|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**КУРСОВАЯ РАБОТА**

по дисциплине: Разработка серверных частей интернет-ресурсов

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Серверная часть аддитивного микропортала по мониторингу пересдач для студентов

Студент: Латышев Иван Леонидович

Группа: ИКБО-16-21

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/ Латышев И. Л. /

(подпись и ф.и.о. студента)

Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/ Синицын А. В. /

(подпись и ф.и.о. рук-ля)

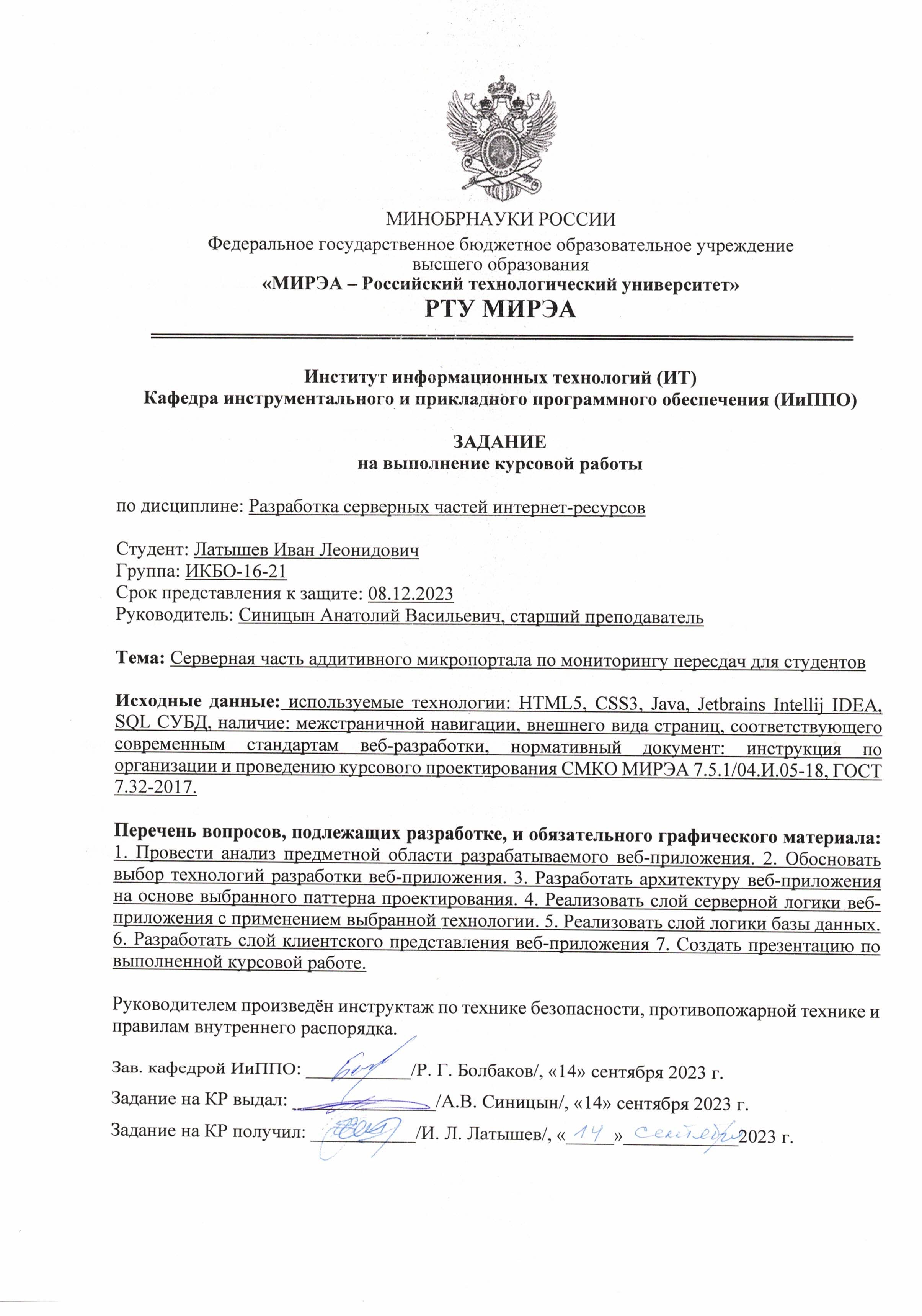
Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.



АННОТАЦИЯ

УДК 004.4

Руководитель курсовой работы: старший преподаватель А.В. Синицын.

Латышев И. Л., Курсовая работа направления подготовки «Программная инженерия» на тему «Серверная часть аддитивного микропортала по мониторингу пересдач для студентов»: М. 2023 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 47 стр., 20 рис., 20 источн. (в т.ч. 1 на английском яз.).

Ключевые слова: RESTful-приложение, пересдачи, студент, Spring Boot, Spring Security, JWT, MVC, React, RTK Query.

Целью работы является проектирование веб-приложения для портала отслеживания успеваемости студентов. Спроектирована информационная система, разработано решение для клиентской стороны, разработана база данных и настроен удаленный сервер.

Latyshev I. L., Course work in the field of training "Software Engineering" on the topic "Server part of an additive microportal for monitoring retakes for students": M. 2023, MIREA – Russian Technological University (RTU MIREA), Institute of Information Technology (IIT), Department of Instrumental and Applied Software (IiPPO) – 47 p., 20 fig., 20 sources. (including 1 in English).

Keywords: RESTful application, retakes, student, Spring Boot, Spring Security, JSP, MVC, React, RTK Query.

The purpose of the work is to design a web application for the student progress tracking portal. An information system has been designed, a solution for the client side has been developed, a database has been developed and a remote server has been configured.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ\_РСЧИР\_ИКБО-16-21\_ЛатышевИЛ.pdf», исполнитель Латышев И. Л.

© И.Л. Латышев

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СКОРАЩЕНИЙ 6](#_Toc152864266)

[ГЛОССАРИЙ 7](#_Toc152864267)

[ВВЕДЕНИЕ 8](#_Toc152864268)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc152864269)

[2 ВЫБОР ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЯ 11](#_Toc152864270)

[2.1 Основные используемые технологии 11](#_Toc152864271)

[3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ 13](#_Toc152864272)

[3.1 Описание архитектуры 13](#_Toc152864273)

[3.2 Реализация архитектуры в разрабатываемом проекте 14](#_Toc152864274)

[4 РАЗРАБОТКА БАЗЫ ДАННЫХ 16](#_Toc152864275)

[4.1 Определение сущностей 16](#_Toc152864276)

[4.1.1 Debt 16](#_Toc152864277)

[4.1.2 Group 17](#_Toc152864278)

[4.1.3 Performance 17](#_Toc152864279)

[4.1.4 Role 18](#_Toc152864280)

[4.1.5 Student 18](#_Toc152864281)

[4.1.6 Subject 19](#_Toc152864282)

[4.1.7 Teacher 19](#_Toc152864283)

[4.1.8 User 19](#_Toc152864284)

[4.2 Проектирование базы данных 20](#_Toc152864285)

[4.2.1 Описание дополнительных таблиц 20](#_Toc152864286)

[4.2.2 Создание ER-диаграммы 22](#_Toc152864287)

[5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ 22](#_Toc152864288)

[5.1 Структура проекта 22](#_Toc152864289)

[5.2 Конфигурация приложения 23](#_Toc152864290)

[5.3 Использование Spring Security для авторизации и аутентификации пользователей 24](#_Toc152864291)

[5.3.1 Класс «SpringConfig» 25](#_Toc152864292)

[5.3.2 Класс «JwtAuthentificationFilter» 26](#_Toc152864293)

[5.3.3 Класс «AuthentificationServiceImpl» 28](#_Toc152864294)

[5.3.4 Класс «JwtServiceImpl» 29](#_Toc152864295)

[5.4 Использование Spring Data JPA 30](#_Toc152864296)

[5.5 Классы, обеспечивающие работу с базой данных 31](#_Toc152864297)

[5.6 Разработка бизнес-логики приложения 32](#_Toc152864298)

[5.6.1 Класс «TeacherSerivce» 32](#_Toc152864299)

[5.6.2 Класс «UserDetailsService» 34](#_Toc152864300)

[5.7 Использование Spring MVC для реализации веб-интерфейса 35](#_Toc152864301)

[6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ 38](#_Toc152864302)

[6.1 Структура проекта 38](#_Toc152864303)

[6.2 Страницы разработанного приложения 38](#_Toc152864304)

[7 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ 42](#_Toc152864305)

[7.1 Тестирование приложения с помощью Insomnia 42](#_Toc152864306)

[ЗАКЛЮЧЕНИЕ 45](#_Toc152864307)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 46](#_Toc152864308)

ПЕРЕЧЕНЬ СКОРАЩЕНИЙ

|  |  |  |
| --- | --- | --- |
| RESTful API  СУБД | —  — | Representational State Transfer API  Система Управления Базами Данных |
| JWT  SQL  URL | —  —  — | JSON Web Token  Structured Query Language  Uniform Resource Locator |
|  |  |  |
|  |  |  |

ГЛОССАРИЙ

1. Серверная часть – часть приложения, которая занимается обработкой сложных данных, взаимодействует с базой данных и может лишь передавать клиенту данные в специальном установленном формате.
2. Клиентская часть - часть приложения, отвечающая за представление данных, полученных от сервера, пользователю. Также клиентская часть, иначе – «визуальная часть», позволяет взаимодействовать с пользователем, принимая от него какую-то информацию и передавая её на сервер.
3. Архитектура приложения - правила разработки приложения, которые часто позволяют упростить разработку, поддержку и расширение приложения.
4. База данных - хранилище с данными, которые разработчик предпочел хранить отдельно от клиентской части (пользователи, сотрудники и т.п.). Такое хранилище управляется посредством серверной части и СУБД.
5. СУБД - система, которая позволяет взаимодействовать с базой данных: добавлять, изменять, получать и удалять данные.
6. Микропортал – небольшой веб-сайт или лендинг, предназначенный для продвижения отдельного продукта, кампании или мероприятия.

ВВЕДЕНИЕ

В настоящее время информационные технологии играют ключевую роль в оптимизации и улучшении различных аспектов образовательного процесса. Одной из важных задач в системе образования является эффективный контроль за процессом учебы, включая проверку и оценку студентов. С увеличением числа обучающихся и разнообразия курсов возникает потребность в создании удобных и надежных программных решений для контроля пересдач студентов.

Система контроля пересдач студентов через портал в сфере высшего образования имеет принципиальное значение, обеспечивая не только прозрачность и эффективность в оценочном процессе, но также удобство и доступность для всех участников образовательного процесса. В условиях постоянно меняющейся образовательной среды такой портал предоставляет возможность контролировать и адаптировать процессы пересдач в соответствии с современными требованиями и стандартами.

Цель данной курсовой работы - разработать серверное приложение для микропортала для отслеживания успеваемости студентов на языке Java с использованием фреймворка Spring и паттерна MVC. Для достижения данной цели поставлены следующие задачи:

1. изучить основы фреймворка Spring, в частности Spring Data Jpa, Spring Security, Spring Core и Spring MVC;
2. провести анализ требований к серверному приложению;
3. спроектировать и разработать серверное приложение;
4. провести тестирование разработанного приложения.

Объектом исследования данной курсовой работы является серверное приложение для портала контроля успеваемости студентов, а предметом исследования - применение фреймворка Spring и паттерна MVC при разработке данного приложения.

При разработке проекта использовались знания, полученные в ходе лекционных и практических занятий по курсу «Разработка серверных частей интернет-ресурсов», а также дополнительные литературные источники.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для создания серверной части аддитивного микропортада по мониторингу пересдач студентов проведен анализ трех сайтов со схожей тематикой:

* <https://online-edu.mirea.ru> – Edu MIREA
* <http://univertv.ru/> – UniverTV.ru
* <https://www.lektorium.tv/>– Лекториум

Результаты анализа приведены в таблице 1.

Таблица 1 – Результат анализа сервисов со схожей тематикой.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Edu MIREA | UniverTV.ru | Лекториум | Вывод |
| Функциональность портала | Предоставляет возможность регистрации и авторизации студентов, а также загрузки необходимых документов для пересдачи. Обеспечивает удобный доступ к расписанию пересдач и информации о предметах. | Предоставляет возможность регистрации и авторизации студентов, проходящих курсы, а также позволяет отслеживать успеваемость. | Предоставляет возможность регистрации и авторизации студентов, проходящих курсы, а также позволяет отслеживать успеваемость. | Реализуемый портал должен предоставлять полный функционал для управления пересдачами и обеспечивать удобство использования. |
| Условие использования портала | Предоставляет прозрачные и удобные правила для регистрации, подачи заявок и получения информации. | Гибкие условия регистрации, подачи заявок и доступа к информации о курсах. | Ограниченные условия с некоторыми ограничениями в регистрации и доступе к информации. | Портал должен обеспечивать удобные и прозрачные условия использования для студентов. |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Удобство использования портала | Портал обладает интуитивно понятным интерфейсом, легким доступом к необходимой информации и удобными инструментами для подачи заявок. | Имеет менее интуитивный интерфейс, требующий дополнительного обучения для использования. | Имеет простой и удобный дизайн, обеспечивающий студентам легкий доступ к функционалу портала. | Разрабатываемое портал призвано обеспечивать максимальное удобство в использовании, учитывая специфику пользовательского опыта студентов. |
| Информативность портала | Предоставляет детальную информацию о предметах, условиях пересдач, требованиях к документам и прочей важной информации для подготовки. | Имеет ограниченную информацию о предметах. | Имеет ограниченную информацию о предметах и предоставляемых курсах. | Сервисное приложение должно быть информативным и обеспечивать студентов всей необходимой информацией для успешной пересдачи. |
| Качество предоставляемых услуг | Обеспечивает высокое качество организации пересдач, и полную информация о пересдачах. | Качество предоставляемых услуг находиться на удовлетворительном уровне. | Обеспечивает низкое качество предоставляемых услуг | Сервисное приложение должно гарантировать высокое качество предоставляемых услуг. |

Таким образом, разрабатываемый сервис должен быть функциональным, иметь четкие условия пользования, понятный и удобный интерфейс, быть информативным для пользователя и предоставлять услуги высокого качества.

1. ВЫБОР ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЯ
   1. Основные используемые технологии

Приложение функционирует в клиент-серверной архитектуре, используя тонкий клиент, где основная часть обработки данных происходит на серверной стороне. Для реализации этого подхода было выбрано взаимодействие между PostgreSQL, Spring и React.

Для реализации функционала приложения использован фреймворк Spring, который является одним из наиболее популярных фреймворков для разработки серверных приложений на языке Java. Spring предоставляет широкие возможности для управления жизненным циклом объектов, интеграции с другими технологиями и фреймворками, а также обеспечивает высокую производительность и безопасность приложения.

В частности, для работы с базой данных использован модуль Spring Data Jpa, который позволяет упростить работу с базой данных и снизить количество кода. Spring Data Jpa обеспечивает поддержку различных баз данных и предоставляет удобные методы для выполнения запросов к базе данных.

В качестве системы управления базами данных выбрана PostgreSQL. PostgreSQL — это реляционная система управления базами данных, которая использует язык SQL для манипулирования данными. PostgreSQL является одной из наиболее популярных СУБД в мире, благодаря своей надежности, масштабируемости и открытости.

Для обеспечения безопасности приложения использован модуль Spring Security, который предоставляет возможности для аутентификации и авторизации пользователей. Spring Security позволяет защитить приложение от различных видов атак и обеспечить безопасность передачи данных.

Для организации бизнес-логики использован модуль Spring Core, который обеспечивает управление зависимостями между объектами приложения. Spring Core позволяет упростить и структурировать код приложения, а также обеспечивает его переносимость и масштабируемость.

Для облегчения разработки и поддержки кода использована библиотека Lombok. Lombok — это библиотека для Java, которая помогает уменьшить объем написания шаблонного кода. Она предоставляет аннотации, которые автоматически генерируют конструкторы, методы доступа и т.д. для классов, что упрощает разработку и поддержку кода.

Для реализации контроллеров и представлений использован модуль Spring MVC, который обеспечивает возможности для реализации модели-представление-контроллер архитектуры приложения. Spring MVC позволяет разработчику легко создавать контроллеры, обрабатывать запросы и формировать ответы на основе представлений.

Среди многочисленных редакторов кода выбран редактор исходного кода IntelliJ IDEA, разработанный компанией JetBrains и предназначенный для использования на платформах Windows, Linux и macOS. IntelliJ IDEA позиционируется как мощное интегрированное средство разработки для создания веб- и облачных приложений, обладающее высокой производительностью и широким набором функций. Этот редактор идеально подходит для разработки сервисных приложений, предоставляя разработчикам удобное и эффективное рабочее окружение.

Выбор React для клиентской части приложения обоснован его компонентной архитектурой, что обеспечивает легкость поддержки и переиспользование кода. Использование виртуального DOM в React улучшает производительность интерфейса, минимизируя операции по обновлению реального DOM. Принцип однонаправленного потока данных в React обеспечивает четкое отслеживание состояния приложения. Большая поддержка сообщества, а также возможность использования React Native для мобильной разработки, делают React эффективным инструментом с широкими возможностями интеграции и поддержки.

Таким образом, приложение реализовано в клиент-серверной архитектуре с использованием тонкого клиента. Для взаимодействия с базой данных выбраны PostgreSQL, Spring и React. Фреймворк Spring обеспечивает эффективное управление жизненным циклом объектов, высокую производительность и безопасность приложения. Spring Data Jpa упрощает работу с базой данных, а Spring Security обеспечивает аутентификацию и авторизацию пользователей. IntelliJ IDEA, выбранный в качестве редактора кода, предоставляет удобное и эффективное рабочее окружение для создания веб- и облачных приложений. Выбор React для клиентской части обоснован его компонентной архитектурой, использованием виртуального DOM, принципом однонаправленного потока данных и поддержкой сообщества.

1. АРХИТЕКТУРА ПРИЛОЖЕНИЯ
   1. Описание архитектуры

В созданном приложении применяется архитектурный подход Model-View-Controller (MVC), который предполагает явное разделение на «модели», «виды» и «контроллеры». Модели в данном контексте определяют структуру сущностей в базе данных, виды представляют данные, которые сервер возвращает клиенту в ответ на запрос, а контроллеры обрабатывают запросы к серверу.

В контексте созданного приложения понятие контроллера подразделяется на три основных компонента: «rest-контроллеры», «сервисы» и «репозитории». Rest-контроллеры предоставляют возможность обработки запросов от клиента, определяя параметры для всех запросов, с которыми сервер должен взаимодействовать. Сервисы осуществляют обработку данных, полученных от клиента, используя репозитории и другие вспомогательные сервисы. Репозитории, в свою очередь, взаимодействуют с базой данных, выполняя SQL-запросы для эффективного управления данными.

На рисунке 3.1.1 показана схема используемой архитектуры.



Рисунок 4.1.1 – Архитектура MVC

* 1. Реализация архитектуры в разрабатываемом проекте

На рисунке 3.2.1 показана используемая архитектура на примере получения информации о предметах.

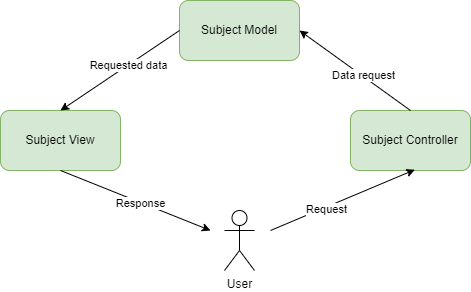


Рисунок 4.2.1 – Пример использования MVC архитектуры

Запрос от клиента поступает на контроллер, который обрабатывает запросы, касаемые предметов - листинг 3.2.1.

Листинг 3.2.1 – Rest – контроллер запросов данных о предметах

@RestController  
public class SubjectController {  
  
 private final SubjectService subjectService;  
 private final UserService userService;  
  
 @Autowired  
 public SubjectController(SubjectService subjectService, UserService userService) {  
 this.subjectService = subjectService;  
 this.userService = userService;  
 }  
  
  
 @GetMapping("/api/user/subject/teachers")  
 public List<Teacher> getSubjectTeachers(@RequestParam Long subjectId){  
 return subjectService.getTeachers(subjectId);  
 }  
}

Далее запрос обрабатывается в сервисе, обеспечивающем бизнес-логику предметов – Листинг 3.2.2

Листинг 3.2.2 – Сервис по работе с предметами

@Service  
public class SubjectService {  
  
 private final SubjectRepo subjectRepo;  
 private final GroupService groupService;  
  
 public Teacher getTeacherBySubject(Long subjectId, Long groupId) {  
 Subject subject = subjectRepo.findById(subjectId).orElseThrow();  
 Group group = groupService.get(groupId);  
 for (Teacher teacher : subject.getTeachers()){  
 for (Group teacherGroup : teacher.getGroups()){  
 if (teacherGroup.equals(group))  
 return teacher;  
 }  
 }  
 return null;  
 }

}

В качестве ответа возвращается модель данных Teacher, которая описывает учителя, ведущего предмет – Листинг 3.2.3.

Листинг 3.2.3 – Модель, возвращаемая сервером

@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Teacher {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String firstName;  
 private String lastName;  
  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 private List<Group> groups;  
  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 @JoinTable(  
 name = "teacher\_subjects",  
 joinColumns = @JoinColumn(  
 name = "teacher\_id", referencedColumnName = "id"),  
 inverseJoinColumns = @JoinColumn(  
 name = "subject\_id", referencedColumnName = "id"))  
 private List<Subject> subjects;  
}

1. РАЗРАБОТКА БАЗЫ ДАННЫХ
   1. Определение сущностей

Debt

Сущность Debt предназначена для описания задолженностей студентов, а именно день пересдачи, семестр, идентификатор студента и идентификатор предмета. SQL-команда для создания таблицы для данной сущности показана на листинге 4.1.1.1.

Листинг 4.1.1.1 – Описание таблицы Debt

CREATE TABLE IF NOT EXISTS debt(  
 id SERIAL PRIMARY KEY NOT NULL,  
 retake\_day DATE,  
 term INTEGER NOT NULL,  
 student\_id BIGINT CONSTRAINT fk6xaptqhsrjgb1eg7mkwof8wm

Продолжение листинга 4.1.1.1

REFERENCES student,  
 subject\_id BIGINT CONSTRAINT uk\_jyfvioi2ka6rv1dldxu8qgff5

UNIQUE  
 CONSTRAINT fkipoufvwep3j5i712827h2p5hf REFERENCES subject  
);

Group

Таблица student\_groups представляет "Группы студентов". В каждой группе есть уникальный идентификатор (id), который автоматически увеличивается с созданием новой группы. У группы также есть название (name), представленное в виде текстовой строки, и срок (term), который указывает на текущий учебный семестр, в котором находится данная группа – листинг 4.1.2.1.

Листинг 4.1.2.1 – Описание таблицы student\_groups

CREATE TABLE IF NOT EXISTS student\_groups(  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 term INT  
);

Performance

Эта сущность представляет "Успеваемость студентов". Каждая запись в таблице содержит уникальный идентификатор (id), который автоматически увеличивается при добавлении новой записи. Также в таблице присутствует оценка (mark), которая представляет собой числовое значение успешности студента. Каждая оценка связана с определенным студентом, что отражено в поле "student\_id", а также с определенным предметом, что отражено в поле "subject\_id".

Дополнительно, есть поле "term", которое указывает на текущий учебный семестр или период времени, к которому относится данная успеваемость. Это позволяет отслеживать успехи студентов в определенный период времени - листинг 4.1.3.1.

Листинг 4.1.3.1 – Описание таблицы Performance

CREATE TABLE IF NOT EXISTS performance(  
 id SERIAL PRIMARY KEY NOT NULL,

Продолжение листинга 4.1.3.1

mark INTEGER NOT NULL,  
 student\_id BIGINT CONSTRAINT fks9pjut2jvjpm3xsl1scdgipdn REFERENCES student,  
 subject\_id BIGINT CONSTRAINT uk\_nrxsudv3v22xjtm294lgatwcu UNIQUE  
 CONSTRAINT fko0c8jglas44w1c5tjwjmq4y6t REFERENCES subject,

Листинг 4.1.3.1 -

term INTEGER  
);

Role

Таблица, в которой хранятся записи сущностей Role представляет "Роли" в системе. У каждой роли есть уникальный идентификатор (id), который автоматически увеличивается при создании новой роли. Также у роли есть название (name), представленное в виде текстовой строки. Данные роли выдаются пользователю, чтобы разграничить предоставляемый функционал – листинг 4.1.4.1.

Листинг 4.1.4.1 – Описание таблицы Role

CREATE TABLE IF NOT EXISTS role(  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL  
);

Student

Сущность Student - представляет собой "Студентов". Каждый студент имеет уникальный идентификатор (id), который автоматически увеличивается при создании новой записи. Поля "first\_name" и "last\_name" содержат имя и фамилию студента соответственно – листинг 4.1.5.1.

Поле "group\_id" связано с таблицей "student\_groups" и указывает на принадлежность студента к определенной группе. Кроме того, у студента есть уникальный код (code) и номер семестра (term), которые отражают соответственно уникальные идентификационные данные и текущий семестр, в котором находится студент.

Поле "user\_id" связано с таблицей "users\_table" и представляет собой ссылку на пользователя, связанного с данным студентом.

Листинг 4.1.5.1 – Описание таблицы Student

CREATE TABLE IF NOT EXISTS student(  
 id SERIAL PRIMARY KEY,  
 first\_name VARCHAR(255),  
 last\_name VARCHAR(255),  
 group\_id INT REFERENCES student\_groups(id),  
 code VARCHAR(255),  
 term INT,  
 user\_id BIGINT REFERENCES users\_table(id)  
);

Subject

Сущность Subject представляет собой "Предметы" в учебной программе. У каждого предмета есть уникальный идентификатор (id), который представлен в виде целого числа большего размера (BIGINT). Поля "name" и "exam" содержат соответственно название предмета и тип экзамена, связанного с этим предметом – листинг 4.1.6.1.

Листинг 4.1.6.1 – Описание таблицы Subject

CREATE TABLE IF NOT EXISTS subject(  
 id BIGINT NOT NULL PRIMARY KEY,  
 name VARCHAR(255),  
 exam VARCHAR(20)  
);

Teacher

Эта таблица представляет собой "Преподавателей". У каждого преподавателя есть уникальный идентификатор (id), который представлен в виде целого числа большего размера (BIGINT). Поля "first\_name" и "last\_name" содержат соответственно имя и фамилию преподавателя – листинг 4.1.7.1.

Листинг 4.1.7.1 – Описание таблицы Teacher

CREATE TABLE IF NOT EXISTS teacher(  
 id BIGINT PRIMARY KEY NOT NULL,  
 first\_name VARCHAR(255),  
 last\_name VARCHAR(255)  
);

User

Эта таблица представляет "Пользователей" системы. У каждого пользователя есть уникальный идентификатор (id), который автоматически увеличивается при создании новой записи. Поля "email" и "password" содержат соответственно электронную почту пользователя и его пароль для входа в систему. Поле "photo\_url" предназначено для хранения URL-адреса фотографии пользователя – листинг 4.1.8.1.

Листинг 4.1.8.1 – Описание таблицы Users\_table

CREATE TABLE IF NOT EXISTS users\_table (  
 id SERIAL PRIMARY KEY,  
 email VARCHAR(255) NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 photo\_url VARCHAR(255)  
);

Таким образом, каждая запись в таблице "users\_table" содержит информацию о конкретном пользователе, включая его электронную почту, пароль и, при наличии, URL-адрес фотографии.

* 1. Проектирование базы данных
     1. Описание дополнительных таблиц

Для связи описанных выше таблиц необходимо разработать таблицы устанавливающие такие связи.

Таблицы "teacher\_groups" и "teacher\_subjects" устанавливают связи между преподавателями, группами студентов и предметами, определяя их преподавательские обязанности – листинги 4.2.1.1 и 4.2.1.2.

Листинг 4.2.1.1 – Описание таблицы Teacher\_groups

create table if not exists teacher\_groups(  
 teacher\_id bigint not null  
 constraint fkh6egnyvnn8toiohupap6fmhu9  
 references teacher,  
 groups\_id bigint not null  
 constraint fk6p16c6kwgh63vxtxdfgd8d6jd  
 references student\_groups  
);

Листинг 4.2.1.2 – Описание таблицы Teacher\_subjects

create table if not exists teacher\_subjects(  
 teacher\_id bigint not null  
 constraint fkjkx6egayo8f0yrpjb1fychxbq  
 references teacher,  
 subject\_id bigint not null  
 constraint fkn680sbj5cc0wo9yeajo62gfwb  
 references subject  
);

Связь "roles\_users" предоставляет информацию о ролях, назначенных пользователям, позволяя определить их права в системе – листинг 4.2.1.3.

Листинг 4.2.1.3 – Описание таблицы Roles\_users

CREATE TABLE IF NOT EXISTS roles\_users(  
 user\_id BIGINT NOT NULL  
 CONSTRAINT fk35a8iedo7x2umx781eo50yf1q  
 REFERENCES users\_table,  
 role\_id BIGINT NOT NULL  
 CONSTRAINT fkiekkooywvstxn9cgt1iyvorq8  
 REFERENCES role,  
 PRIMARY KEY (user\_id, role\_id)  
);

Дополнительные таблицы "student\_groups\_students" и "student\_groups\_subjects" обеспечивают связи между студентами, группами и предметами, подчеркивая их принадлежность и вовлеченность в учебный процесс – листинги 4.2.1.4 и 4.2.1.5.

Листинг 4.2.1.4 – Описание таблицы Studet\_groups\_students

CREATE TABLE IF NOT EXISTS student\_groups\_students(  
 group\_id BIGINT NOT NULL CONSTRAINT fkjtkghxwraaigkx4k79pjvtapx  
 references student\_groups,  
 students\_id BIGINT NOT NULL  
 CONSTRAINT uk\_nvj53yjr5m07kju6lnc0lh6h3  
 UNIQUE  
 CONSTRAINT fkehhaff5u7mjcxrrp58mxoj9uy  
 REFERENCES student  
);

Листинг 4.2.1.5 – Описание таблицы Studet\_groups\_students

CREATE TABLE IF NOT EXISTS student\_groups\_subjects(  
 group\_id BIGINT NOT NULL  
 CONSTRAINT fkqk2s15grlfl1n7dmhtubnovyt

Продолжение листинга 4.2.1.5

REFERENCES student\_groups,  
 subjects\_id BIGINT NOT NULL  
 CONSTRAINT uk\_4v42jgpgqhqw7eo13v4j1wuou  
 UNIQUE  
 CONSTRAINT fkkjrhc4quuom143fu2ns3oqi40  
 REFERENCES subject  
);

* + 1. Создание ER-диаграммы

На рисунке 4.2.2.1 показана ER-диаграмма используемой базы данных.

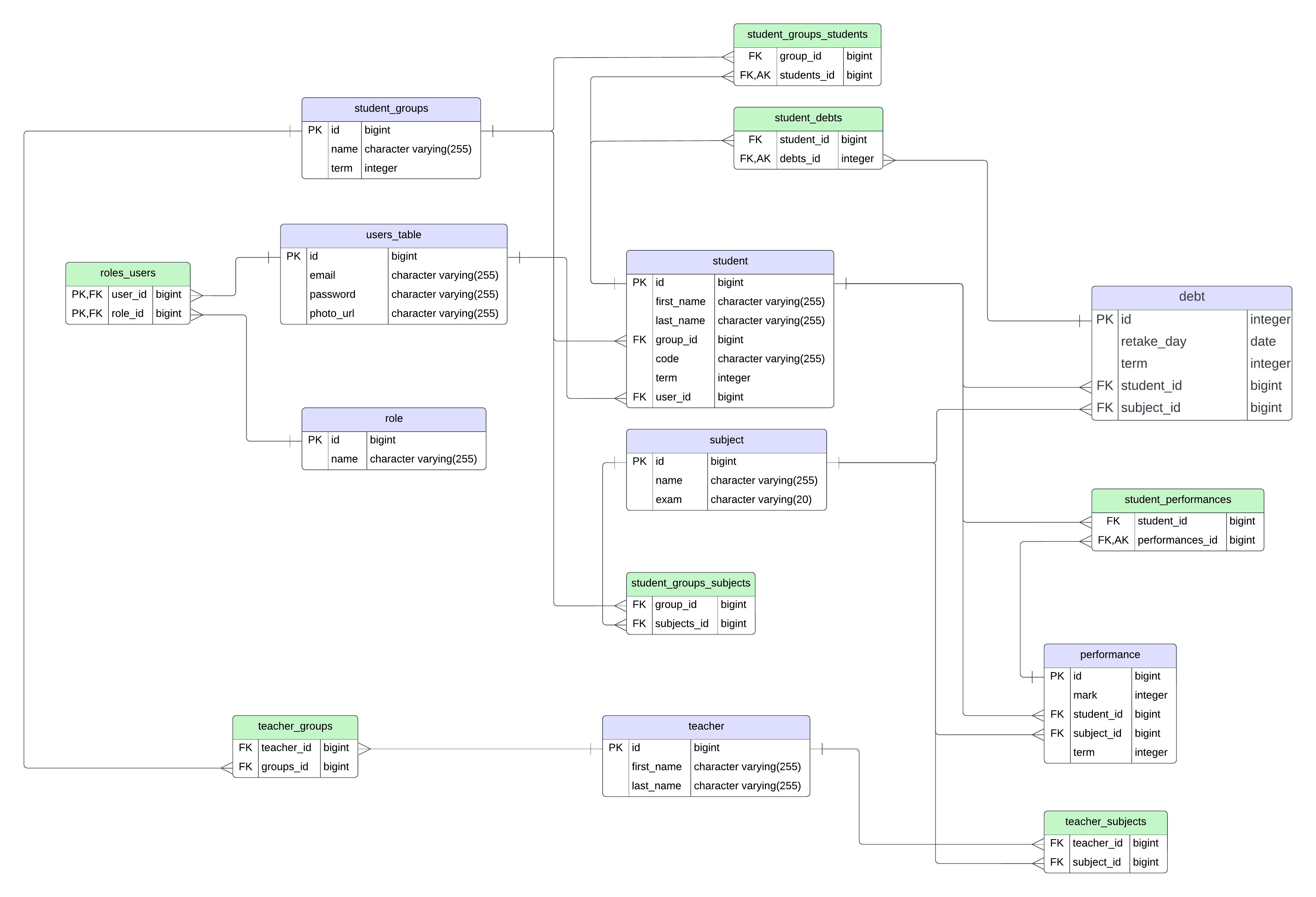


Рисунок 4.2.2.1 – ER-диаграмма спроектированной базы данных

1. РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ
   1. Структура проекта

Структура проекта показана на рисунке 5.1.1. В папке проекта находятся директории с Java файлами для работы приложения, а также технические файлы: файл для сборки Maven проекта, конфигурация для работы в InteliJ IDEA. Помимо этого, присутствует dockerfile для создания образа и последующего создания контейнера с использованием данного образа и папка с SQL криптами для создания таблиц в базе данных и наполнения их необходимыми данными.

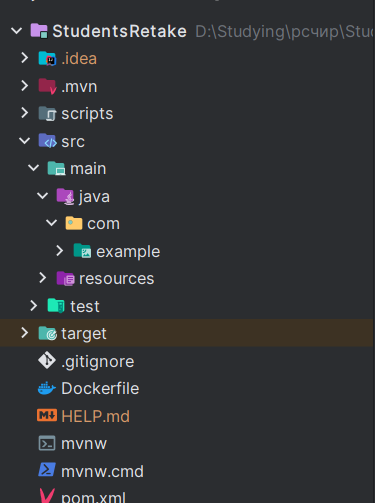


Рисунок 5.1.1 – Структура проекта

Основной код проекта находится в папке main. Там находятся файлы конфигурации проекта, а также директории: контроллеров, сервисов, сущностей, dao и репозиториев – рисунок 5.1.2.

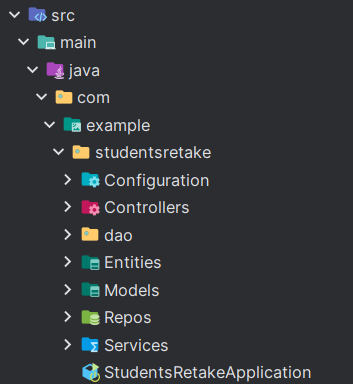


Рисунок 5.1.2 – Структура директории main

* 1. Конфигурация приложения

Конфигурация приложения описана в файле application.properties, который представлен на листинге 5.2.1.

Листинг 5.2.1 – Файл application.properties

logging.level.com.example.studentsretake.Controllers=debug  
logging.level.com.example.studentsretake.Services=debug  
logging.level.com.example.studentsretake.Configuration=debug  
  
spring.jpa.hibernate.ddl-auto=update  
  
spring.datasource.url=jdbc:postgresql://dpg-cllpl4kjtl8s73epm6u0-a.frankfurt-postgres.render.com/dbcw  
spring.datasource.username=vanyapro2004  
spring.datasource.password=jKkNYlHfnMqyHtfScMMKctyNZige9dd3  
  
logging.level.org.springframework.security=INFO  
logging.level.org.springframework.security.authentication=DEBUG  
  
spring.mvc.static-path-pattern=/static/\*\*  
  
token.signing.key=413F4428472B4B6250655368566D5970337336763979244226452948404D6351

Данный файл содержит необходимую конфигурацию для подключения к базе данных, настройку уровня логирования, путь к статическим файлам проекта и переменную, которая используется для генерации JWT.

* 1. Использование Spring Security для авторизации и аутентификации пользователей

Авторизация в данном Spring Boot приложении реализована с помощью Spring Security и JSON Web Tokens. Разработанное приложение поддерживает 3 роли для пользователей: STUDENT, TEACHER и ADMIN – листинг 5.3.1.

Листинг 5.3.1 – Класс-сущность для определения роли пользователя

@Entity  
@Data  
public class Role implements GrantedAuthority {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
  
 @Override  
 public String getAuthority() {

Продолжение листинга 5.3.1

return name;  
 }  
}

* + 1. Класс «SpringConfig»

Для реализации авторизации при помощи Spring Security JWT был создан конфигурационный класс SecurityConfig – листинг 5.3.1.1.

Листинг 5.3.1.1 – Класс SecurityConfig

@Configuration  
@RequiredArgsConstructor  
@EnableWebSecurity  
public class SecurityConfiguration {  
 private final JwtAuthenticationFilter jwtAuthenticationFilter;  
 @Resource  
 private UserDetailsService userDetailsService;  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(AbstractHttpConfigurer::disable)  
 .cors(Customizer.*withDefaults*())  
 .authorizeHttpRequests(  
 (request) -> request  
 .requestMatchers("api/admin/\*\*").hasRole("ADMIN")  
 .requestMatchers("api/student/\*\*").hasAnyRole("STUDENT", "ADMIN")  
 .requestMatchers("api/teacher/\*\*").hasAnyRole("TEACHER", "ADMIN")  
 .requestMatchers("/public/\*\*", "/api/auth/\*\*", "/static/\*\*").permitAll()  
 .anyRequest().authenticated()  
 )  
 .sessionManagement(manager -> manager.sessionCreationPolicy(*STATELESS*))  
 .authenticationProvider(authenticationProvider()).addFilterBefore(  
 jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);  
  
 return http.build();  
 }  
  
 @Bean  
 public CorsConfigurationSource corsConfigurationSource() {

Продолжение листинга 5.3.1.1.

CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(List.*of*("http://localhost:3000"));  
 configuration.setAllowedMethods(Arrays.*asList*("GET", "POST", "PUT", "DELETE"));  
 configuration.setAllowedHeaders(Arrays.*asList*("content-type", "authorization"));  
  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration("/\*\*", configuration);  
  
 return source;  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public DaoAuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();  
 authenticationProvider.setUserDetailsService(userDetailsService);  
 authenticationProvider.setPasswordEncoder(passwordEncoder());  
 return authenticationProvider;  
 }  
  
 @Bean  
 public AuthenticationManager authenticationManager(AuthenticationConfiguration config)  
 throws Exception {  
 return config.getAuthenticationManager();  
 }  
  
}

* + 1. Класс «JwtAuthentificationFilter»

Также потребовался класс JwtAuthentificationFilter, который отвечает за аутентификацию при помощи JWT – листинг 5.3.2.1.

Листинг 5.3.2.1 – Класс JwtAuthentificationFilter

public class JwtAuthenticationFilter extends OncePerRequestFilter {  
  
 @Autowired  
 private JwtService jwtService;

@Autowired  
private UserDetailsService userDetailsService;  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain filterChain) throws ServletException, IOException {  
 final String authHeader = request.getHeader("Authorization");  
 final String jwt;  
 final String userEmail;  
 if (StringUtils.*isEmpty*(authHeader) || !StringUtils.*startsWith*(authHeader, "Bearer ")) {  
 filterChain.doFilter(request, response);  
 return;  
 }  
 jwt = authHeader.substring(7);  
 userEmail = jwtService.extractUserName(jwt);  
 *log*.debug(userEmail);  
 if (StringUtils.*isNotEmpty*(userEmail)  
 && SecurityContextHolder.*getContext*().getAuthentication() == null) {  
 UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);  
 if (jwtService.isTokenValid(jwt, userDetails)){  
 SecurityContext context = SecurityContextHolder.*createEmptyContext*();  
 UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(  
 userDetails, null, userDetails.getAuthorities()  
 );  
 authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
 context.setAuthentication(authenticationToken);  
 SecurityContextHolder.*setContext*(context);  
 }  
 }  
 filterChain.doFilter(request, response);  
 }  
}

* + 1. Класс «AuthentificationServiceImpl»

Для обработки запросов пользователя на авторизацию создан интерфейс AuthentificationService и его реализация – AuthentificationServiceImpl, листинг 5.3.3.1.

Листинг 5.3.3.1 – Класс AuthentificationServiceImpl

@Service  
@RequiredArgsConstructor  
public class AuthenticationServiceImpl implements AuthenticationService {  
 private final UserRepo userRepo;  
 private final PasswordEncoder passwordEncoder;  
 private final JwtService jwtService;  
 private final AuthenticationManager authenticationManager;  
 private final UserService userService;  
 private final StudentService studentService;  
  
  
 @Override  
 public JwtAuthenticationResponse signUp(SignUpRequest request) {  
 var user = User.*builder*().email(request.getEmail()).password(passwordEncoder.encode(request.getPassword()))  
 .build();  
 user = userService.saveUserWithRole(user, request.getRole());  
 Student student = Student.*builder*().user(user).build();  
 studentService.create(student);  
 var jwt = jwtService.generateToken(user);  
 return JwtAuthenticationResponse.*builder*().token(jwt).build();  
 }  
  
 @Override  
 public JwtAuthenticationResponse signIn(SignInRequest request) {  
 authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword()));  
 User user = userRepo.findUserByEmail(request.getEmail()).orElseThrow(() ->  
 new UsernameNotFoundException(""));  
 var jwt = jwtService.generateToken(user);  
 return JwtAuthenticationResponse.*builder*().token(jwt).roles(user.getRoles()).build();  
 }  
}

* + 1. Класс «JwtServiceImpl»

На листинге 5.3.4.1 представлен код класса, реализующего интерфейс JwtService, который необходим для авторизации с помощью JWT.

Листинг 5.3.4.1 – Класс JwtServiceImpl

@Service  
@Slf4j  
public class JwtServiceImpl implements JwtService {  
  
 @Value("${token.signing.key}")  
 private String jwtSigningKey;  
  
  
 @Override  
 public String extractUserName(String token) {  
 return extractClaim(token, Claims::getSubject);  
 }  
  
 private <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  
 final Claims claims = extractAllClaims(token);  
 return claimsResolver.apply(claims);  
 }  
  
 private Claims extractAllClaims(String token) {  
 return Jwts.*parser*().setSigningKey(getSigningKey()).build().parseClaimsJws(token)  
 .getPayload();  
 }  
  
 private Key getSigningKey() {  
 byte[] keyBytes = Decoders.*BASE64*.decode(jwtSigningKey);  
 return Keys.*hmacShaKeyFor*(keyBytes);  
 }  
  
 @Override  
 public String generateToken(UserDetails userDetails) {  
 return generateToken(new HashMap<>(), userDetails);  
 }  
  
 private String generateToken(Map<String, Object> extraClaims, UserDetails userDetails){  
 *log*.debug("\n" + userDetails.getUsername() + "\n");  
 return Jwts.*builder*().claims(extraClaims).subject(userDetails.getUsername())  
 .issuedAt(new Date(System.*currentTimeMillis*()))  
 .expiration(new Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60 \* 24))  
 .signWith(getSigningKey(),

Продолжение листинга 5.3.4.1.

SignatureAlgorithm.*HS256*).compact();  
 }  
  
 @Override  
 public boolean isTokenValid(String token, UserDetails

userDetails) {  
 final String userName = extractUserName(token);  
 return (userName.equals(userDetails.getUsername()) && !isTokenExpired(token));  
 }  
  
 private boolean isTokenExpired(String token) {  
 return extractExpiration(token).before(new Date());  
 }  
  
 private Date extractExpiration(String token) {  
 return extractClaim(token, Claims::getExpiration);  
 }  
}

* 1. Использование Spring Data JPA

В данном проекте выбран именно Spring Data JPA для взаимодействия с базой данных PostgreSQL. Этот фреймворк позволяет создавать репозитории, которые автоматически реализуют CRUD-операции (Create, Read, Update, Delete) над объектами базы данных, что существенно упрощает написание кода и сокращает время разработки.

Для абстрагирования от сложностей взаимодействия с базой данных напрямую и работать с объектами на более высоком уровне абстракции использованы компоненты сущности, такие компоненты помечаются аннотацией «Entity». В процессе разработки созданы следующие сущности: Debt, Group, Performance, Role, Student, Subject, Teacher, User.

На листинге 5.4.1 представлен код класса, который реализует сущность Teacher.

Листинг 5.4.1 – Код класса Teacher

@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Teacher {

Продолжение листинга 5.4.1

@Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String firstName;  
 private String lastName;

@ManyToMany(fetch = FetchType.*EAGER*)  
 private List<Group> groups;  
  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 @JoinTable(  
 name = "teacher\_subjects",  
 joinColumns = @JoinColumn(  
 name = "teacher\_id", referencedColumnName = "id"),  
 inverseJoinColumns = @JoinColumn(  
 name = "subject\_id", referencedColumnName = "id"))  
 private List<Subject> subjects;  
}

Остальные сущности разработаны по такому же принципу и отличаются только некоторыми полями, в зависимости от конкретных требований реализации.

* 1. Классы, обеспечивающие работу с базой данных

JpaRepository предоставляет набор готовых методов для работы с сущностями в базе данных, таких как findAll(), save(), delete(), и т.д.

В процессе разработки созданы следующие интерфейсы: DebtRepo, GroupRepo, PerformanceRepo, RoleRepo, StudentRepo, SubjectRepo, TeacherRepo, UserRepo.

На листинге 5.5.1 показан код интерфейса TeacherRepo. Остальные интерфейсы схожи с представленным на листинге.

Листинг 5.5.1 – Код интерфейса TeacherRepo

@Repository  
public interface TeacherRepo extends JpaRepository<Teacher, Long> {  
}

* 1. Разработка бизнес-логики приложения

Для реализации бизнес-логики приложения были разработаны следующие классы: DebtService, GroupService, StudentService, SubjectService, TeacherService, UserService и UserDetailsService. Эти сервисы предоставляют функциональность для работы с данными, выполнения операций и обеспечивают взаимодействие между различными компонентами системы. Каждый класс сервиса выделен для конкретного функционального блока, гарантируя четкую структуру и модульность бизнес-логики. Далее рассмотрим класс «TeacherSerivce», остальные классы-сервисы спроектированы похожим образом, отличаясь только определенными методами, необходимыми под определенные бизнес задачи, кроме класса «UserDetailsService», который будет рассмотрен отдельно.

Класс «TeacherSerivce»

Данный класс обеспечивает логику работы, касающуюся учителей в данном проекте – листинг 5.6.1.1.

Листинг 5.6.1.1 – Код класса «TeacherServiceImpl»

@Service  
@RequiredArgsConstructor  
public class TeacherServiceImpl implements TeacherService {  
  
 private final TeacherRepo teacherRepo;  
  
 @Override  
 public Teacher create(Teacher teacher) {  
 return teacherRepo.save(teacher);  
 }  
  
 @Override  
 public Teacher read(Long id) {  
 return teacherRepo.findById(id).orElseThrow();  
 }  
  
  
 @Override  
 public Teacher update(Long id, Teacher newStudent) {  
 return null;  
 }  
  
 @Override  
 public void delete(Long id) {  
 teacherRepo.deleteById(id);  
 }

Продолжение листинга 5.6.1.1

@Override  
 public Teacher getById(Long teacherId) {  
 return teacherRepo.findById(teacherId).orElseThrow();  
 }  
  
 @Override  
 public List<Subject> getTeacherSubjects(Long teacherId) {  
 Teacher teacher = teacherRepo.findById(teacherId).orElseThrow();  
 return teacher.getSubjects();  
 }

В рассмотренном примере класс, помеченный аннотацией «@Serivce», представляет сервисную реализацию интерфейса «TeacherSerice». Внедрение зависимости «@RequiredArgsConstructor» осуществляется через конструктор, где передается экземпляр TeacherRepo – интерфейса репозитория для взаимодействия с базой данных.

Метод «create» отвечает за сохранение нового объекта преподавателя в базе данных с использованием метода save репозитория.

Метод «read» возвращает объект преподавателя по указанному идентификатору, используя метод findById репозитория. В случае отсутствия записи с указанным идентификатором, выбрасывается исключение.

Метод «update» предполагает обновление данных о преподавателе по его идентификатору. В данной реализации метода отсутствует логика обновления, и он всегда возвращает null. Необходимо доработать данный метод для реализации конкретной логики обновления.

Метод «delete» удаляет запись о преподавателе по указанному идентификатору с использованием метода deleteById репозитория.

Метод «getById» возвращает объект преподавателя по указанному идентификатору или выбрасывает исключение, если запись не найдена.

Метод «getTeacherSubjects» возвращает список предметов, преподаваемых указанным преподавателем. Для этого он получает объект преподавателя из репозитория и вызывает метод getSubjects.

Общий функционал класса включает в себя основные CRUD-операции для объектов преподавателей и предоставление информации о предметах, которые преподаватель ведет.Начало формы

Класс «UserDetailsService»

Отдельно рассмотрим класс UserDetailsService, который необходим для правильной работы Spring Security – листинг 5.6.2.1.

Листинг 5.6.2.1 – Код класса «UserDetailsService»

@Service  
public class UserDetailsService implements org.springframework.security.core.userdetails.UserDetailsService {  
 @Autowired  
 private UserRepo userRepo;  
 @Override  
 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {  
 User user = userRepo.findUserByEmail(email).orElseThrow(() -> new UsernameNotFoundException(email));  
  
 return org.springframework.security.core.userdetails.User.*withUsername*(user.getEmail())  
 .authorities(user.getAuthorities())  
 .password(user.getPassword())  
 .build();  
 }  
}

Данный класс, отмеченный аннотацией «@Service», предоставляет реализацию интерфейса UserDetailsService, предназначенного для загрузки пользовательских данных в Spring Security. Метод loadUserByUsername выполняет загрузку данных о пользователе по его электронной почте (предполагается, что электронная почта является уникальным идентификатором пользователя). Если пользователь с указанным адресом электронной почты не найден, выбрасывается исключение UsernameNotFoundException.

Далее, на основе данных пользователя, создается объект UserDetails из пакета Spring Security. Этот объект представляет собой интерфейс, предоставляющий информацию о пользователе, которую Spring Security использует для аутентификации и авторизации. Метод возвращает объект UserDetails, содержащий электронную почту, роли (или другие права доступа), и хэшированный пароль пользователя.

Таким образом, этот класс является частью конфигурации безопасности Spring и предоставляет мост между системой аутентификации Spring Security и хранилищем данных о пользователях (в данном случае, базой данных, представленной UserRepo).

* 1. Использование Spring MVC для реализации веб-интерфейса

Для правильного маппинга запросов пользователя разработаны контроллеры с использованием Spring MVC. Каждый контроллер отвечает за обработку конкретных HTTP-запросов, управление данными и вызов соответствующих сервисов для обеспечения бизнес-логики. А именно разработаны следующие классы: AdminUserController, AuthenticationController, DebtController, GroupController, RegistrationController, RoleController, StudentController, StudentGroupsController, SubjectController, TeacherController, UserController.

Рассмотрим класс GroupController, отвечающий за маппинг запросов, касаемых студенческих групп – листинг 5.7.1.

Листинг 5.7.1 – Код класса «GroupController»

@RestController  
@RequiredArgsConstructor  
@RequestMapping("/api/admin/groups")  
public class GroupController {  
  
 private final GroupService groupService;  
 private final UserService userService;  
 private final StudentServiceImpl studentService;  
 private final SubjectService subjectService;  
  
  
 @PostMapping("/add-group")  
 public ResponseEntity<?> createGroup(@RequestBody GroupFormRequest request){  
 Group newGroup = groupService.create(request);  
  
 URI location = ServletUriComponentsBuilder  
 .*fromCurrentRequest*()  
 .path("/{id}")  
 .buildAndExpand(newGroup.getId())  
 .toUri();

Продолжение листинга 5.7.1

return ResponseEntity.*created*(location).body(newGroup);  
 }  
  
 @DeleteMapping("/delete-group")  
 public ResponseEntity<?> deleteGroup(@RequestParam Long id){  
 groupService.delete(id);  
 return ResponseEntity.*ok*("Group deleted");  
 }  
  
 @PutMapping("/add-student")  
 public ResponseEntity<?> addStudentToGroup(@RequestBody GroupFormRequest request){  
 Group group = groupService.get(request.getGroupId());  
 Student student = studentService.read(request.getStudentId());  
 if (group != null && student != null){  
 student.setGroup(group);  
 group.getStudents().add(student);  
 groupService.update(group, request.getGroupId());  
 studentService.update(request.getStudentId(), student);  
 return ResponseEntity.*ok*("Student was added successfully");  
 }else{  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @DeleteMapping("/admin/group-student/remove")  
 public ResponseEntity<?> deleteStudentFromGroup(@RequestParam Long groupId,  
 @RequestParam Long studentId){  
  
 Group group = groupService.get(groupId);  
 Student student = studentService.read(studentId);  
  
 if (group != null && student != null){  
 group.getStudents().clear();  
 student.setGroup(null);  
 groupService.update(group, groupId);  
 studentService.create(student);  
 return ResponseEntity.*ok*("Student " + student.getFirstName() + " was deleted from group " + group.getName()  
 + " successfully");  
 }else{  
 return ResponseEntity.*notFound*().build();  
 }  
 }

Продолжение листинга 5.7.1

@PutMapping("/add-subject")  
 public ResponseEntity<?> addSubjectToGroup(@RequestBody GroupFormRequest request){  
 Subject subject = subjectService.getSubjectFromBd(request.getSubjectId());  
 Group group = groupService.get(request.getGroupId());  
  
 if (subject != null && group != null){  
 group.getSubjects().add(subject);  
 groupService.update(group, request.getGroupId());  
 return ResponseEntity.*ok*("Subject " + subject.getName() + " was added to group " + group.getName()  
 + " successfully");  
 }else{  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @DeleteMapping("/remove-subject")  
 public ResponseEntity<?> removeSubjectFromGroup(@RequestParam Long groupId,  
 @RequestParam Long subjectId){  
 Subject subject = subjectService.getSubjectFromBd(subjectId);  
 Group group = groupService.get(groupId);  
  
 if (subject != null && group != null){  
 group.getSubjects().clear();  
 groupService.update(group, groupId);  
 return ResponseEntity.*ok*("Subject " + subject.getName() + " was deleted from group " + group.getName()  
 + " successfully");  
 }else{  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

Каждый класс в данном разделе описывает логику обработки запросов от пользователя. Остальные класса реализуют данную логику таким же образом, у каждого класса есть методы, которые обрабатывают определенные запросы различных типов (GET, POST, DELETE и PUT) и их реализация.

1. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ
   1. Структура проекта

Структура проекта представлена на рисунке 6.1.1.

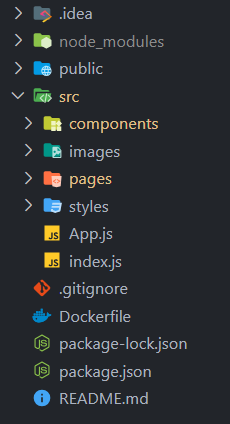


Рисунок 6.1.1 – Структура проекта

В основной директории находятся конфигурационные файлы, Dockerfile для создания образа, а также директория src, которая содержит файлы приложения, файлы стилей, компонентов и страниц.

* 1. Страницы разработанного приложения

На рисунках 6.2.1 – 6.2.9 показаны разработанные страницы клиентской части проекта.

Главная страница содержит основную информацию, необходимую для пользователя. Страница профиля пользователя содержит основную информацию о пользователе. Страница с траекторией обучения содержит информацию о предметах в текущем семестре, а также информацию о преподавателе, который ведет этот предмет. Страница с электронной зачетной книжкой хранит информацию об успеваемости студента, также присутствует возможность быстрого поиска предмета с помощью сортировки или поиска по названию. Страница с задолженностями хранит информацию о прошедших и предстоящих пересдачах студента.

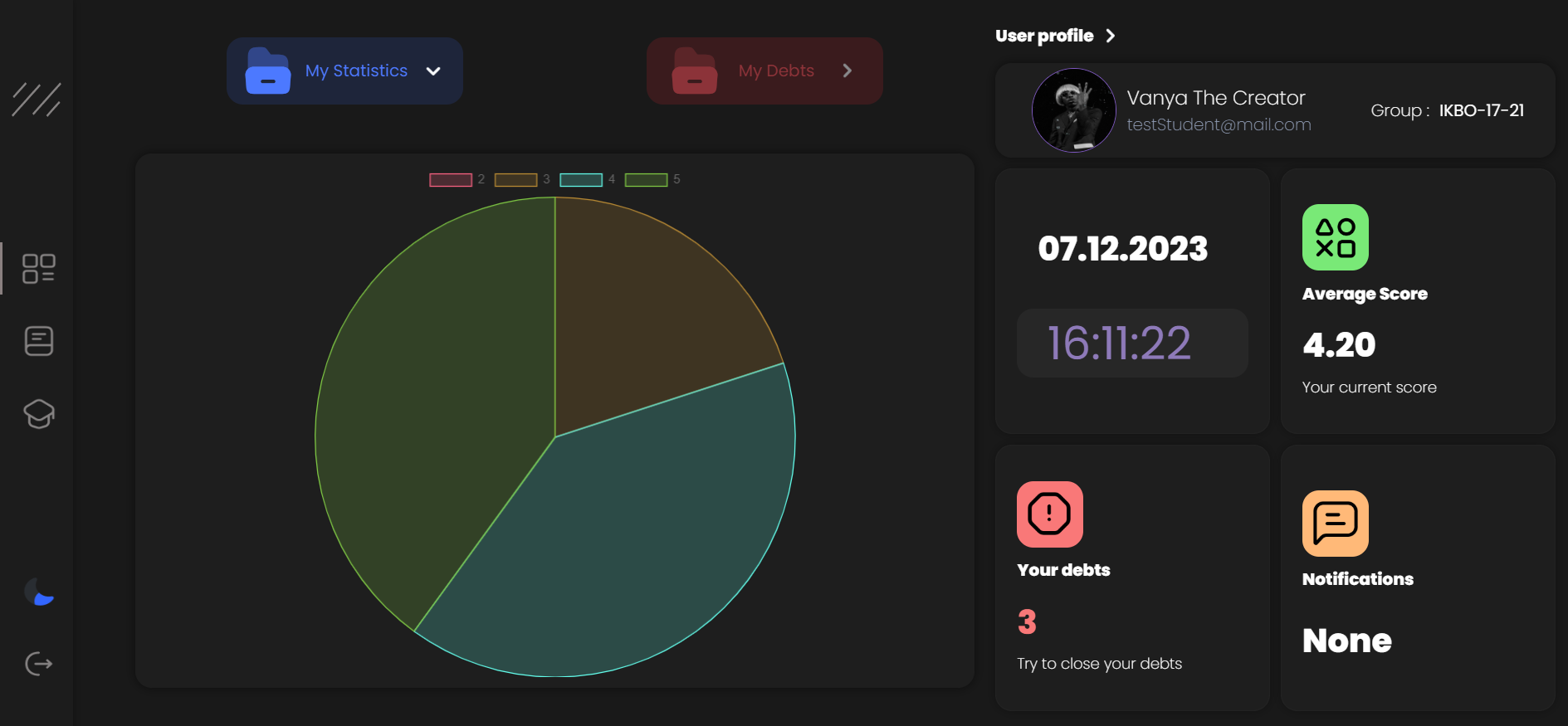


Рисунок 6.2.1 – Главная страница

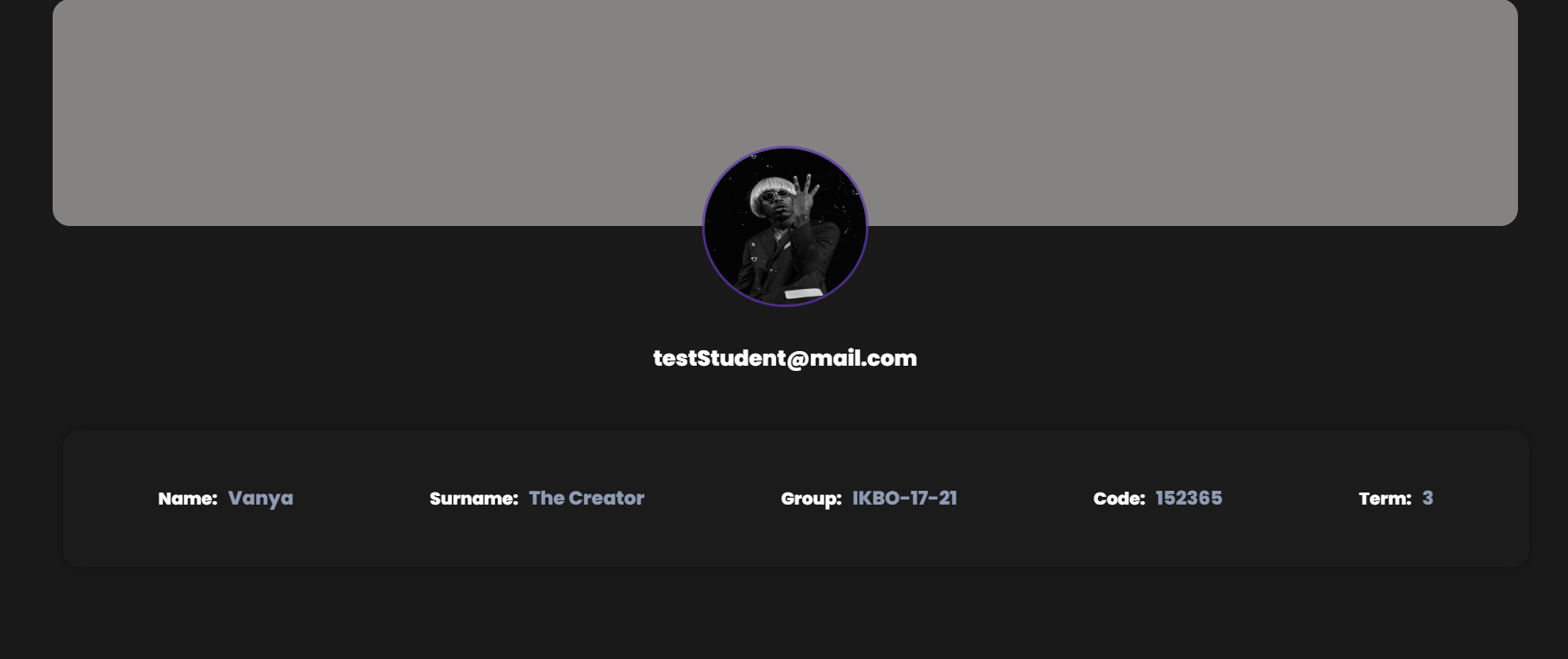


Рисунок 6.2.2 – Профиль пользователя

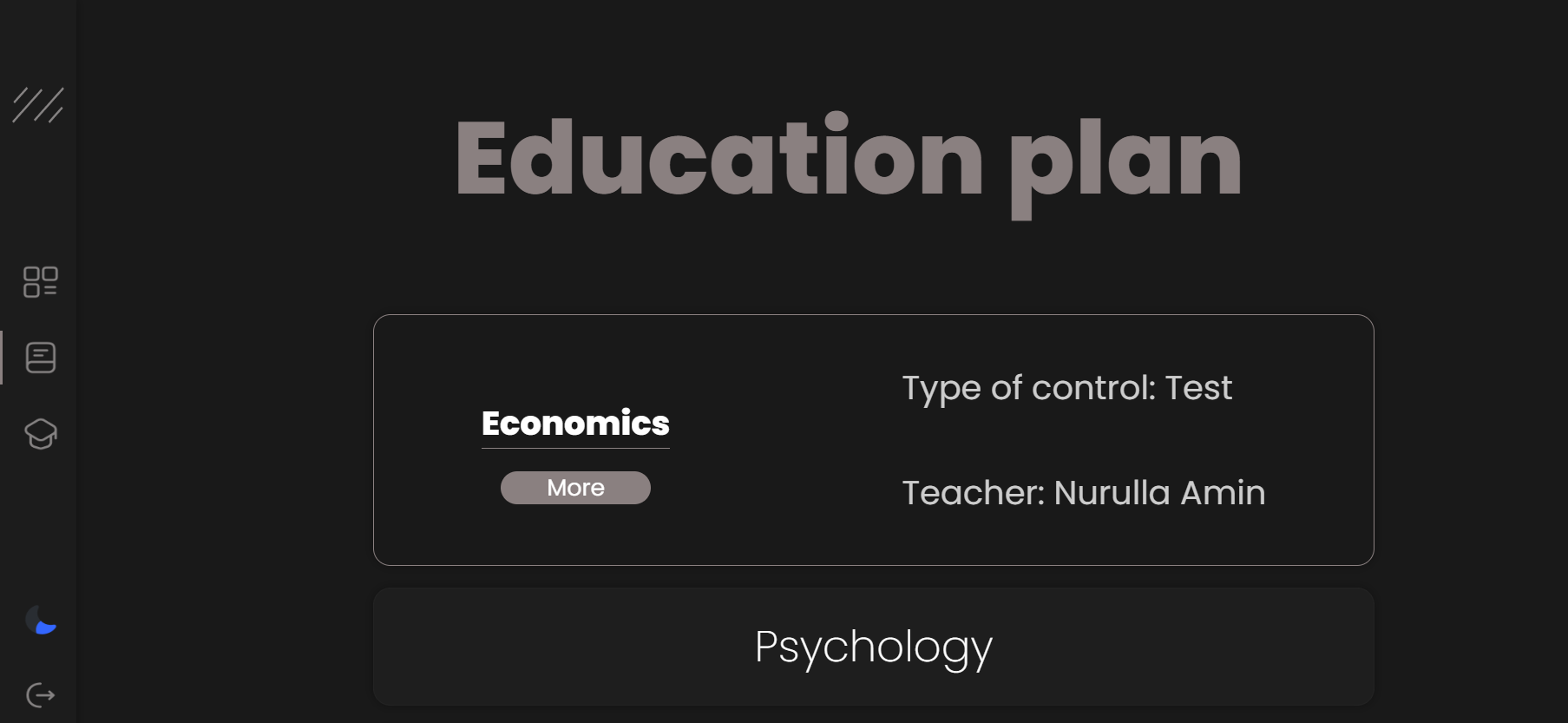


Рисунок 6.2.3 – Страница текущей траектории обучения

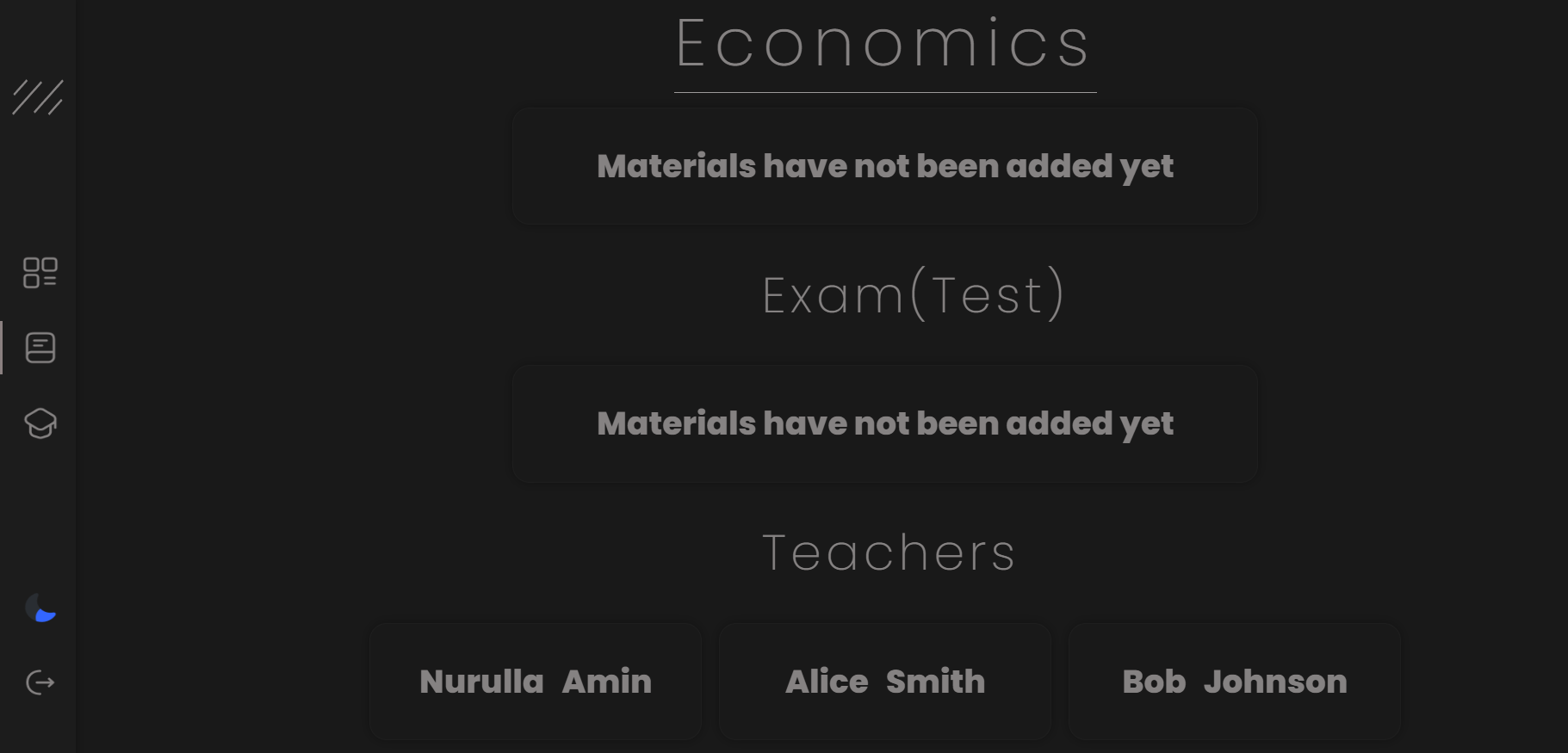


Рисунок 6.2.4 – Страница определенного предмета

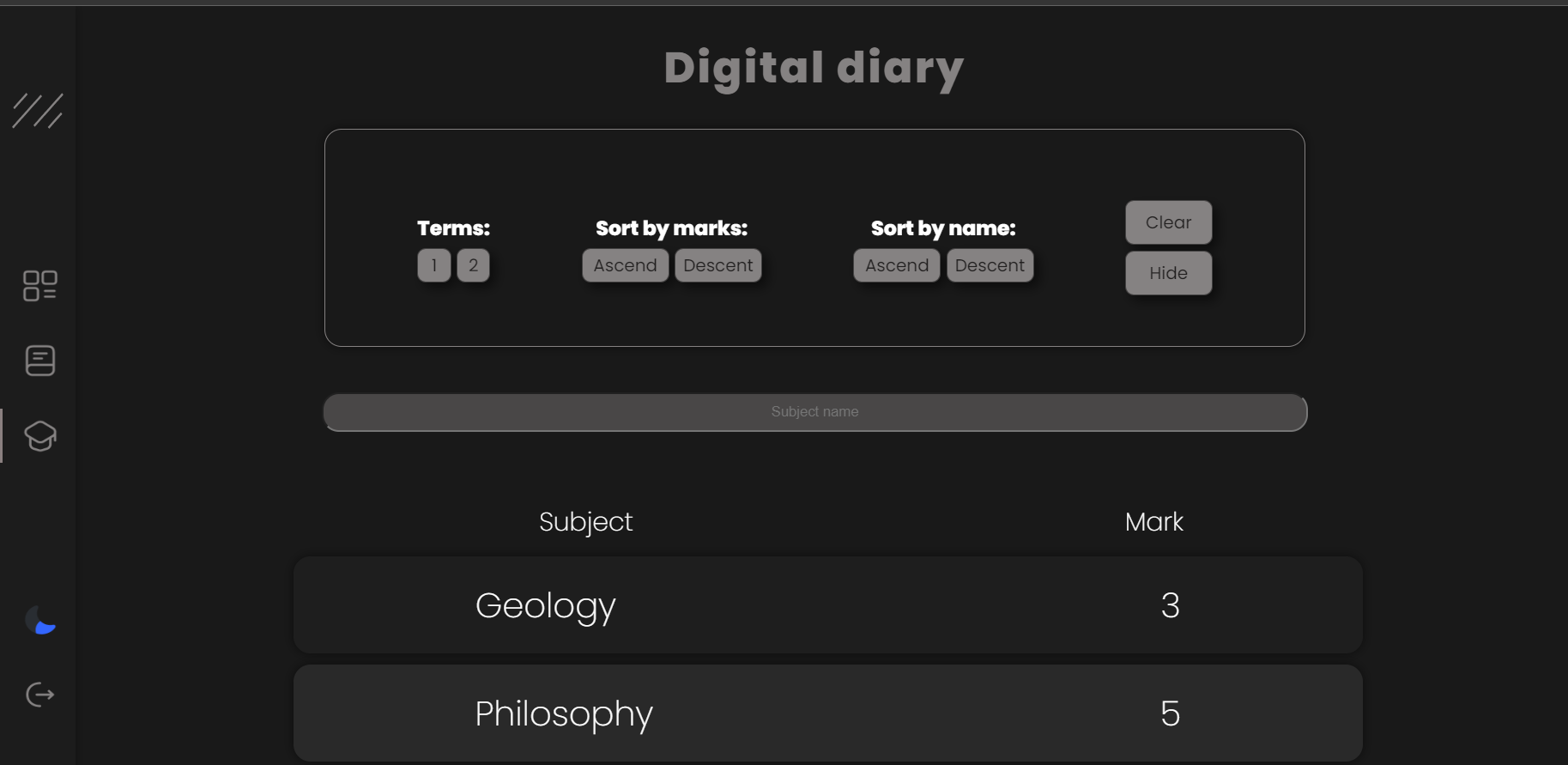


Рисунок 6.2.5 – Электронная зачетная книжка пользователя

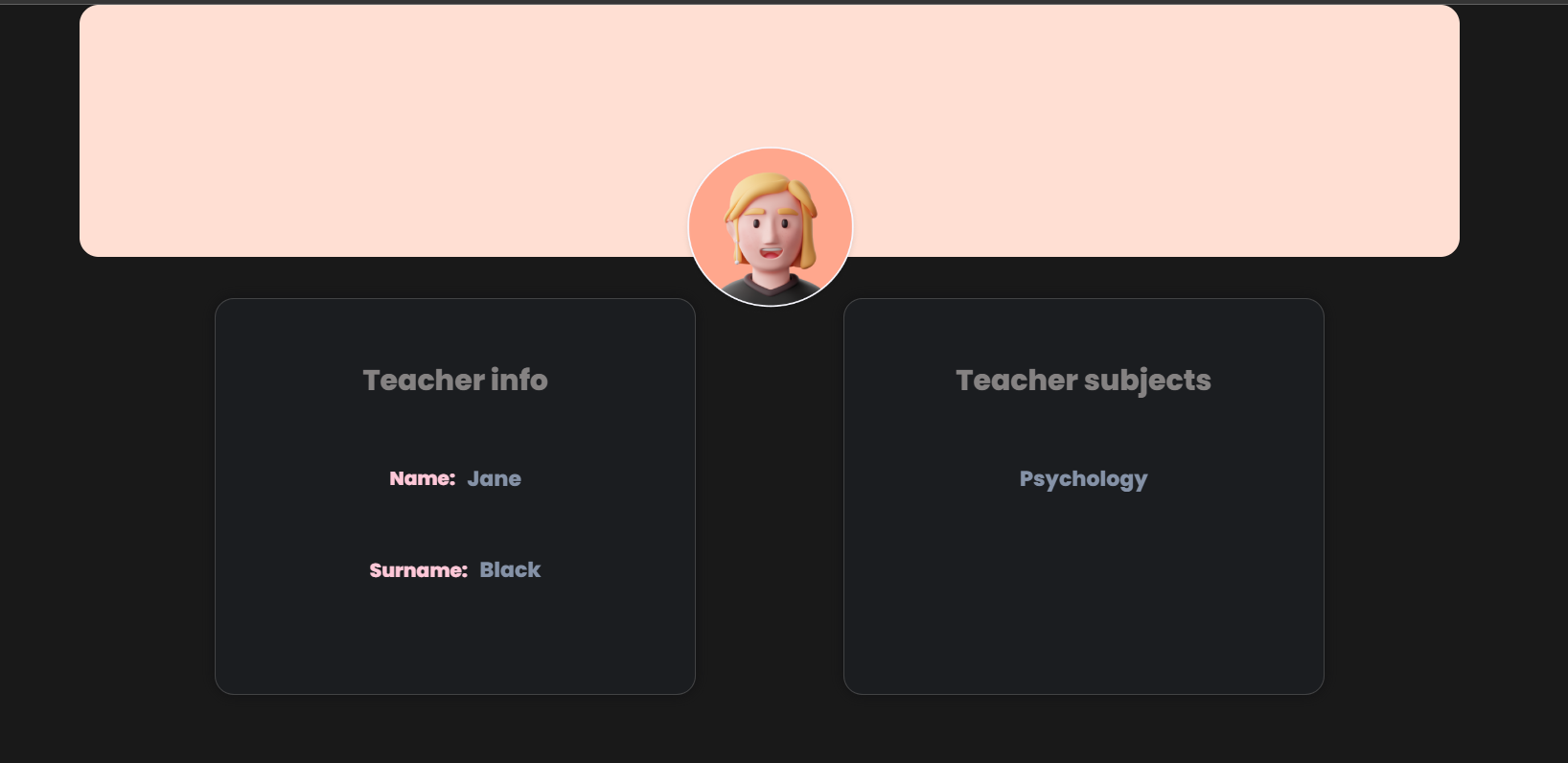


Рисунок 6.2.6 – Профиль преподавателя

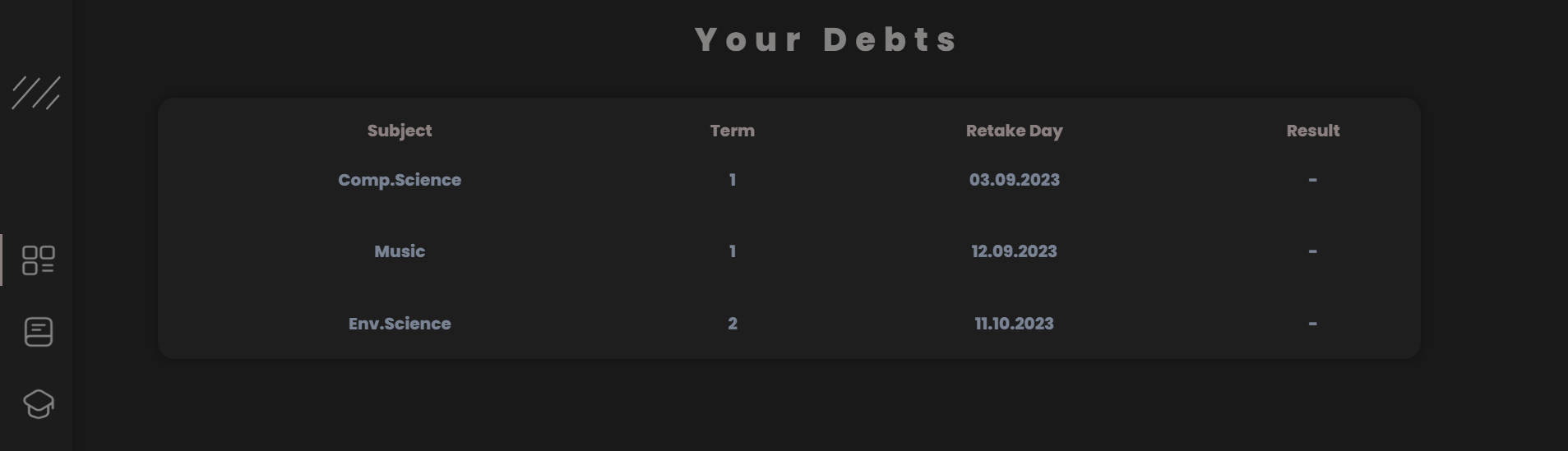


Рисунок 6.2.7 – Страница с информацией о задолженностях пользователя

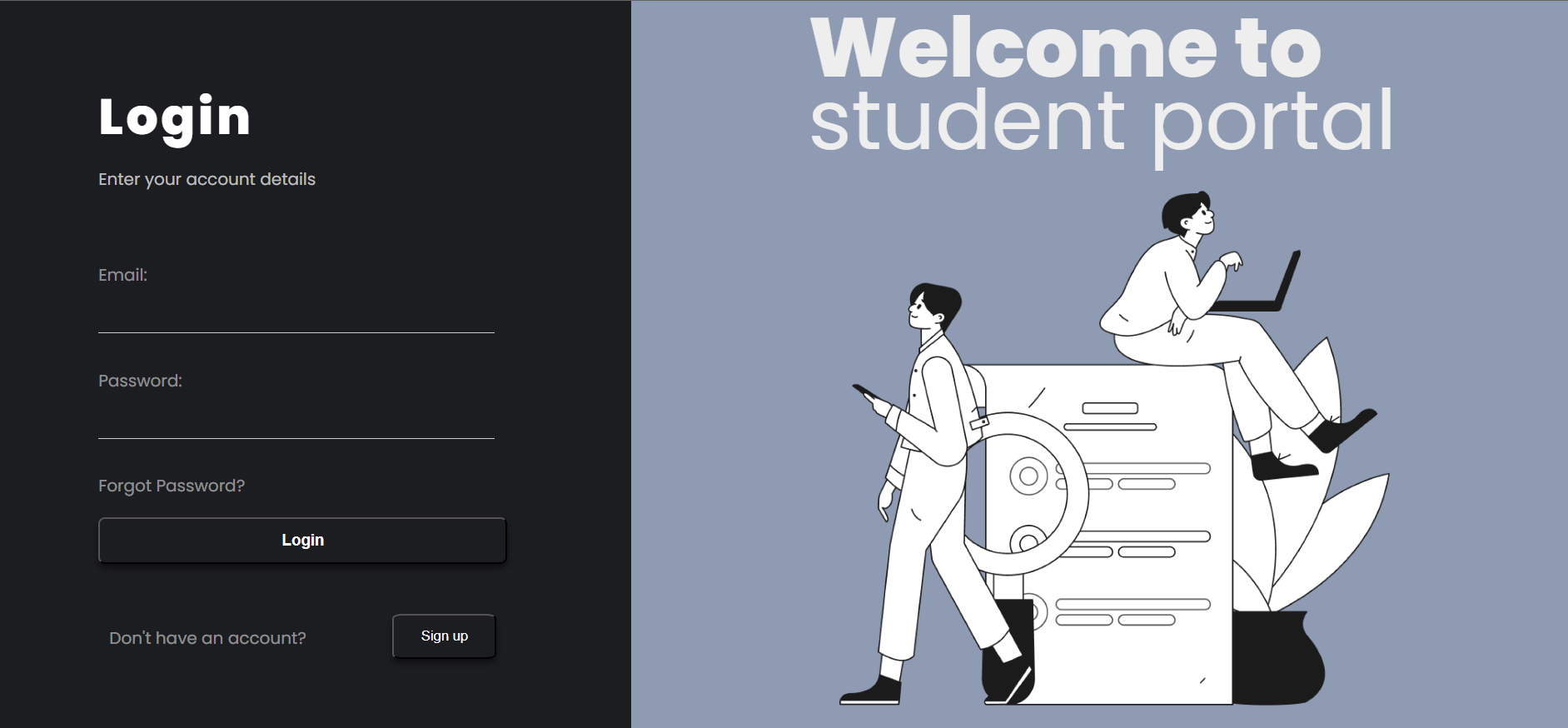


Рисунок 6.2.8 – Страница входа в систему

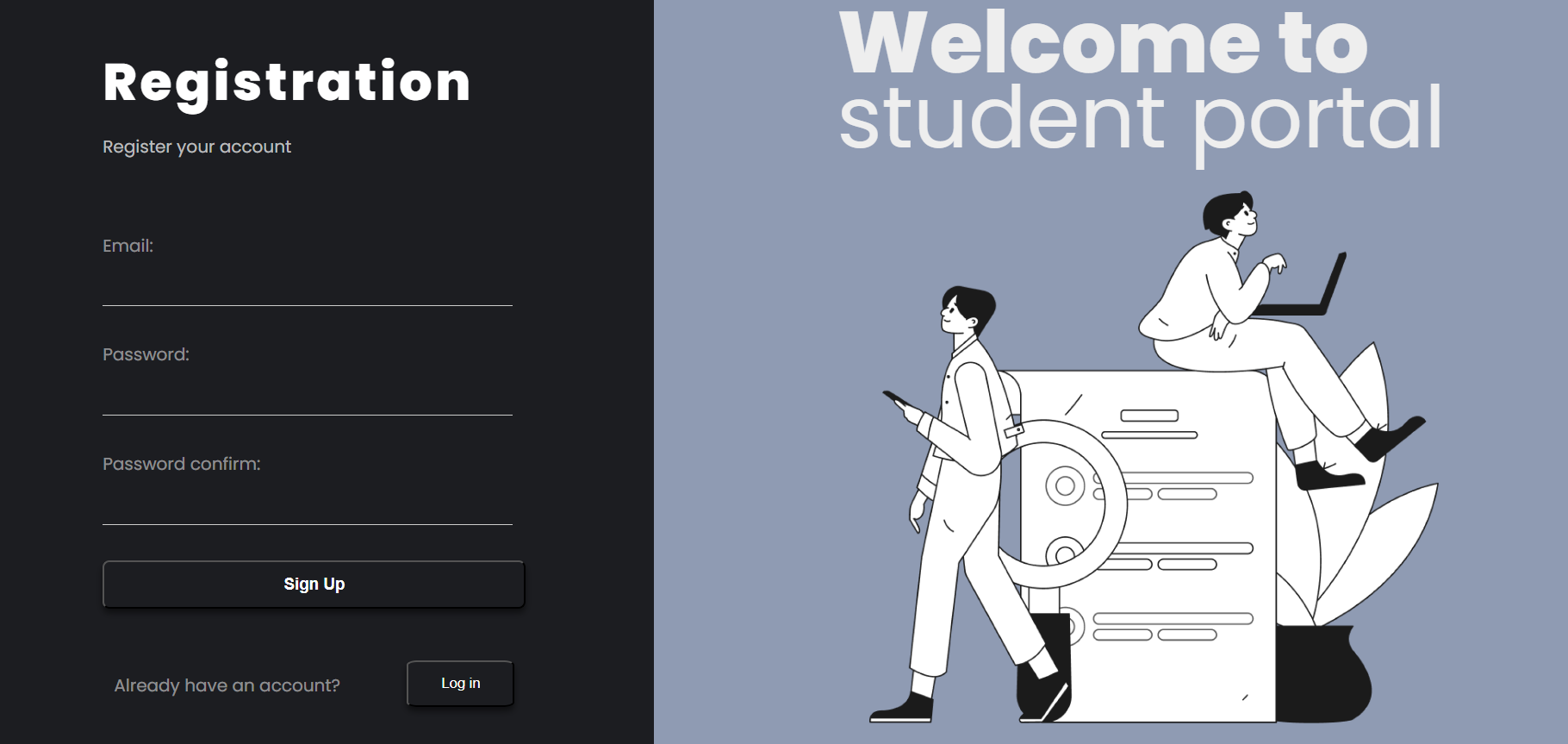


Рисунок 6.2.9 – Страница создания нового профиля пользователя

1. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ
   1. Тестирование приложения с помощью Insomnia

Проверим работу обработчика POST запроса на авторизацию пользователя - рисунок 7.1.1.

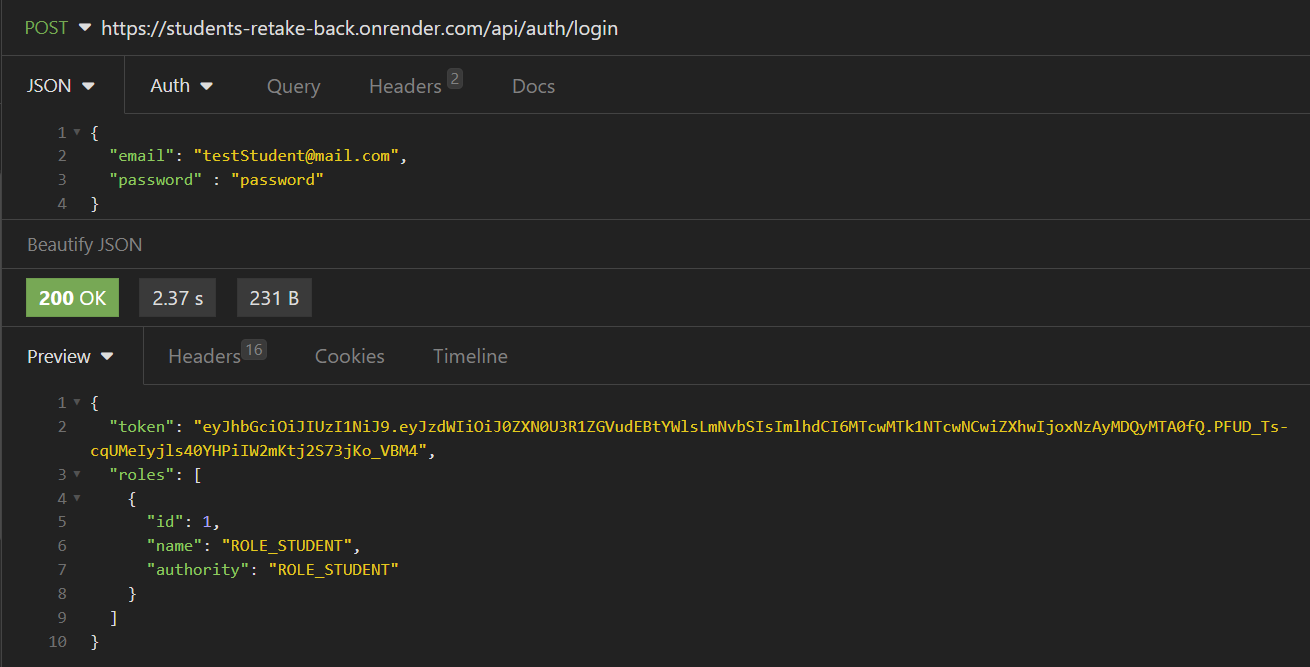


Рисунок 7.1.1 – POST запроса на авторизацию пользователя

Результатом отправки POST запроса является ответ от сервера с кодом 200, который говорит об успешном выполнении запроса. Также в теле ответа был передан token для аутентификации пользователя.

Далее протестируем обработку GET запроса на получение данных о пользователе – рисунок 7.1.2. В теле запроса присутствует token для того, чтобы сервер понял, что запрос от авторизованного пользователя.

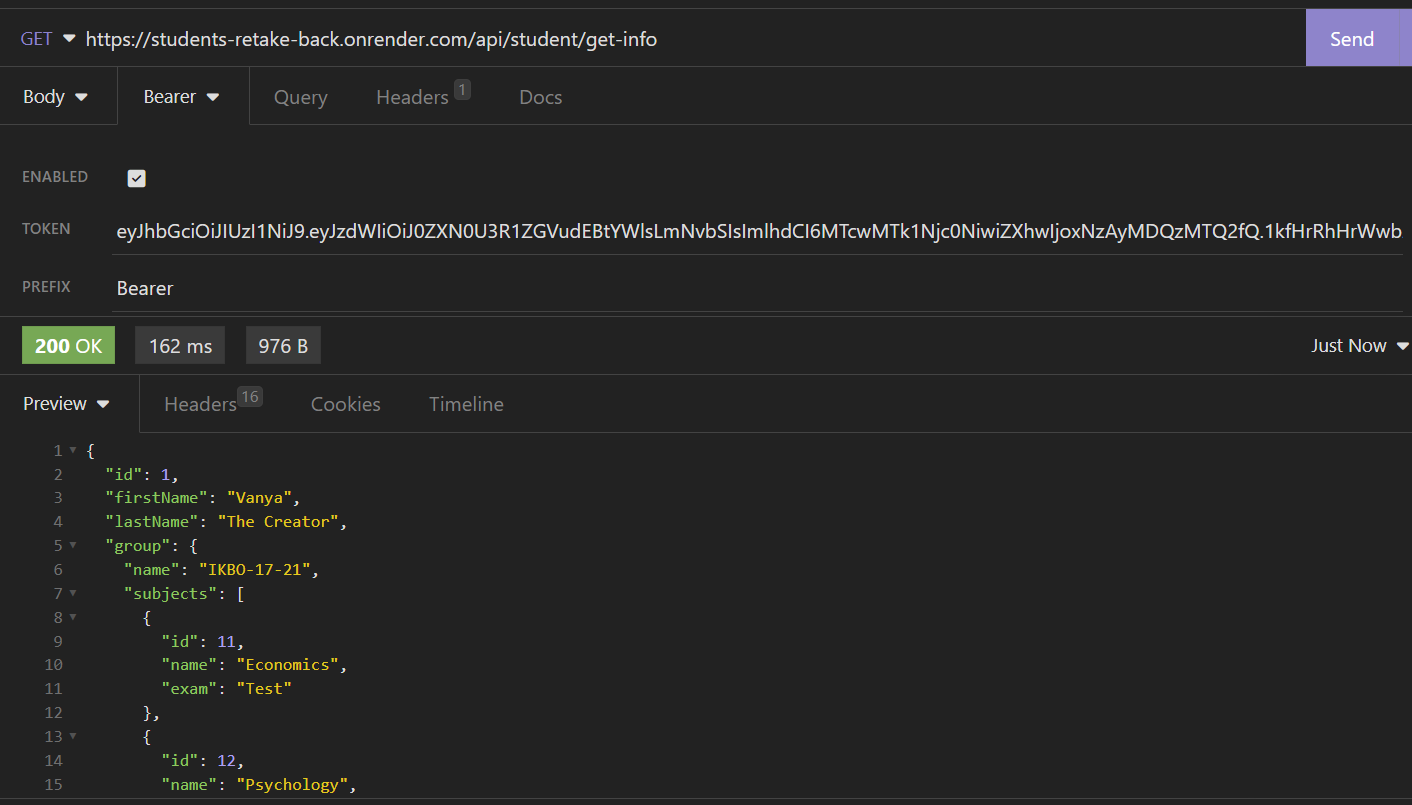


Рисунок 7.1.2 – Ответ на GET запрос данных о студенте

Протестируем работу обработчика GET запроса на получения списка предметов – рисунок 7.1.3.

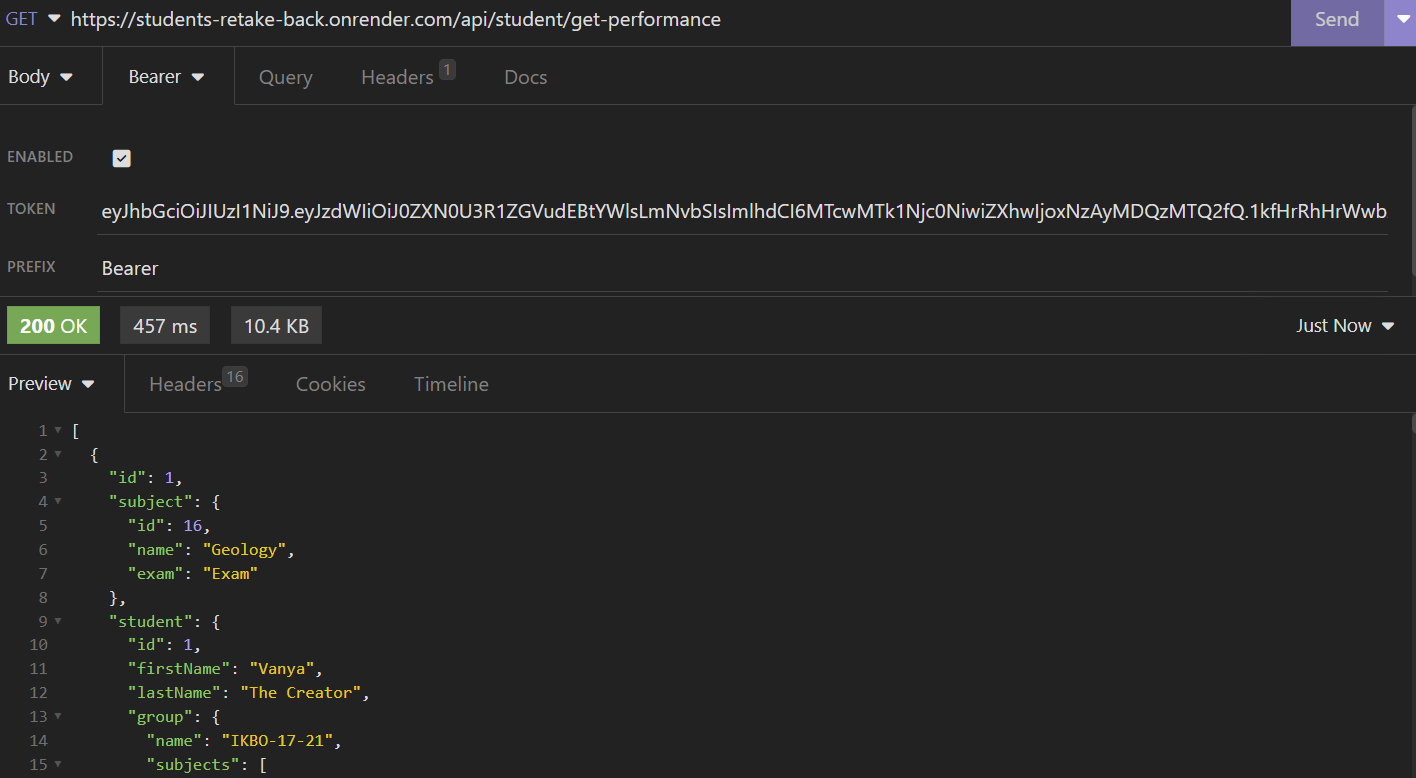


Рисунок 7.1.3 – Отправка GET запроса на получение данных о предметах

Проверим работу обработчика GET запроса конкретного предмета по его идентификатору – рисунок 7.1.4.

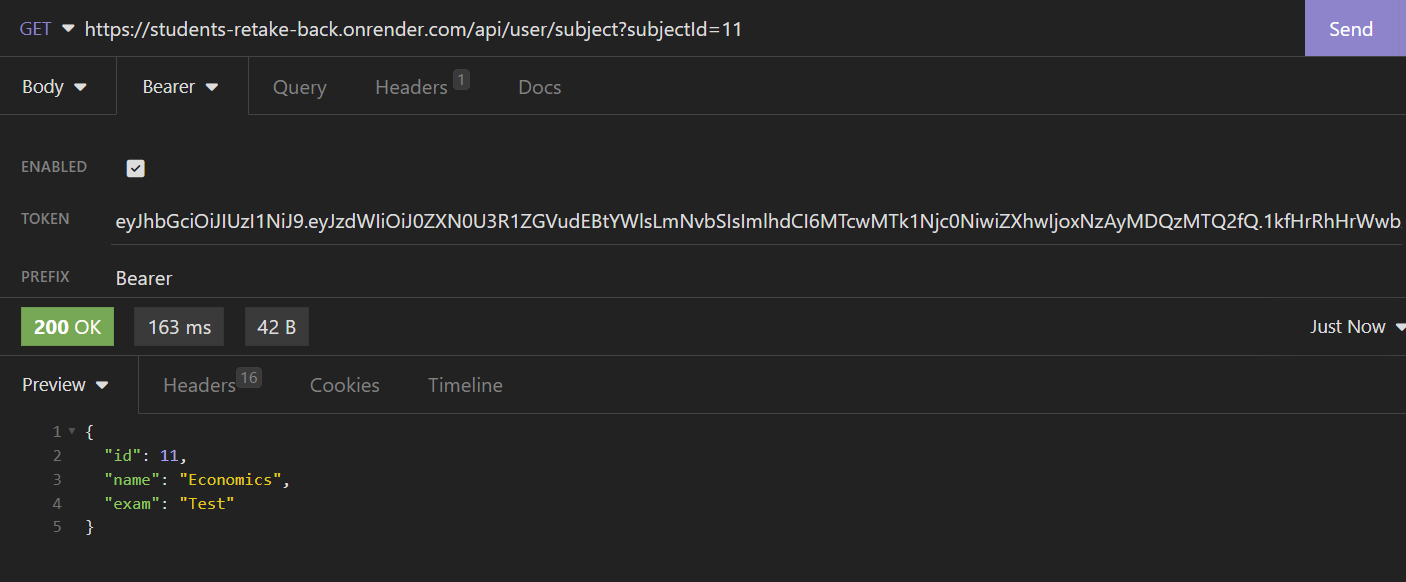


Рисунок 7.1.4 – Отправка GET запроса на получение данных о конкретном предмете

Протестируем обработчик POST запроса на регистрацию нового пользователя – рисунок 7.1.5.

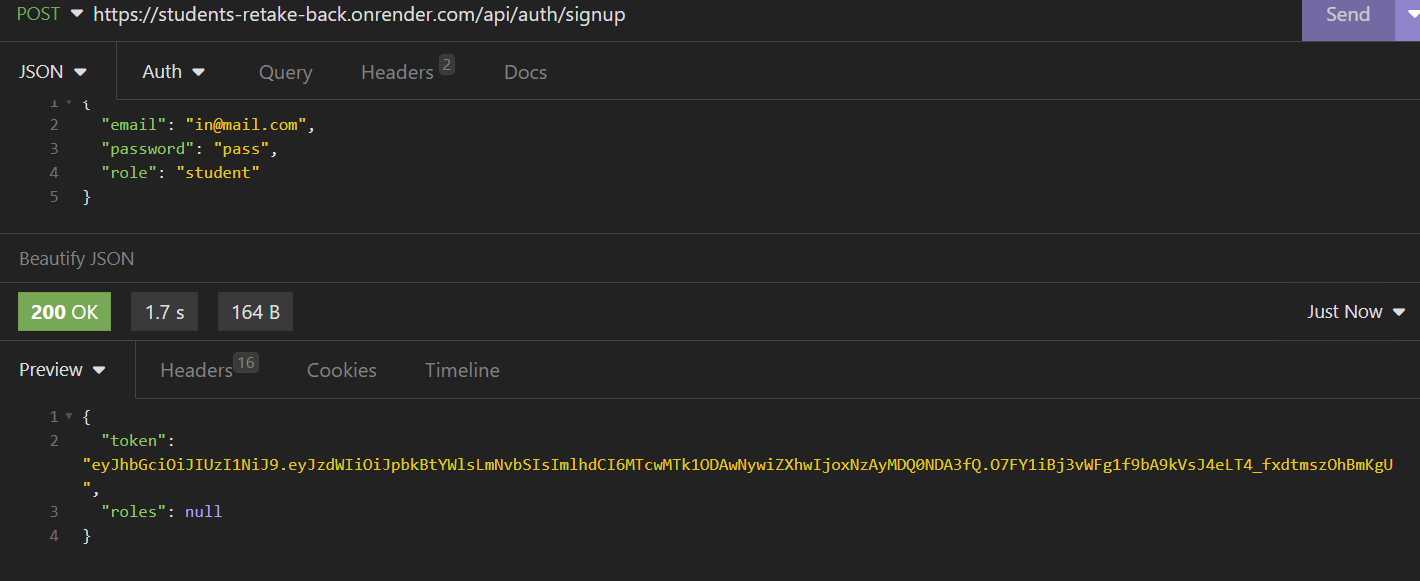


Рисунок 7.1.5 – Ответ на POST запрос создания нового пользователя

ЗАКЛЮЧЕНИЕ

Результатом выполнения курсовой работы на тему "Аддитивный микропортал контроля пересдач для студентов" стало разработанное серверное программное приложение, которое написано с использованием языка Java и фреймворка Spring, включая Spring Data JPA, Spring Security, Spring Core и Spring MVC. Для создания серверного приложения требовались навыки отличного владения и знания всех нюансов среды разработки, в которой написана работа, поэтому в ходе ее выполнения приобретены навыки работы в InteliJ IDEA. Изучен фреймворк Spring, который предназначен для разработки приложений на языке Java. Он предоставляет широкий набор инструментов и библиотек для упрощения процесса разработки, повышения производительности и улучшения масштабируемости приложений. Также изучена библиотека JavaScript – React. Она позволяет создавать компоненты, которые описывают структуру и внешний вид элементов интерфейса, и управлять ими динамически.

Благодаря глубокому анализу предметной области, учтены и проработаны все недочеты веб-ресурсов с похожей тематикой. Разработанный веб-ресурс содержит необходимые элементы, такие как изображения, текст и визуальное оформление, которые совместимы между собой и обеспечивают единообразие стилей и шрифтов на всех страницах, а также имеет всю функциональность.

Учитывая все вышеперечисленные пункты, можно с уверенностью сказать, что все требования, поставленные в задании на курсовую работу, соблюдены и выполнены успешно, цель работы достигнута. Разработанное приложение является полноценным и функциональным инструментом для отслеживания успеваемости студентов.

Исходный код Интернет-ресурса по курсовой работе доступен по ссылке: [https://github.com/Res-lay/students-retake](https://github.com/Res-lay/students-retake%20)

Обратится к разработанному интернет-ресурсу можно по ссылке: <https://student-retake-web.vercel.app/>

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Документация Spring Framework [Электронный ресурс]. – URL: [https://docs.spring.io/spring-framework/docs/current/reference/html/core.html](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html%20) (дата обращения: 20.11.2023).
2. Документация Spring Boot [Электронный ресурс] – URL: [https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/%20) (дата обращения: 20.11.2023).
3. Руководство по Spring MVC [Электронный ресурс] – URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html> (дата обращения: 22.11.2023).
4. Руководство по Spring Security [Электронный ресурс] – URL: <https://spring.io/guides/gs/securing-web/> (дата обращения: 24.11.2023).
5. Руководство по Spring Data JPA [Электронный ресурс] – URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> (дата обращения: 25.11.2023).
6. Статья на Habr «Введение в Spring Boot» [Электронный ресурс] – URL: <https://habr.com/ru/articles/435144/> (дата обращения: 25.11.2023).
7. Статья на Medium «Building a simple RESTful API with Spring boot» [Электронный ресурс] – URL: <https://medium.com/nerd-for-tech/building-a-simple-restful-api-with-spring-boot-2351687ecab0> (дата обращения: 27.11.2023).
8. Статья на Habr «Рекомендации по REST API» [Электронный ресурс] – URL: <https://habr.com/ru/articles/483374/> (дата обращения: 27.11.2023).
9. Руководство по создания RESTful API на Java [Электронный ресурс] – URL: <https://spring.io/guides/tutorials/rest/> (дата обращения: 27.11.2023).
10. Руководство по тестированию RESTful API на Medium [Электронный ресурс] – URL: <https://medium.com/@oyetoketoby80/how-to-write-unit-test-for-your-rest-api-f8f71376273f> (дата обращения: 29.11.2023).
11. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 5-е изд.. – СПб.: Питер, 2021 – 768 с.
12. Руководство по использованию JSON Web Tokens (JWT) [Электронный ресурс] – URL: <https://habr.com/ru/articles/533868/> (дата обращения: 01.12.2023).
13. Spring Security JWT Tutorial [Электронный ресурс] – URL: <https://www.toptal.com/spring/spring-security-tutorial> (дата обращения: 01.05.2023).
14. Диков А. Клиентские технологии веб-дизайна. HTML5 и CSS3. Учебное пособие для вузов. 2-е изд.. – Санкт-Петербург: Лань, 2023 – 188 с.
15. Видео – курс на YouTube «Spring Security Tutorial» [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=b9O9NI-RJ3o&t=2825s> (дата обращения: 28.11.2023).
16. Руководство по PostgreSQL [Электронный ресурс] – URL: [https://www.postgresql.org/docs/](https://www.postgresql.org/docs/%20) (дата обращения: 29.11.2023).
17. Уоллс К. Spring в действии – М.: ДМК Пресс, 2013. – 752 c
18. Мардан А. React быстро. Веб-приложения на React, JSX, Redux и GraphQL – СПб.: Питер, 2019. – 560 c.
19. Раджпут Д. Spring. Все паттерны проектирования. – СПб.: Питер, 2019 – 320 с.
20. Хоффман Э. Безопасность веб-приложений – СПб.: Питер, 2021. – 354 c.