</build>*

*</project>*

application.properties

*spring.datasource.url= jdbc:postgresql://localhost:5432/exams*

*spring.datasource.username= postgres*

*spring.datasource.password= admin*

*spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation= true*

*spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect*

*# Hibernate ddl auto (create, create-drop, validate, update)*

*spring.jpa.hibernate.ddl-auto= update*

ExamController

*package com.example.examsapp.controller;*

*import com.example.examsapp.model.Exam;*

*import com.example.examsapp.repository.ExamRepository;*

*import org.springframework.beans.factory.annotation.Autowired;*

*import org.springframework.stereotype.Controller;*

*import org.springframework.ui.Model;*

*import org.springframework.web.bind.annotation.\*;*

*import org.springframework.web.servlet.ModelAndView;*

*import java.util.List;*

*import java.util.Optional;*

*@Controller*

*@RequestMapping("/api/exams")*

*public class ExamController {*

*@Autowired*

*ExamRepository examRepository;*

*@GetMapping("")*

*public String getAllExams(Model model) {*

*List<Exam> examsList = examRepository.findAll();*

*model.addAttribute("examsList", examsList);*

*return "exams";*

*}*

*@RequestMapping("/edit/{id}")*

*public ModelAndView showEditExamPage(@PathVariable(name = "id") Long id) {*

*ModelAndView mav = new ModelAndView("edit\_exam");*

*Exam exam = examRepository.getOne(id);*

*mav.addObject("exam", exam);*

*return mav;*

*}*

*@RequestMapping("/new")*

*public String showNewExamForm(Model model){*

*Exam exam = new Exam();*

*model.addAttribute("exam",exam);*

*return "new\_exam";*

*}*

*@RequestMapping(value = "/save", method = RequestMethod.POST)*

*public String saveExam(@ModelAttribute("exam") Exam exam){*

*examRepository.save(exam);*

*return "redirect:/api/exams";*

*}*

*@RequestMapping("/delete/{id}")*

*public String deleteExam(@PathVariable(name = "id") long id) {*

*examRepository.deleteById(id);*

*return "redirect:/api/exams";*

*}*

*}*

StudentController

*package com.example.examsapp.controller;*

*import com.example.examsapp.model.Exam;*

*import com.example.examsapp.model.Student;*

*import com.example.examsapp.repository.ExamRepository;*

*import com.example.examsapp.repository.StudentRepository;*

*import org.springframework.beans.factory.annotation.Autowired;*

*import org.springframework.stereotype.Controller;*

*import org.springframework.ui.Model;*

*import org.springframework.web.bind.annotation.\*;*

*import org.springframework.web.servlet.ModelAndView;*

*import javax.transaction.Transactional;*

*import java.util.\*;*

*@Controller*

*@RequestMapping("/api/students")*

*public class StudentController {*

*@Autowired*

*StudentRepository studentRepository;*

*@Autowired*

*ExamRepository examRepository;*

*@GetMapping("")*

*public String getAllStudents(Model model) {*

*List<Student> studentsList = studentRepository.findAll();*

*model.addAttribute("studentsList", studentsList);*

*return "index";*

*}*

*@GetMapping("/{id}")*

*public String getStudentById(@PathVariable("id") long id, Model model) {*

*Optional<Student> studentData = studentRepository.findById(id);*

*if (studentData.isPresent()) {*

*List<Exam> exams = new ArrayList<Exam>();*

*examRepository.findByStudentId(id).forEach(exams::add);*

*model.addAttribute("firstName", studentData.get().getFirstName());*

*model.addAttribute("lastName", studentData.get().getLastName());*

*model.addAttribute("exams", exams);*

*return "student";*

*} else {*

*return null;*

*}*

*}*

*@RequestMapping("/edit/{id}")*

*public ModelAndView showEditStudentPage(@PathVariable(name = "id") Long id) {*

*ModelAndView mav = new ModelAndView("edit\_student");*

*Optional<Student> student = studentRepository.findById(id);*

*mav.addObject("student", student);*

*return mav;*

*}*

*@RequestMapping("/new")*

*public String showNewStudentForm(Model model){*

*Student student = new Student();*

*model.addAttribute("student",student);*

*return "new\_student";*

*}*

*@RequestMapping(value = "/save", method = RequestMethod.POST)*

*public String saveStudent(@ModelAttribute("student") Student student){*

*studentRepository.save(student);*

*return "redirect:/api/students";*

*}*

*@Transactional*

*@RequestMapping("/delete/{id}")*

*public String deleteStudent(@PathVariable(name = "id") long id) {*

*studentRepository.deleteById(id);*

*examRepository.deleteByStudentId(id);*

*return "redirect:/api/students";*

*}*

*}*

Exam

*package com.example.examsapp.model;*

*import javax.persistence.\*;*

*@Entity*

*@Table(name = "exam")*

*public class Exam {*

*@Id*

*@GeneratedValue(strategy = GenerationType.AUTO)*

*private long id;*

*@Column(name = "student\_id")*

*private long studentId;*

*@Column(name = "subject")*

*private String subject;*

*@Column(name = "score")*

*private int score;*

*public Exam() {}*

*public Exam(long studentId, String subject, int score) {*

*this.studentId = studentId;*

*this.subject = subject;*

*this.score = score;*

*}*

*public long getId() {*

*return id;*

*}*

*public void setId(long id) {*

*this.id = id;*

*}*

*public long getStudentId() {*

*return studentId;*

*}*

*public void setStudentId(long studentId) {*

*this.studentId = studentId;*

*}*

*public String getSubject() {*

*return subject;*

*}*

*public void setSubject(String subject) {*

*this.subject = subject;*

*}*

*public int getScore() {*

*return score;*

*}*

*public void setScore(int score) {*

*this.score = score;*

*}*

*}*

Student

*package com.example.examsapp.model;*

*import javax.persistence.\*;*

*@Entity*

*@Table(name = "students")*

*public class Student {*

*@Id*

*@GeneratedValue(strategy = GenerationType.AUTO)*

*private long id;*

*@Column(name = "firstname")*

*private String firstName;*

*@Column(name = "lastname")*

*private String lastName;*

*public Student() {}*

*public Student(String firstName, String lastName) {*

*this.firstName = firstName;*

*this.lastName = lastName;*

*}*

*public long getId() {*

*return id;*

*}*

*public void setId(long id) {*

*this.id = id;*

*}*

*public String getFirstName() {*

*return firstName;*

*}*

*public void setFirstName(String firstName) {*

*this.firstName = firstName;*

*}*

*public String getLastName() {*

*return lastName;*

*}*

*public void setLastName(String lastName) {*

*this.lastName = lastName;*

*}*

*@Override*

*public String toString() {*

*return "Student [id=" + id + ", firstName=" + ", lastName=" + lastName + "]";*

*}*

*}*

ExamRepository

*package com.example.examsapp.repository;*

*import com.example.examsapp.model.Exam;*

*import org.springframework.data.jpa.repository.JpaRepository;*

*import org.springframework.stereotype.Repository;*

*import java.util.List;*

*@Repository*

*public interface ExamRepository extends JpaRepository<Exam, Long> {*

*List<Exam> findByStudentId(long studentId);*

*void deleteByStudentId(long studentId);*

*}*

StudentRepository

*package com.example.examsapp.repository;*

*import com.example.examsapp.model.Student;*

*import org.springframework.data.jpa.repository.JpaRepository;*

*public interface StudentRepository extends JpaRepository<Student, Long> {*

*}*

ExamsAppApplication

*package com.example.examsapp;*

*import org.springframework.boot.SpringApplication;*

*import org.springframework.boot.autoconfigure.SpringBootApplication;*

*@SpringBootApplication*

*public class ExamsAppApplication {*

*public static void main(String[] args) {*

*SpringApplication.run(ExamsAppApplication.class, args);*

*}*

*}*

edit\_exam.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8" />*

*<title>Редактировать Экзамен</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Редактировать Экзамен</h1>*

*<br />*

*<form action="#" th:action="@{/api/exams/save}" th:object="${exam}" method="post">*

*<table border="0" cellpadding="10">*

*<tr>*

*<td>ID Экзамена:</td>*

*<td>*

*<input type="number" th:field="\*{id}" readonly="readonly" />*

*</td>*

*</tr>*

*<tr>*

*<td>ID Студента:</td>*

*<td>*

*<input type="number" th:field="\*{studentId}" />*

*</td>*

*</tr>*

*<tr>*

*<td>Предмет:</td>*

*<td><input type="text" th:field="\*{subject}" /></td>*

*</tr>*

*<tr>*

*<td>Оценка:</td>*

*<td><input type="number" th:field="\*{score}" /></td>*

*</tr>*

*<tr>*

*<td colspan="2"><button type="submit">Save</button> </td>*

*</tr>*

*</table>*

*</form>*

*<p><a href="/api/exams/">Назад</a></p>*

*</div>*

*</body>*

*</html>*

edit\_student.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8" />*

*<title>Редактировать Студента</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Редактировать Студента</h1>*

*<br />*

*<form action="#" th:action="@{/api/students/save}" th:object="${student}" method="post">*

*<table border="0" cellpadding="10">*

*<tr>*

*<td>ID Студента:</td>*

*<td>*

*<input type="number" th:field="\*{id}" readonly="readonly" />*

*</td>*

*</tr>*

*<tr>*

*<td>Имя:</td>*

*<td>*

*<input type="text" th:field="\*{firstName}" />*

*</td>*

*</tr>*

*<tr>*

*<td>Фамилия:</td>*

*<td><input type="text" th:field="\*{lastName}" /></td>*

*</tr>*

*<tr>*

*<td colspan="2"><button type="submit">Сохранить</button> </td>*

*</tr>*

*</table>*

*</form>*

*<p><a href="/api/students/">Назад</a></p>*

*</div>*

*</body>*

*</html>*

exams.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8"/>*

*<title>Список Экзаменов</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Список Экзаменов</h1>*

*<a href="/api/exams/new">Добавить</a>*

*<br/><br/>*

*<table border="1" cellpadding="10">*

*<thead>*

*<tr>*

*<th>ID Экзамена</th>*

*<th>ID Студента</th>*

*<th>Предмет</th>*

*<th>Оценка</th>*

*<th>Действия</th>*

*</tr>*

*</thead>*

*<tbody>*

*<tr th:each="exam : ${examsList}">*

*<td th:text="${exam.id}">ID Экзамена</td>*

*<td th:text="${exam.studentId}">ID Студента</td>*

*<td th:text="${exam.subject}">Предмет</td>*

*<td th:text="${exam.score}">Оценка</td>*

*<td>*

*<a th:href="@{'/api/exams/edit/' + ${exam.id}}">Редактировать</a>*

*&nbsp;&nbsp;&nbsp;*

*<a th:href="@{'/api/exams/delete/' + ${exam.id}}">Удалить</a>*

*</td>*

*</tr>*

*</tbody>*

*</table>*

*<p><a href="/api/students/">К списку студентов</a></p>*

*</div>*

*</body>*

*</html>*

index.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8"/>*

*<title>Студенты</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Студенты</h1>*

*<a href="/api/students/new">Добавить Студента</a>*

*<br/><br/>*

*<table border="1" cellpadding="10">*

*<thead>*

*<tr>*

*<th>ID Студента</th>*

*<th>Имя</th>*

*<th>Фамилия</th>*

*<th>Действия</th>*

*</tr>*

*</thead>*

*<tbody>*

*<tr th:each="student : ${studentsList}">*

*<td th:text="${student.id}">Student ID</td>*

*<td th:text="${student.firstName}">First Name</td>*

*<td th:text="${student.lastName}">Last Name</td>*

*<td>*

*<a th:href="@{'/api/students/edit/' + ${student.id}}">Редактировать</a>*

*&nbsp;&nbsp;&nbsp;*

*<a th:href="@{'/api/students/delete/' + ${student.id}}">Удалить</a>*

*&nbsp;&nbsp;&nbsp;*

*<a th:href="@{'/api/students/' + ${student.id}}">Открыть</a>*

*</td>*

*</tr>*

*</tbody>*

*</table>*

*<br/>*

*<br/>*

*<br/>*

*<div >*

*<p><a href="/api/exams/">Все экзамены</a><br>*

*Посторонним вход воспрещен!</p>*

*</div>*

*</div>*

*</body>*

*</html>*

new\_exam.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8" />*

*<title>Добавить Экзамен</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Добавить Экзамен</h1>*

*<br />*

*<form action="#" th:action="@{/api/exams/save}" th:object="${exam}"*

*method="post">*

*<table border="0" cellpadding="10">*

*<tr>*

*<td>ID Студента:</td>*

*<td><input type="number" th:field="\*{studentId}" /></td>*

*</tr>*

*<tr>*

*<td>Предмет:</td>*

*<td><input type="text" th:field="\*{subject}" /></td>*

*</tr>*

*<tr>*

*<td>Оценка:</td>*

*<td><input type="number" th:field="\*{score}" /></td>*

*</tr>*

*<tr>*

*<td colspan="2"><button type="submit">Сохранить</button> </td>*

*</tr>*

*</table>*

*</form>*

*<p><a href="/api/exams/">Назад</a></p>*

*</div>*

*</body>*

*</html>*

new\_student.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8" />*

*<title>Добавить Студента</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1>Добавить Студента</h1>*

*<br />*

*<form action="#" th:action="@{/api/students/save}" th:object="${student}"*

*method="post">*

*<table border="0" cellpadding="10">*

*<tr>*

*<td>Имя:</td>*

*<td><input type="text" th:field="\*{firstName}" /></td>*

*</tr>*

*<tr>*

*<td>Фамилия:</td>*

*<td><input type="text" th:field="\*{lastName}" /></td>*

*</tr>*

*<tr>*

*<td colspan="2"><button type="submit">Сохранить</button> </td>*

*</tr>*

*</table>*

*</form>*

*<p><a href="/api/students/">Назад</a></p>*

*</div>*

*</body>*

*</html>*

student.html

*<!DOCTYPE html>*

*<html xmlns="http://www.w3.org/1999/xhtml"*

*xmlns:th="http://www.thymeleaf.org">*

*<head>*

*<meta charset="utf-8"/>*

*<title>Студент</title>*

*</head>*

*<body>*

*<div align="center">*

*<h1 th:text="${firstName} + ' ' + ${lastName}"></h1>*

*<table border="1" cellpadding="10">*

*<thead>*

*<tr>*

*<th>ID Экзамена</th>*

*<th>Предмет</th>*

*<th>Оценка</th>*

*</tr>*

*</thead>*

*<tbody>*

*<tr th:each="exam : ${exams}">*

*<td th:text="${exam.id}">Student ID</td>*

*<td th:text="${exam.subject}">First Name</td>*

*<td th:text="${exam.score}">Last Name</td>*

*</tr>*

*</tbody>*

*</table>*

*<p><a href="/api/students/">Назад</a></p>*

*</div>*

*</body>*

*</html>*