МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка web-приложения для системы контроля успеваемости СПО

Студент _____ Лебединский Данил Константинович
(подпись, дата) (ФИО полностью)

Руководитель _____ Попова Н. А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«25» декабря 2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Лебединский Данил Константинович гр. 20ВИ1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Разработка web-приложения для системы контроля успеваемости СПО

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

2. Руководитель работы_____

3. Задание на работу (назначение разработки, исходные данные и т.п.) Разработать web-приложение для системы контроля успеваемости СПО. Система должна выполнять следующие функции: авторизация пользователя, создание пользователя, выставление баллов студенту, просмотр успеваемости.

4. Перечень подлежащих разработке вопросов Пояснительная записка должна содержать: предпроектное обследование предметной области, технико-экономическое обоснование целесообразности разработки системы, техническое задание, разработку функциональной модели, разработку модели данных, описание программы, программу и методику испытаний, результаты испытаний, описание применения

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка 106 листов, 58 рисунков, 28 таблиц, 5 приложений.

ОЦЕНИВАНИЕ, БАЛЛЫ, ГРУППА, ФАКУЛЬТЕТ, КАФЕДРА, ПРЕДМЕТ, СПЕЦИАЛЬНОСЬ, УСПЕВАИМОСТЬ, ВЕБ, WEB, ПРИЛОЖЕНИЕ.

Объект работы: автоматизация деятельности образовательного учреждения по контролю успеваемости.

Цель работы – разработать web-приложение приложение для системы контроля успеваемости СПО.

В результате проделанной работы разработано web-приложение приложение для системы контроля успеваемости СПО, с помощью которого можно вести учет успеваемости студентов СПО.

При разработке приложения использовались среда NodeJS и React.

База данных спроектирована с помощью инструментального средства ERwin. База данных разработана для СУБД PostgreSQL 13.

Эффективность разработки заключается в снижении трудовых и временных затрат на проведение учета успеваемости за студентами.

Содержание

Введение 7
1 Общий раздел 8
1.1 Анализ предметной области 8
1.2 Концепция и содержание проекта 14
1.3 Постановка задачи 18
1.4 Анализ нефункциональных требований 21
2 Планирование и расчет бюджета разрабатываемой системы 22
3 Проектирование программного обеспечения 27
3.1 Выбор технологий и инструментальных средств разработки 27
3.2 Проектирование архитектуры программы 29
3.3 Диаграмма деятельности 29
3.4 Проектирование базы данных 31
3.4.1 Концептуальное проектирование базы данных 31
3.4.2 Логическое проектирование базы данных 32
3.5 Проектирование серверной части приложения 34
3.6 Проектирование интерфейса пользователя 34
4 Реализация программного обеспечения 39
4.1 Реализация базы данных 39
4.2 Реализация серверной части приложения 45
4.3 Реализация клиентской части 46
5 Тестирование программного обеспечения 48
5.1 Тестирование серверной части приложения 48
5.2 Тестирование клиентской части приложения 50
5.3 Метрики кода 51
5.3.1 Метрики размера программы 52
5.3.2 Метрики Холстеда 53
Заключение 55
Список используемых источников 56

Приложение А (рекомендуемое) Модели данных 58
Приложение Б (рекомендуемое) Код клиентской части 61
Приложение В (рекомендуемое) Код серверной части 77
Приложение Г (рекомендуемое) Результаты тестирования серверной части 91
Приложение Д (рекомендуемое) Результаты тестирования серверной части .. 103

Введение

Система контроля успеваемости студентов среднего профессионального образования играет решающую роль в обеспечении качества образования и поддержании высоких академических стандартов. Традиционные методы ведения учета успеваемости, такие как использование бумажных журналов и ручных расчетов, могут быть трудоемкими, подверженными ошибкам и не обеспечивать эффективных средств для отслеживания и анализа данных студентов.

Цифровизация процессов контроля успеваемости становится все более распространенной, поскольку она предлагает многочисленные преимущества, такие как автоматизация, точность и улучшенная отчетность. В этой работе предлагается разработка веб-приложения для системы контроля успеваемости студентов среднего профессионального образования, которая позволит автоматизировать ключевые процессы и повысить эффективность управления успеваемостью.

Веб-приложение будет разработано с использованием передовых технологий и будет ориентировано на удовлетворение потребностей как преподавателей, так и студентов. Оно обеспечит надежную и удобную платформу для ведения учета успеваемости, расчета оценок, отслеживания посещаемости и предоставления своевременной обратной связи студентам. Цель данной работы – создать решение, которое оптимизирует процессы контроля успеваемости, поддерживает академический рост студентов и облегчает принятие решений преподавателями и администрацией.

В соответствии с целью были поставлены следующие задачи:

- изучить предметную область;
- спроектировать программное обеспечение;
- разработать пользовательский интерфейс;
- разработать серверную часть сервиса;
- протестировать работу веб-сервиса.

1 Общий раздел

1.1 Анализ предметной области

Система контроля успеваемости студентов нужна как для студентов, чтобы они могли отслеживать результаты своей учебной деятельности, так и для работников учебных заведений, чтобы они могли выставлять оценки или баллы студентов.

Для такой системы важно, чтобы она была в постоянном доступе у каждого пользователя. Поэтому, имеет смысл разрабатывать именно веб-приложение для его дальнейшего развертывания в сети интернет. Реализация сайта позволяет довести взаимодействие с пользователями до автоматизма, при этом повышая эффективность и надежность обмена информацией.

Для создания сайта, соответствующего заданному направлению, необходимо проанализировать ресурсы схожей тематики, выделить аспекты, на основе которых, станет возможным сформировать общую структуру проекта.

В качестве первого аналога был выбран сайт «moodle», который использует большое количество университетов. Его главная страница приведена на рисунке 1.

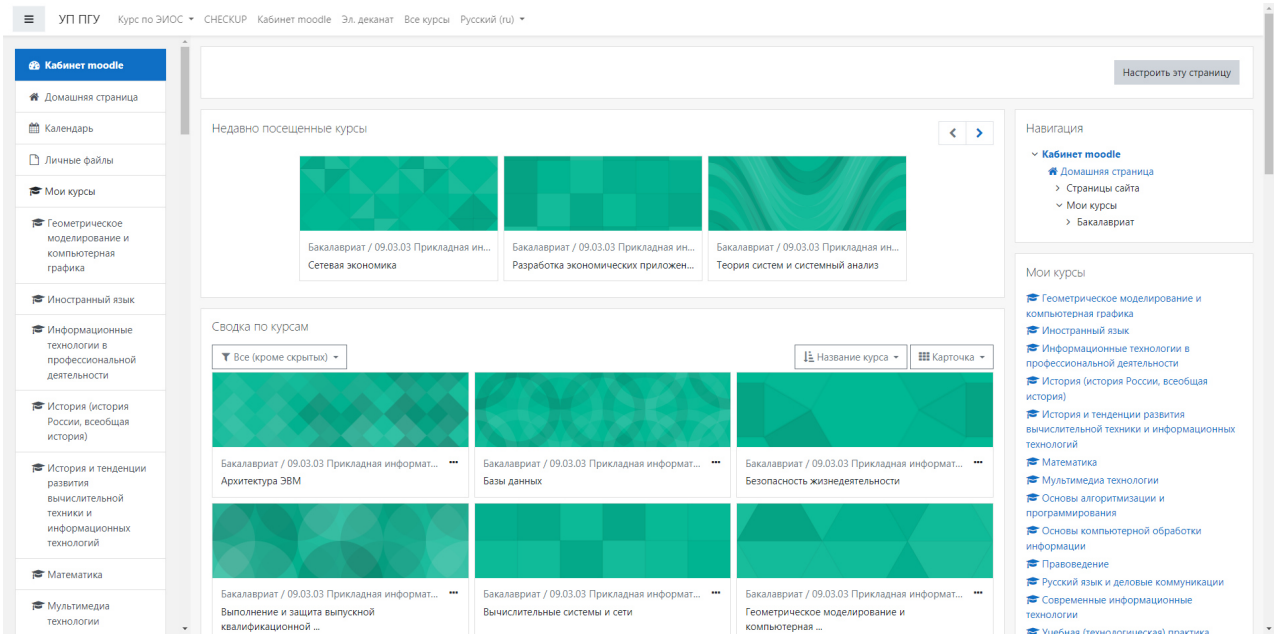


Рисунок 1 – Главная страница сайта «moodle»

На главной странице вверху расположена шапка сайта, на которой в свою очередь представлены ссылки на другие разделы. Удобная навигация по сайту является важной частью пользовательского интерфейса, именно поэтому данный раздел можно взять референса при разработке шапки сайта.

По центру находится несколько панелей: «Недавно посещенные курсы» и «Сводка по курсам» для более удобной навигации по сайту.

Для того что посмотреть оценки, нужно нажать на ссылку «Оценки», после чего откроется страница, на которой будут отображены результаты всех учебных дисциплин. Страница с оценками по учебным дисциплинам приведена на рисунке 2.

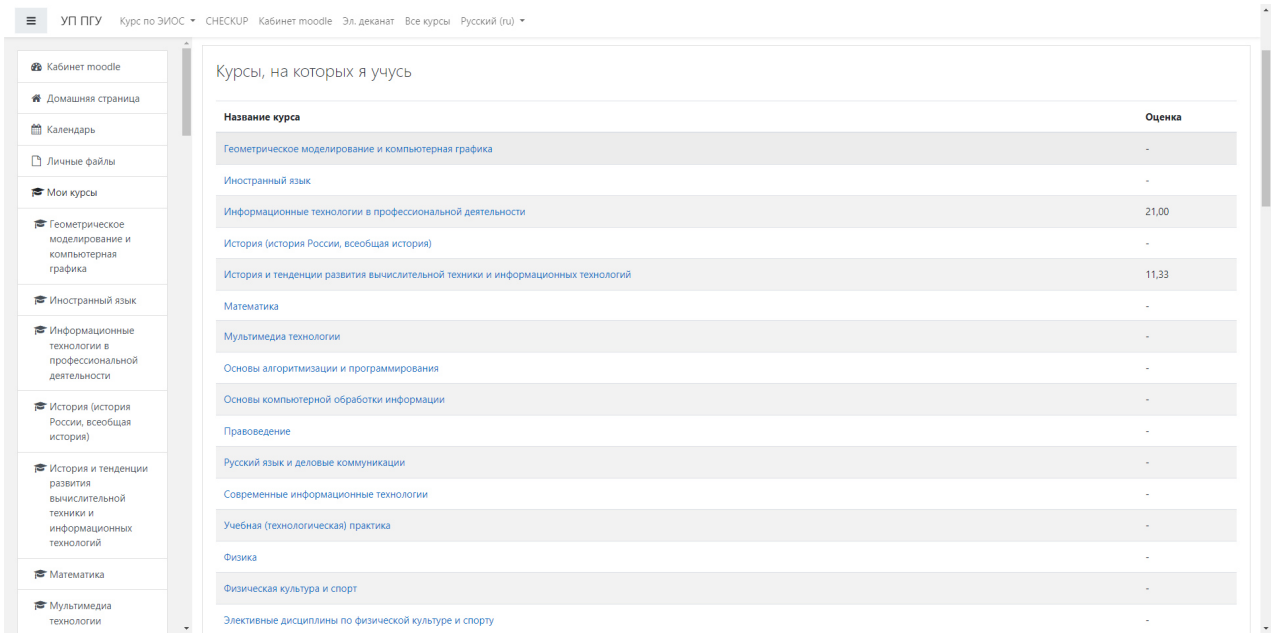
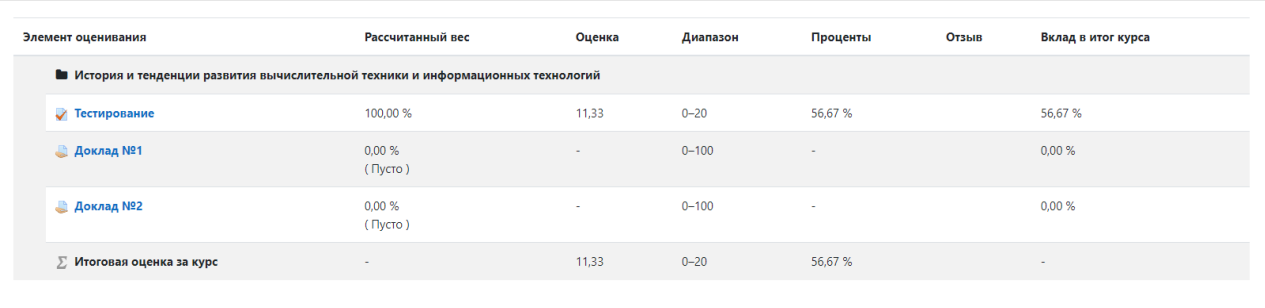


Рисунок 2 – Страница с оценками по учебным дисциплинам

Эту страницу можно считать плохо реализованной, так она представляет собой своеобразную «кучу» учебных дисциплин, у которую нельзя сортировать ни по курсам, ни по алфавиту

Также, нажав на ссылку с названием предмета, откроется страница с более подробным описанием оценивания студента по данной учебной дисциплине, что можно отнести к плюсом данного сайта. Страница с подробным описанием оценивания приведена на рисунке 3.



Элемент оценивания	Рассчитанный вес	Оценка	Диапазон	Проценты	Отзыв	Вклад в итог курса
История и тенденции развития вычислительной техники и информационных технологий						
Тестирование	100,00 %	11,33	0–20	56,67 %		56,67 %
Доклад №1	0,00 % (Пусто)	-	0–100	-		0,00 %
Доклад №2	0,00 % (Пусто)	-	0–100	-		0,00 %
Итоговая оценка за курс	-	11,33	0–20	56,67 %		-

Рисунок 3 – Страница с подробным описанием оценивания по дисциплине

Дизайн сайта использует палитру цветовой схемы по минимуму, что является логичным решением, исходя из тематики сайта. Основными цветами являются синий и темно-зеленый, который используется в кнопках, блоках и прочих элементах. Вся остальная страница имеет белый либо серый фон и черный цвет шрифта. Такое сочетание цветов вызывает доверие у пользователей и выражает профессионализм.

Для анализа используемых технологий на сайте был использован сервис 2IP. Данный сервис помогает определить, при помощи какой системы управления контентом был создан и продолжает администрироваться сайт. Если же при разработке была использована собственная система, то ресурс 2IP сообщит о том, что при анализе сайта признаков использования той или иной CMS обнаружено не было. Результат представлен на рисунке 4.

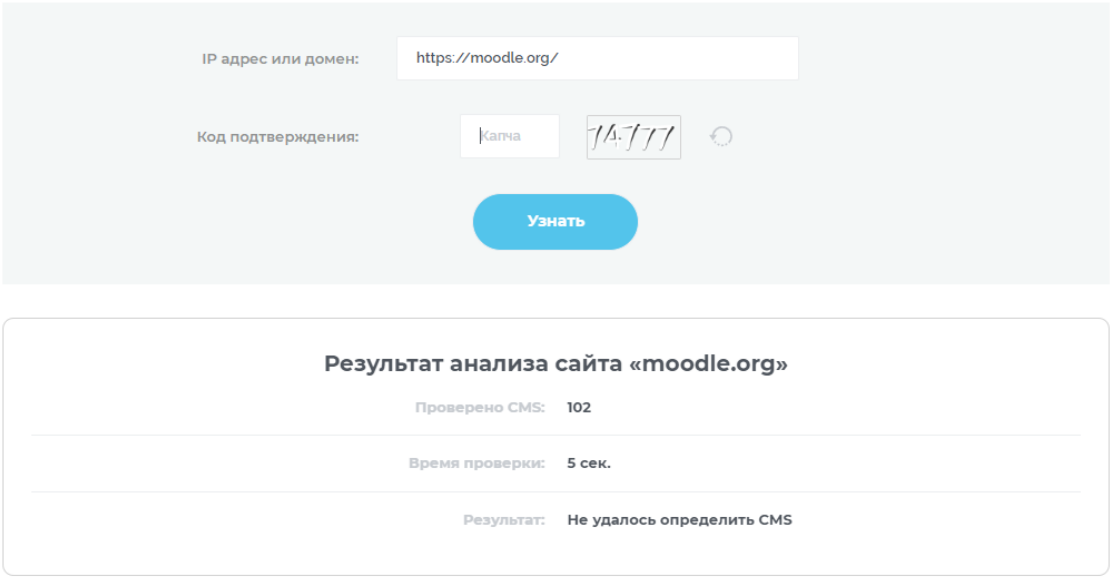


Рисунок 4 – Определение системы управления сайтом с помощью сервиса 2IP

Согласно результатам сканирования, система выявила, что сайт администрируется при помощи собственной системы.

Еще одним способом анализа технологий, которые были применены при разработке сайта является непосредственный просмотр кода страницы.

Код, выполняемый на сервере, отследить невозможно в виду причин, связанных с организацией того, как выполняется код, но некоторые файлы подключаются внутри самого HTML-документа, который уже можно просмотреть.

По общим правилам дополнительные плагины и файлы подключаются в конце HTML кода. Таким образом, по крайней мере, некоторую часть подключенных плагинов можно просмотреть.

Зачастую этими плагинами являются сторонние подключаемые библиотеки, необходимые для сбора статистических данных, либо же SEO-оптимизации, что в свою очередь поднимает рейтинг сайта для поисковых машин. Отрывок из html кода страницы представлен на рисунке 5.

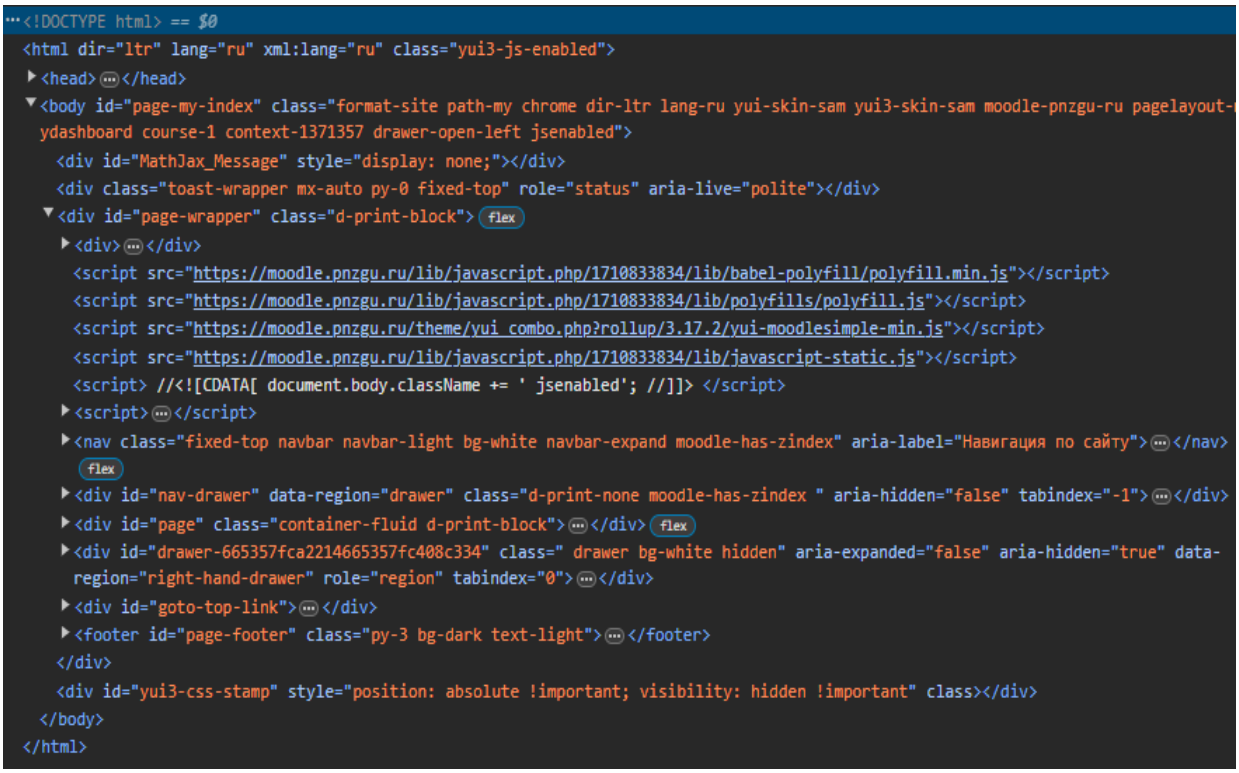


Рисунок 5 – Часть HTML кода сайта «moodle»

Исходя из информации, которая представлена на рисунке, в коде видно использование нескольких подключаемых файлов, которые предназначены для сбора статистики о действиях пользователя на сайте.

Подобная статистика помогает принимать дальнейшие решения, связанные с модернизацией интернет-ресурса, а иногда и всего предприятия в целом.

Вторым сайтом для анализа был выбран интернет-ресурс «iSpring learn», главная страница которого представлена на рисунке 6.

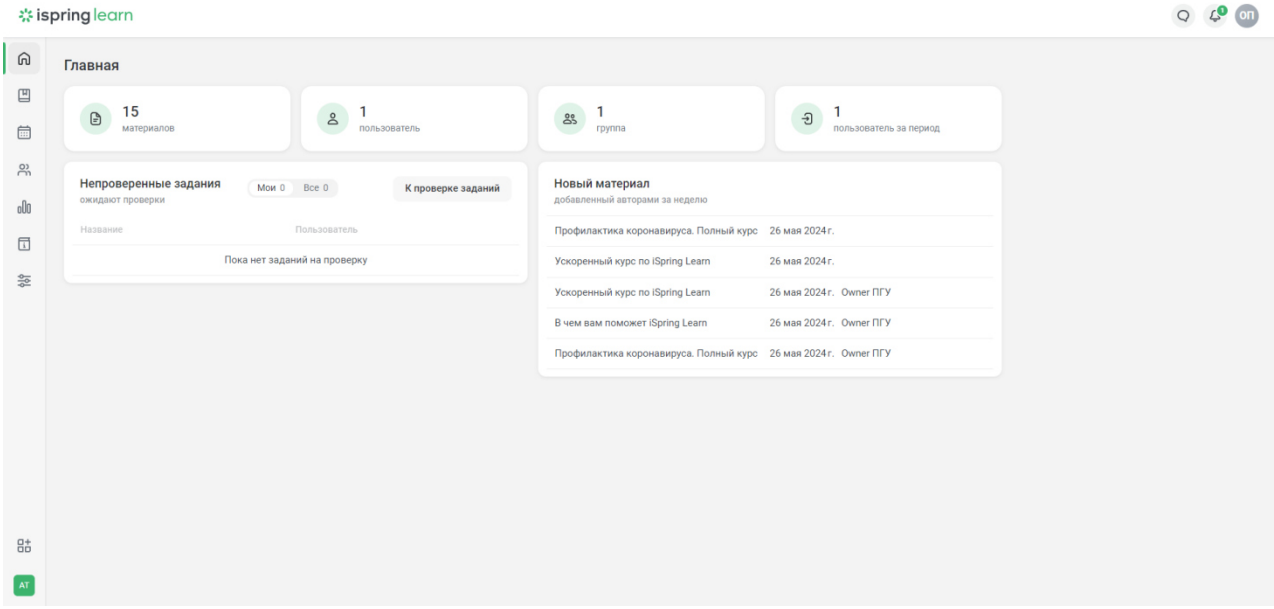


Рисунок 6 – Главная страница сайта «iSpring learn»

Данный сайт, по сравнению с прошлым, выделяется своим современным дизайном. Минималистичный и плавный стиль с отвечающими современным стандартам иконками создает ощущение, что ты пользуешься качественным продуктом. Эта особенность будет взята за основу дизайна разрабатываемого проекта.

В шапке находится логотип сайта, который ведет на главную страницу сайта, кнопки, которые открывают окно с сообщения и оповещения соответственно.

Футер у данного сайта отсутствует, так как в нем смысла в такой системе (футер расположен на рекламном лендинге данного веб-приложения).

Слева находится навигационная панель, которая ведет по различным частям сайта. Ссылки на эти страницы реализованы в виде кнопок, на которых изображены иконки, соответствующие определенному разделу сайта. Данная панель будет служить референсом при разработке нашего приложения. Среди них нет страницы, на которой можно поставить оценку обучающемуся, либо посмотреть оценку, что является минусом данного интернет-ресурса.

В центре главной страницы расположено несколько блоков с основной информацией, среди нее: запланированные задания, новые учебные материалы, количество материалов, количество пользователей созданной локальной системы, и количество групп.

Для определения системы управления контентом, используемой на сайте, вновь был задействован сервис 2IP. Результат анализа представлен на рисунке 7.

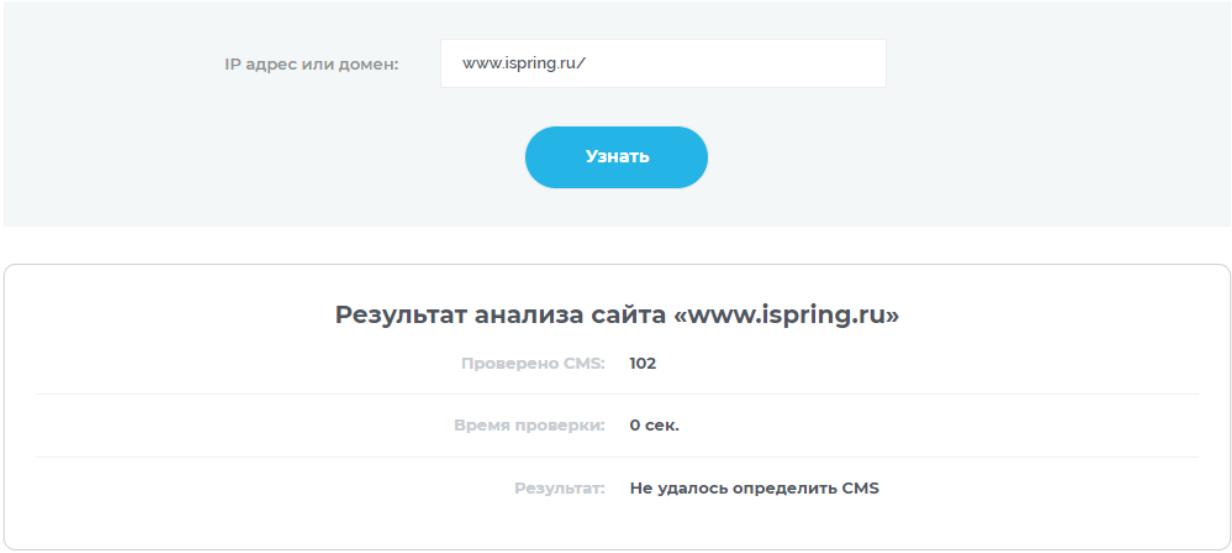


Рисунок 7 – Результат анализа сайта «iSpring learn»

Согласно результатам сканирования, система выявила, что сайт администрируется при помощи собственной системы.

Подводя итоги анализа предметной области систем контроля успеваемости студентов, становится возможным сделать общий вывод об общих требованиях и принципах построения дизайна и функционала пользовательского интерфейса.

Прежде всего, главная страница должна иметь понятное и информативное навигационное меню, на котором должны быть представлены логотип и ссылки на разделы сайта, а также кнопка, перенаправляющая на страницу авторизации.

Важным дополнением является возможность получения доступа к элементам меню навигации на любом этапе прокрутки сайта. То есть шапка сайта должна быть зафиксирована и менять свое состояние в зависимости от того, на каком этапе просмотра контента пользователь сейчас находится.

Пользователь должен иметь возможность быстро находить соответствующий раздел и осуществлять поиск информации о статусе заказа.

1.2 Концепция и содержание проекта

Исходя из анализа предметной области, можно сделать выводы о том, какой вид должна иметь функциональная часть сайта. А именно какие задачи станет возможным решить пользователю, преподавателю и администратору.

Прежде всего, главная страница будет представлять собой информационную страницу, на которой будет представлена основная информация о проходящих или преподающих учебных дисциплинах.

Помимо этого, в проекте должна присутствовать система авторизации пользователей, причем регистрацию пользователей должен проводить администратор, чтобы доступ к системе имели только люди, имеющие к ней отношение. Вводимые при создании профиля данные должны проходить процедуру валидации и проверки на наличие в базе данных, таким образом, чтобы под уже существующим логином или адресом электронной почты было невозможно зарегистрироваться. Зарегистрированные пользователи будут иметь возможность просмотра своей успеваемости по проходимым учебным дисциплинам.

Наличие личного кабинета является важным элементом разрабатываемой системы. С его помощью пользователь должен иметь возможность отслеживать информацию, связанную с ним.

С точки зрения удобства пользования, наличие простого и понятного меню, а также грамотного наполнения контентом является важным пунктом, который стоит учитывать при разработке дизайна и верстке сайта.

Преимущества и достоинства создаваемого проекта включают в себя возможность автоматизировать процесс оценивания студента за счет удобной системы ведения отчетности, благодаря которой информацию о тех или иных совершенных действиях может быстро получить как пользователь, так и преподаватель.

При разработке сайта всегда стоит опираться на то, каким образом он будет попадать в поисковую выдачу. Создание постоянного притока новых клиентов является одной из основных задач, с которыми справляются сайты компаний.

Большая часть посетителей сайта попадают туда при вводе запросов в поисковых машинах. Таким образом, важно определить ключевые слова, которые будут использоваться в мета-тегах, либо в содержании контента сайта, для охвата как можно большего количества людей, но при этом, важным является и тот факт, что сайт должен подходить под тематику запроса, заданного пользователем, иначе неэффективное использование ключевых слов только понизит рейтинг сайта в поисковой выдаче, а также будет терять посетителей, не нашедших на сайте нужной информации, которую они искали.

Для анализа ключевых слов можно воспользоваться специальными сервисами. Одним из таких сервисов является Yandex.Wordstat, также называемый подбор слов Яндекса. Это специальный сервис для анализа заинтересованности пользователей к тем или иным запросам, основанный на собранной статистике поисковых запросов. Результаты анализа запроса «успеваемость студентов» представлены на рисунке 8.

Формулировка	Число запросов
успеваемость студентов	10 252
оценка успеваемости студента	2 785
таблица успеваемости студента	2 574
успеваемость студентов вуза	2 378
система успеваемость студента	2 319
результаты успеваемости студентов	2 139
система оценки успеваемости студентов	1 875
успеваемость группы студентов	1 782
учет успеваемости студентов	804
средняя успеваемость студента	542
учебная успеваемость студентов	441
средний балл успеваемости студента	402
академическая успеваемость студентов	368
контроль успеваемости студентов	328
исследования успеваемости студентов	306
программа успеваемость студентов	288
анализ успеваемости студентов	266
уровень успеваемости студента	265
база данных успеваемость студентов	256
журнал успеваемости студентов	237
система учета успеваемости студентов	221
текущая успеваемость студентов	206

Рисунок 8 – Статистика по словам запроса «успеваемость студентов»

На основе представленных вариантов следует сделать выводы какие из них более точно подходят под описание того, что представлено на сайте. Например, вариант «база данных успеваемость студентов» не является подходящим, так как на сайте не все смогут посмотреть успеваемость студентов.

Для грамотного продвижения сайта простого случайного перебора ключевых слов недостаточно. В целях корректного индексирования в поисковой выдаче необходимо провести кластеризацию семантического ядра.

Кластеризация семантического ядра – это разделение ключевых слов на группы, которые в последствии будут использоваться для оптимизации страниц на сайте. Такое разделение позволяет эффективнее удерживать потенциального клиента на странице, корректнее прорабатывать структуру информационного ресурса, а также способствует увеличению притока пользователей на сайт. Основная стадия кластеризации чаще всего проводится с помощью специализированных сервисов. В данном случае в роли этого сервиса выступил сайт coolakov.ru. Результаты группировки запросов представлены на рисунке 9.

№	Запросы	Запросов в кластере	Конкуренты
1	система успеваемости студентов колледжа система контроля успеваемости студентов колледжа система успеваемости студентов система контроля успеваемости студентов	4	Показать/скрыть
2	успеваемость студентов успеваемость среди студентов успеваемость студентов университетов	3	Показать/скрыть
3	система успеваемости студентов университета система контроля успеваемости студентов университета	2	Показать/скрыть
4	успеваемость студентов колледжей	1	Показать/скрыть

Рисунок 9 – Результаты кластеризации

Согласно результатам группировки, можно заметить, что несколько из пунктов представляет собой абсолютно не связанные с тематикой сайта запросы. Такие запросы в рамках кластеризации считаются нецелевым вхождением.

Это значит, что эффективность продвижения сайта упадет, в связи с тем, что пользователь, зашедший по запросу «успеваемость студентов колледжей», попадая на сайт, на котором нет данной информации, покинет ресурс, не найдя нужной информации. Большую роль в принципах, по которым поисковые машины выдают результаты играет поисковый интент.

Интент – это смысл, или же намерение, которое пользователь вкладывает в поисковую фразу, именно на его основе оценивается релевантность тех или иных запросов.

Одним из требований, предъявляемых к разрабатываемому сайту, является реализация системы, в которой администратор будет создавать профили для преподавателей и студентов для доступа с ресурсу.

Исходя из этого следует, что пользователи сайта будут делиться на группы, различающиеся по доступному им функционалу. Таким образом, необходимо заранее предусмотреть какие возможности будут у того или иного пользователя в зависимости от прав доступа.

1.3 Постановка задачи

Диаграмма вариантов использования отображает отношения между действующими ролями и прецедентами, которые так же называются вариантами использования. Прецеденты представляют собой последовательность действий, который может осуществлять система, взаимодействуя с действующими лицами. В качестве отношений между прецедентами выступают: расширение, которое является разновидностью зависимости между базовым вариантом использования и его особым случаем, и включение, в котором в отличии от расширения, функциональное поведение всегда задействуется базовым сценарием использования. Диаграмма вариантов использования представлена на рисунке 10.

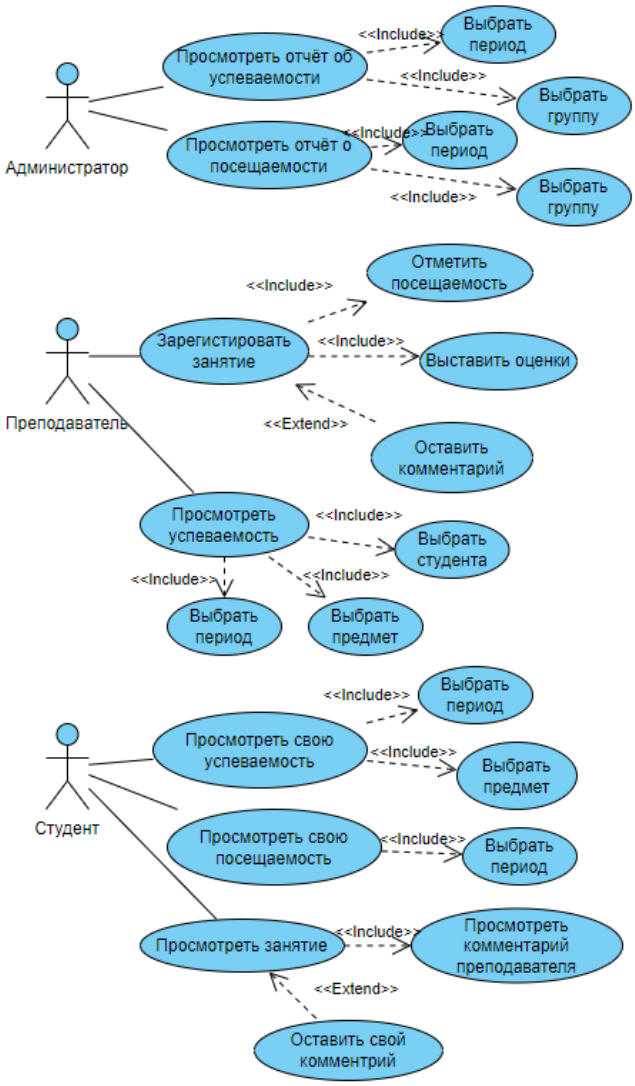


Рисунок 10 – Диаграмма вариантов использования

На диаграмме вариантов использования представлен сценарий для трех типов пользователей:

Студент, просмотреть свою успеваемость, выбрать период просмотра успеваемости, выбрать предмет для просмотра по нему успеваемости, посмотреть свою посещаемость, выбрать период просмотра посещаемости, просмотреть занятие, посмотреть комментарий преподавателя по занятию, оставить свои комментарий по занятию.

Преподаватель, зарегистрировать занятие, отметётесь посещаемость, выставить оценки, оставить комментарий по занятию, просмотреть успеваемость, выбрать студента для просмотра его успеваемости, выбрать предмет и выбрать период для просмотра успеваемости.

Администратор, посмотреть отчет об успеваемости, выбрать период для отчета об успеваемости, выбрать группа для отчета об успеваемости, посмотреть отчет о посещаемости, выбрать период для отчета о посещаемости, выбрать группа для отчета о посещаемости.

Помимо этого, для удобства и интуитивной понятности пользовательского интерфейса, необходимо описать пользовательский сценарий, чтобы в последующем, опираясь на него, определить требования к поведению системы. Пользовательский сценарий представлен на рисунке 11.

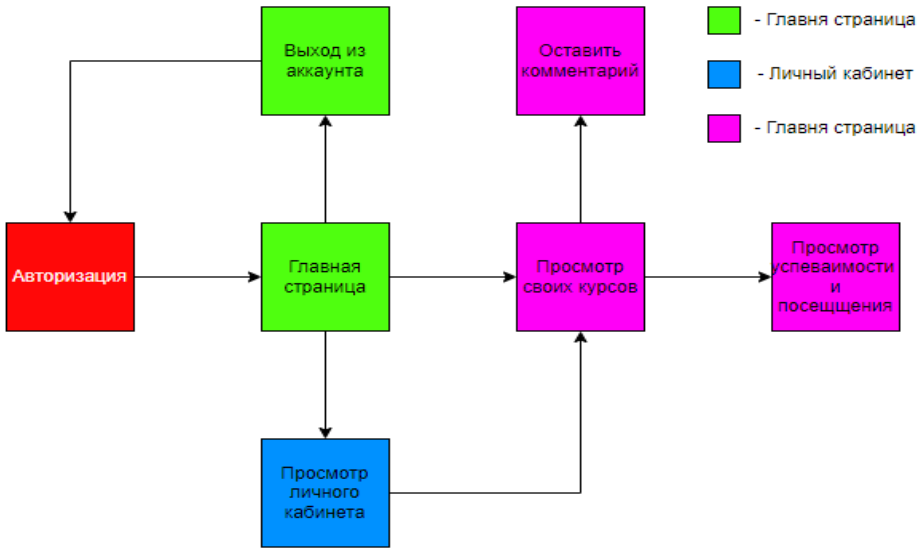


Рисунок 11 – Пользовательский сценарий

Прежде всего, пользователь, вошедший на сайт, должен пройти процедуру авторизации, чтобы получить доступ к системе. После авторизации, пользователь попадает на главную страницу.

После попадания на главную струнцу, пользователь может переходить в различные разделы веб-приложения согласно пользовательскому сценарию.

Несмотря на всю свою полезность, одного пользовательского сценария недостаточно для формирования структуры при разработке веб-приложений. Для решения подобных задач подходит карта сайта, которая представляет собой своеобразное содержание сайта с полным список страниц и их разделов.

Зачастую карты применяются для поисковых систем, которые в свою очередь используют эту информацию для более однонаправленного ответа на поисковый запрос, что в свою очередь позволяет точечно выдать пользователю именно то, что он желает найти.

Помимо этого, карта сайта является отличным дополнительным элементом навигации, в моменты, когда представленного на сайте меню недостаточно. Это рационально, так как размещать полноценную карту сайту в качестве меню навигации в шапке страницы навредит дизайну и общей интуитивной понятности меню навигации для гостя. Именно поэтому подобные карты представляются в виде блока и располагаются там, где они будут уместнее, например многие разработчики располагают карту либо в подвале сайта, либо и вовсе уделяют отдельную страницу на ресурсе. Схематическая карта сайта представлена на рисунке 12.

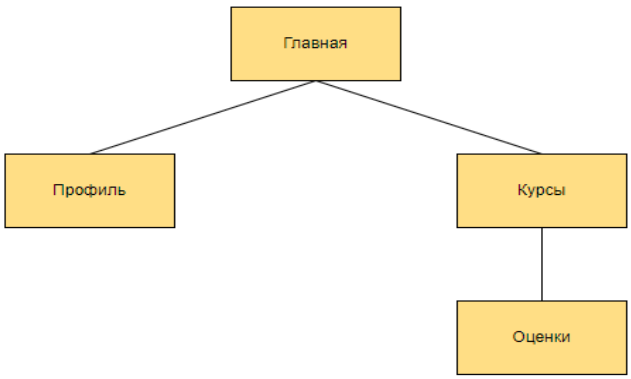


Рисунок 12 – Схематическая карта

Согласно представленной на рисунке 11 карте сайта все страницы связаны с главной, так как она является их связующим звеном. Все разделы, связанные в первую очередь с ней, являются главными страницами своего раздела, что свидетельствует о наличии упорядоченной группировки на категории.

Таким образом, следует выстраивать структуру проекта согласно карте сайта, а учет результатов, полученных при создании пользовательского сценария и диаграммы вариантов использования необходим для расставления верных приоритетов при проектировании пользовательского интерфейса.

1.4 Анализ нефункциональных требований

На данном этапе были определены нефункциональные требования к разрабатываемому программному обеспечению. Должны быть использованы такие технологии разработки, как FP, MVC.

Серверная часть приложения должна быть разработана на языке программирования JavaScript на платформе NodeJS в среде программирования WebStorm 2023.3.2. Данные на сервере должны храниться в базе данных, реализованной с использованием СУБД PostgreSQL 13, запросы к базе данных должны осуществляться на языке SQL с использованием библиотеки Express. Серверная часть должна работать под управлением операционной системы Windows 10 и выше.

Клиентская часть приложения должна быть реализована на языке программирования JavaScript в среде программирования WebStorm 2023.3.2. Она должна представлять собой веб-приложение с удобным и интуитивно понятным пользовательским интерфейсом. Клиентское приложение должно быть отлажено и должно успешно работать под управлением операционной системы Windows 10 / Linux / Mac OS.

2 Планирование и расчет бюджета разрабатываемой системы

Для успешного управления проектами требуется определить конечные цели и обеспечить их выполнение, путем планирования необходимых ресурсов, времени, качества и бюджета проекта, а также проведения оценки возможных рисков и трудозатрат. Наличие подробного плана заранее также играет ключевую роль в обеспечении эффективного проектного управления.

Для планирования и расчета бюджета разработки web-приложения для системы контроля успеваемости СПО необходимо пройти этапы, перечисленные ниже:

– определить объем и спецификации функциональных требований;

– оценить трудозатраты и затраты на разработку системы;

– составить план работ, расписание и бюджет, на основе оценки трудозатрат и затрат;

– оценить экономическую эффективность системы.

Существует несколько формализованных методов для планирования и управления выполнением проектных работ. Один из основных методов – это построение линейного графика Ганта.

Диаграмма Ганта – это инструмент визуализации временных интервалов выполнения задач и проектов. Она позволяет представить графически все этапы проекта, от начала до конца, и определить продолжительность каждой задачи.

Назначение диаграммы Ганта – помочь определить общую продолжительность проекта, сроки выполнения каждой задачи, количество ресурсов, необходимых для выполнения проекта, а также выявить возможные риски и проблемы, которые могут возникнуть в процессе выполнения задач.

Построение диаграммы Ганта для осуществления планирования и расчета бюджета разрабатываемой системы выполнялось в web-сервисе для управления проектами GanttPRO.

В ходе планирования проекта было выделено 25 задач, описание которых представлено в таблице 1.

Таблица 1 – Задачи проекта

№ п/п	Название задачи	Длительность	Начало	Окончание	Предшественники
1	Начало проекта	0 дней	12.04.2024	16.04.2024	
2	Анализ предметной области и требований к системе	4 дня	12.04.2024	16.04.2024	
3	Анализ бизнес-процессов предприятия	1 день	12.04.2024	12.04.2024	
4	Поиск и анализ информационных систем аналогичного назначения	1 день	13.04.2024	13.04.2024	3
5	Анализ требований и постановка задач	1 день	14.04.2024	14.04.2024	4
6	Выбор технологий проектирования и разработки	1 день	15.04.2024	15.04.2024	5
7	Завершение анализа предметной области и требований к системе	0 дней	16.04.2024	16.04.2024	2
8	Проектирование информационной системы	10 дней	17.04.2024	26.04.2024	7
9	Проектирование логической модели базы данных	4 дня	17.04.2024	20.04.2024	
10	Проектирование структуры данных	2 дня	21.04.2024	22.04.2024	9
11	Проектирование пользовательского интерфейса	4 дня	23.04.2024	26.04.2024	10
12	Завершение проектирования информационной системы	0 дней	26.04.2024	26.04.2024	8
13	Реализация информационной системы	13 дней	27.04.2024	11.05.2024	12
14	Реализация конфигурации	5 дня	27.04.2024	01.05.2024	
15	Реализация веб-интерфейса системы	8 дней	03.05.2024	11.05.2024	14
16	Завершение реализации информационной системы	0 дней	11.05.2024	11.05.2024	13
17	Тестирование программных средств	10 дней	12.05.2024	21.05.2024	16
18	Тестирование метрикой Холстеда	6 дней	12.05.2024	17.05.2024	
19	Функциональное тестирование	4 дня	18.05.2024	21.05.2024	
20	Завершение тестирования программных средств	0 дней	21.05.2024	21.05.2024	17
21	Оформление документации	11 дней	22.05.2024	01.06.2024	20
22	Написание документации	7 дней	22.05.2024	28.05.2023	
23	Оформление и печать документации	3 дня	29.05.2024	31.06.2024	22
24	Завершение оформления документации	0 дней	01.06.2024	01.06.2024	21
25	Конец проекта	0 дней	01.06.2024	01.06.2024	24

Также были определены необходимые ресурсы. Список ресурсов представлен в таблице 2.

Таблица 2 – Ресурсы и затраты проекта

№ п/п	Название ресурса	Тип ресурса	Стандартная ставка	Начисление
1	Аналитик	Трудовой	200,00 ₽/ч	Пропорциональное
2	Проектировщик	Трудовой	200,00 ₽/ч	Пропорциональное
3	Программист	Трудовой	250,00 ₽/ч	Пропорциональное
4	Тестировщик	Трудовой	200,00 ₽/ч	Пропорциональное
5	Компьютер	Затраты		В начале
6	Принтер	Затраты		В начале
7	Бумага	Материальный	300 ₽	В начале
8	Интернет	Затраты		В начале
9	Электроэнергия	Затраты		По окончании

Выделенные задачи и ресурсы были занесены в проект в GanttPro. Результаты представлены на рисунке 13.

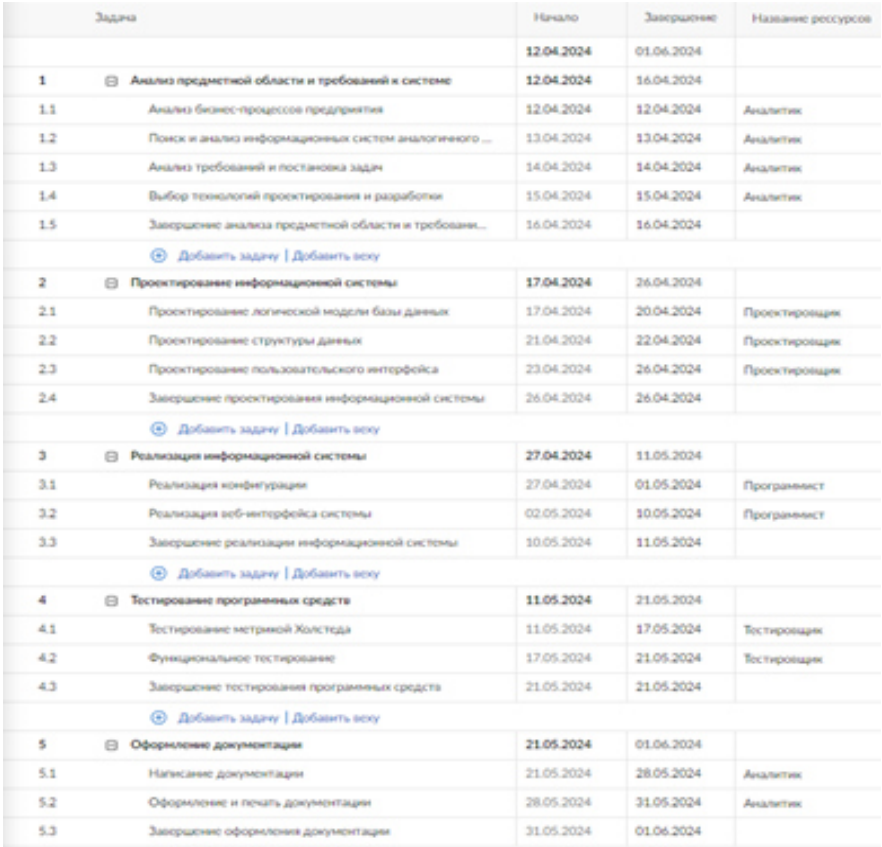


Рисунок 13 – Задачи проекта с распределенными по ним ресурсами в GanttPro

В соответствии с занесенными задачами и ресурсами была построена диаграмма Ганта, представленная на рисунке 14.

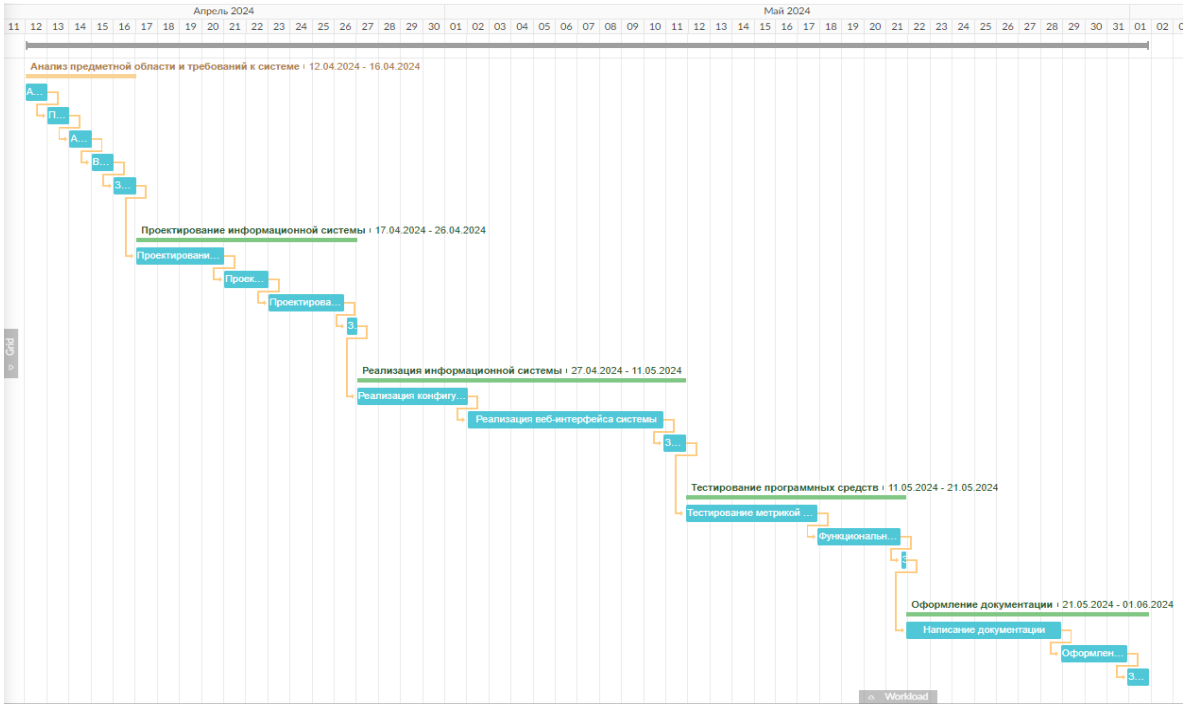
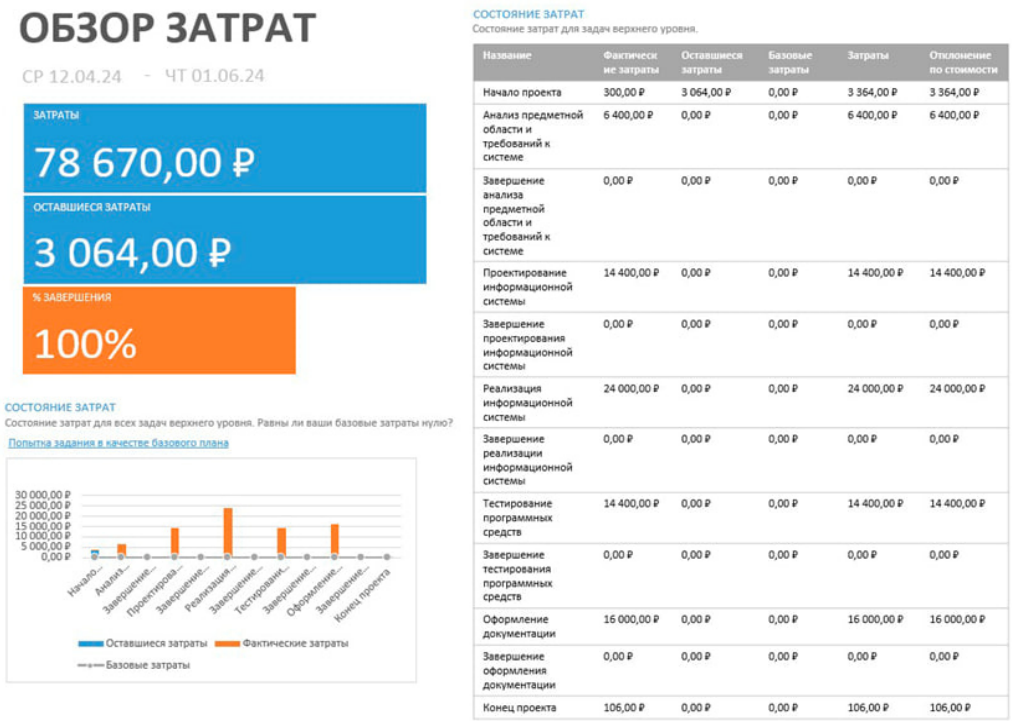


Рисунок 14 – Диаграмма Ганта

В соответствии с построенной диаграммой был сформирован отчет по затратам на разработку, который представлен на рисунке 15.



ОБЗОР ЗАТРАТ

СР 12.04.24 - ЧТ 01.06.24

ЗАТРАТЫ
78 670,00 ₽

ОСТАВШИЕСЯ ЗАТРАТЫ
3 064,00 ₽

% ЗАВЕРШЕНИЯ
100%

СОСТОЯНИЕ ЗАТРАТ
Состояние затрат для всех задач верхнего уровня. Равны ли ваши базовые затраты нулю?
Попытка задания в качестве базового плана

СОСТОЯНИЕ ЗАТРАТ
Состояние затрат для задач верхнего уровня.

Название	Фактические затраты	Оставшиеся затраты	Базовые затраты	Затраты	Отклонение по стоимости
Начало проекта	300,00 ₽	3 064,00 ₽	0,00 ₽	3 364,00 ₽	3 364,00 ₽
Анализ предметной области и требований к системе	6 400,00 ₽	0,00 ₽	0,00 ₽	6 400,00 ₽	6 400,00 ₽
Завершение анализа предметной области и требований к системе	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Проектирование информационной системы	14 400,00 ₽	0,00 ₽	0,00 ₽	14 400,00 ₽	14 400,00 ₽
Завершение проектирования информационной системы	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Реализация информационной системы	24 000,00 ₽	0,00 ₽	0,00 ₽	24 000,00 ₽	24 000,00 ₽
Завершение реализации информационной системы	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Тестирование программных средств	14 400,00 ₽	0,00 ₽	0,00 ₽	14 400,00 ₽	14 400,00 ₽
Завершение тестирования программных средств	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Оформление документации	16 000,00 ₽	0,00 ₽	0,00 ₽	16 000,00 ₽	16 000,00 ₽
Завершение оформления документации	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽	0,00 ₽
Конец проекта	106,00 ₽	0,00 ₽	0,00 ₽	106,00 ₽	106,00 ₽

Рисунок 15 – Отчет по затратам проекта

Таким образом, в результате функционально-стоимостного анализа, временные затраты для выполнения проекта составят 58 дней, включая выходные и праздничные дни, а стоимость его выполнения, с учетом прочих затрат на ресурсы, составит 78670 рублей. Оценка стоимости является неотъемлемой частью планирования любого проекта. Она позволяет установить возможный бюджет на разработку, определить оптимальный способ расходования средств и контролировать затраты на всех этапах работы над проектом. Это дает возможность руководству и исполнителям проекта понимать, какие ресурсы нужны, чтобы выполнить задачи в соответствии с ожиданиями, анализировать стоимостную эффективность различных решений и планировать нужные корректировки. Более того, правильная оценка стоимости создает базу для принятия качественных решений и снижения рисков в процессе выполнения проекта.

3 Проектирование программного обеспечения

3.1 Выбор технологий и инструментальных средств разработки

Для создания web-приложения для системы контроля успеваимости подобраны следующие инструменты для разработки, которые используются в современной web-разарботке: СУБД PostgreSQL, языки программирования JavaScript, TypeScript, HTML, CSS, React, Node.js.

PostgreSQL – это открытая СУБД с высокой производительностью. Для разработки web-могут использоваться различные СУБД, однако PostgreSQL лучше остальных систем по следующим критериям:

– мощность и надежность;

– масштабируемость.

PostgreSQL обеспечивает более эффективную масштабируемость, особенно в случае больших и сложных приложений. Он может обрабатывать более высокие нагрузки и лучше масштабироваться при увеличении количества пользователей и объема данных.

React – это фреймворк для языка программирования JavaScript, который упращает разработку сложных структур при разработке пользовательского инфтерфейса. React использует прицип одностраничных приложений, при котором используется один HTML-документ в который динамично подгружаются HTML, CSS, JavaScript файлы посредством AJAX. Прямым аналогом React является фреймворк Angular. Ниже перечислены некоторые преимущества использования React над Angular при web-разработке:

– оптимизация из-за наличия Virtual DOM в React позволяет производить на странце ререндерин только тех элементов, которые нуждаются в обновлении;

– широкий выбор различных подключаемых библиотек, что делает разработку более простой;

– производительность за счёт возможности рендеринига HTML-страницы на сервере.

Node.js – это среда выполнения JavaScript, которая позволяет выполнять скрипты языка программирования JavaScrip на серверной. Сейчас имеется много различных аналогов данной технологии (ASP.NET, phpMyAdmin и т.д.), однако у Node.js есть следующие преимущества:

– высокая производительность: Node.js работает на основе событийного цикла, что позволяет обрабатывать большое количество одновременных запросов без блокировки потока исполнения;

– быстрое разработка: Node.js имеет огромную библиотеку модулей, что упрощает процесс разработки и ускоряет создание приложений;

– одноязычность: Node.js использует только JavaScript, что упрощает процесс разработки;

– легкость масштабирования: Node.js поддерживает горизонтальное масштабирование, что позволяет увеличивать количество серверов при необходимости;

– асинхронность: Node.js позволяет выполнять операции асинхронно, что повышает производительность приложения;

– кроссплатформенность: Node.js может работать на разных операционных системах, таких как Windows, Mac и Linux.

JavaScript – язык программирования, который используется в основном для создания веб-приложений. Причной, по которой для разработки был выбран данный язык, являтся то что большенство web-приложений пишут именно не нем и оналогов этому языку программирования почти нет

TypeScript – надстройка над языком программирования JavaScript, которая позваляет использовать строгую типизацию при разрабоки web-приложения

CSS и HTML – это два языка, которые используются для создания каркаса страницы и ее стилизации. Данные языки используются в вместе с JavaScript. Используются из-за отсутствия аналогов.

3.2 Проектирование архитектуры программы

В данной работе было принято решение использовать трёхуровневую систему «клиент-сервер». Ее схема представлена на рисунке 16.

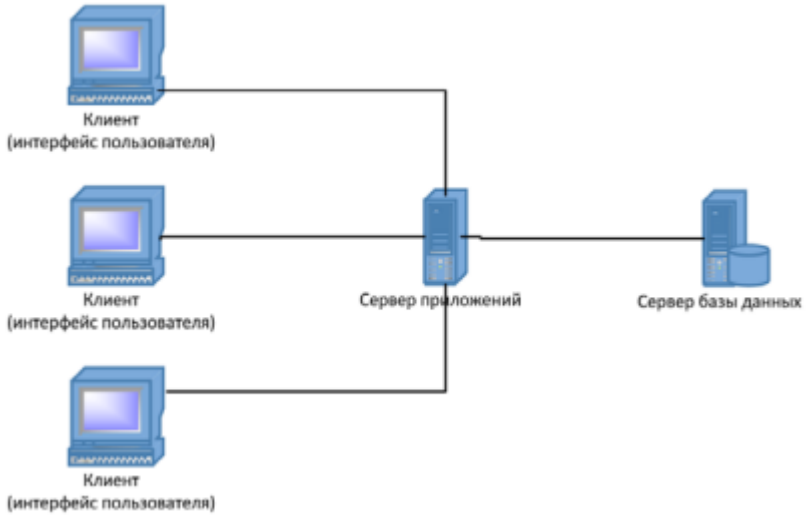


Рисунок 16 – Трехуровневая архитектура «клиент-сервер»

В данной архитектуре на сервере базы данных осуществляется управление данными. Сервер приложений реализует функционал всего проекта и отвечает за взаимодействие с базой данных. Клиентская же часть, в первую очередь, обеспечивает интерфейс, позволяющий просматривать информацию, полученную с сервера приложений, и взаимодействовать с данными путем удаленного обращения к серверу. В данной работе в качестве клиентов будут выступать мобильные устройства. Основным преимуществом трехуровневой архитектуры «клиент-сервер» является то, что каждый уровень в ней независим и может разрабатываться параллельно с разработкой других уровней. Также клиент в данной архитектуре не работает напрямую с базой данных, в связи с чем повышается безопасность данных.

3.3 Диаграмма деятельности

Диаграмма деятельности – это графическое представление последовательности действий или процесса в системе. Она состоит из узлов и

связей, которые показывают порядок выполнения задач и переходы между ними [13].

Для разработки диаграммы деятельности был выбран вариант использования «Выставить оценки». Диаграмма деятельности в нотации UML приведена на рисунке 17.

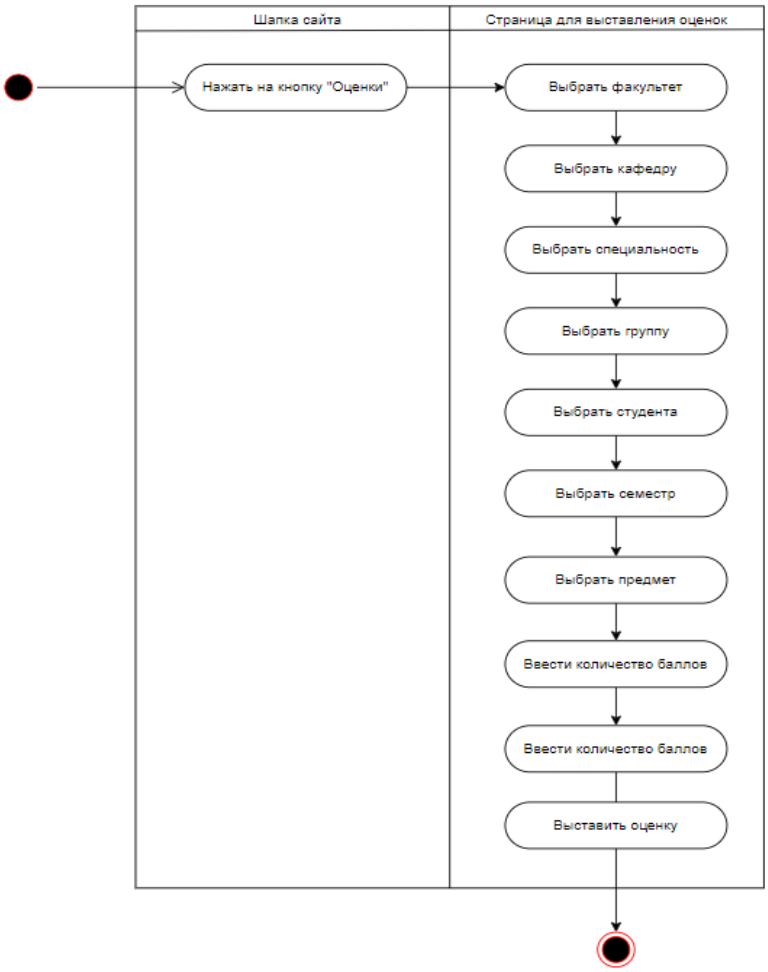


Рисунок 17 – Диаграмма деятельности

Рассмотрим приведенную диаграмму деятельности. Для выставления оценок необходимо нажать на кнопку «Оценки» в шапке сайта. После открытия страницы для выставления оценки необходимо выбрать следующие параметры: «Факультет», «Кафедра», «Специальность», «Группа», «Студент», «Семестр», «Предмет», после чего ввести количество выставляемых данных. После проведенных действий, необходимо выставить оценку. Таким образом в системе появится новая оценка.

3.4 Проектирование базы данных

3.4.1 Концептуальное проектирование базы данных

На основе анализа предметной области была построена концептуальная модель базы данных. Графическое представление концептуальной модели БД в виде диаграммы «сущность-связь» представлено на рисунке 18.

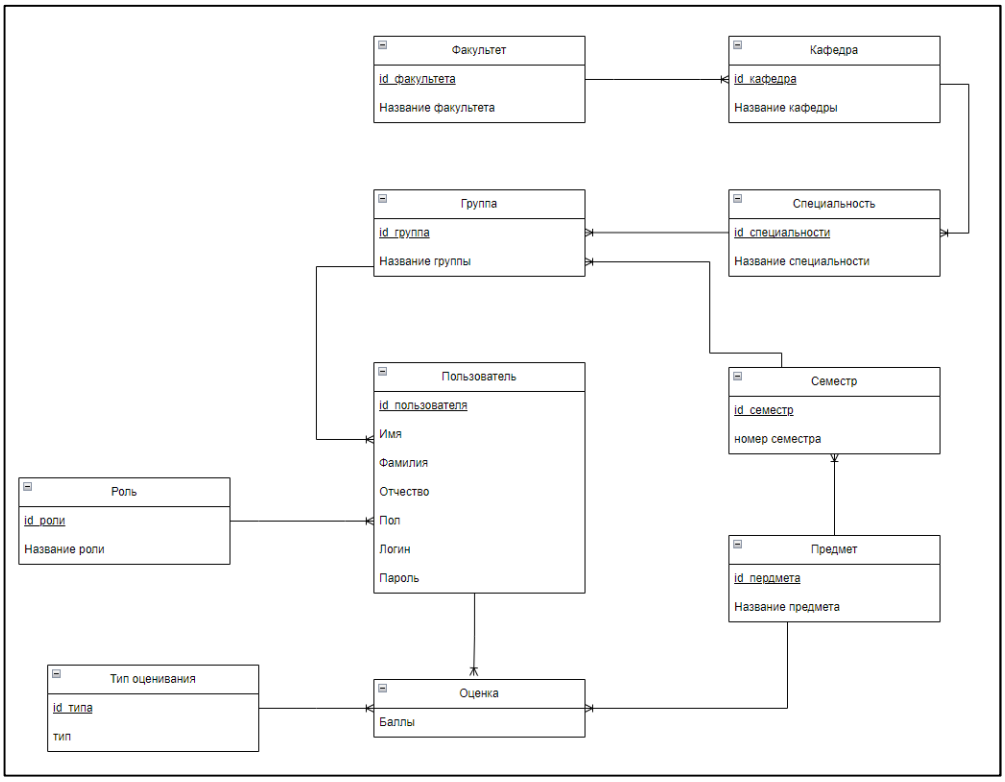


Рисунок 18 – Графическое представление концептуальной модели базы данных

Сущности концептуальной модели данных представлены в таблице 3.

Таблица 3 – Описание концептуальной модели

№	Название	Описание
1	Факультет	Список факультетов в образовательном учреждении
2	Кафедра	Список кафедр в факультете
3	Специальность	Список специальностей в кафедре
4	Группа	Список групп у специальности
5	Пользователь	Пользователь системы
6	Роль	Роль пользователя
7	Семестр	Семестр группы
8	Предмет	Учебная дисциплина
9	Оценка	Баллы, получаемые студент
10	Тип оценки	Тип оценивания, за которые студент получает баллы (лабораторная работа, зачет, экзамен)

В таблице 4 представлено описание связей между сущностями.

Таблица 4 – Описание связей между сущностями концептуальной модели

№	Связь	Мощность
1	Список кафедр на факультете – Факультет	1-М
2	Список кафедр на факультете – Кафедра	1-М
3	Список специальностей на кафедре – Кафедра	1-М
4	Список специальностей на кафедре – Специальность	1-М
5	Группы специальности – Специальность	1-М
6	Группы специальности – Группа	1-М
7	Список студентов в группе – Группа	1-М
8	Список студентов в группе – Пользователь	1-М
9	Список предметов в семестре – Семестр	1-М
10	Список предметов в семестре – Предмет	1-М

Таким образом, были спроектированы основные сущности с атрибутами и связями между ними, без включения атрибутов первичных и внешних ключей.

3.4.2 Логическое проектирование базы данных

На основании концептуальной модели была спроектирована логическая модель базы данных, представленная на рисунке 19.

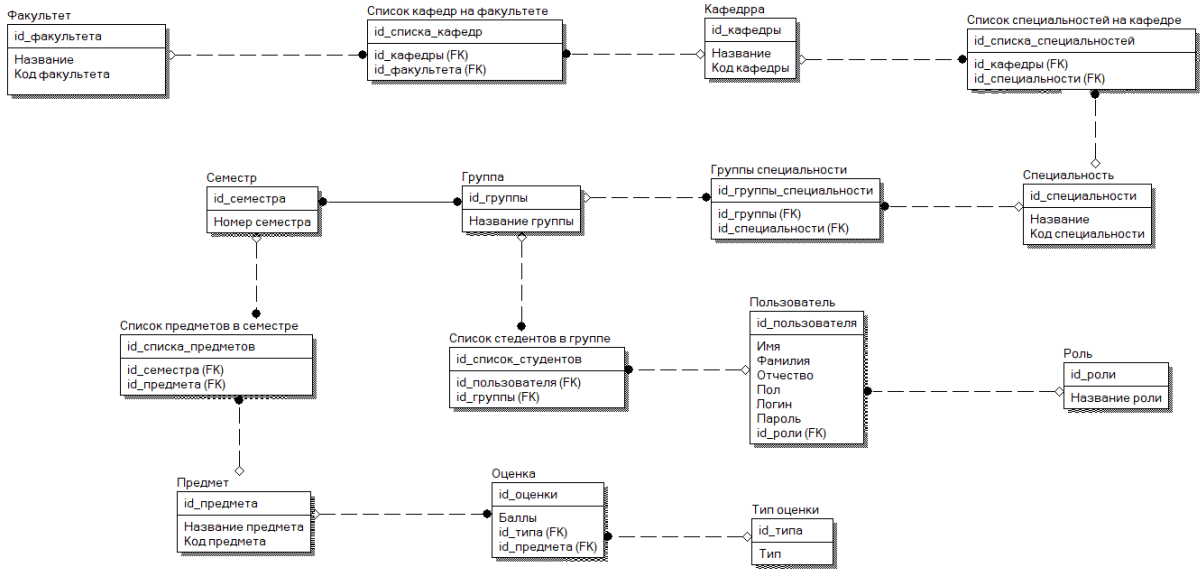


Рисунок 19 – Логическая модель

Сущности логической модели данных, не определенные в концептуальной модели, представлены в таблице 5.

Таблица 5 – Сущности логической модели

№	Название	Описание
1	Список кафедр на факультете	Список факультетов в образовательном учреждении
2	Список специальностей на кафедре	Список кафедр в факультете
3	Группы специальности	Список специальностей в кафедре
4	Список студентов в группе	Список групп у специальности
5	Список предметов в семестре	Пользователь системы

Также в таблице 6 представлено описание связей между сущностями логической модели, не определенных в результате концептуального моделирования.

Таблица 6 – Описание связей между сущностями логическими моделями

№	Связь	Мощность
1	Список кафедр на факультете – Факультет	1-М
2	Список кафедр на факультете – Кафедра	1-М
3	Список специальностей на кафедре – Кафедра	1-М
4	Список специальностей на кафедре – Специальность	1-М
5	Группы специальности – Специальность	1-М
6	Группы специальности – Группа	1-М
7	Список студентов в группе –Группа	1-М
8	Список студентов в группе – Пользователь	1-М
9	Список предметов в семестре –Семестр	1-М
10	Список предметов в семестре – Предмет	1-М
11	Роль – Пользователь	1-М
12	Группа – Семестр	1-М
13	Оценка – Предмет	1-М
14	Оценка – Тип оценки	1-М

Несмотря на достаточную содержательность данной модели, сущности все еще представлены в абстрактной форме, без привязки к конкретной реализации базы данных.

3.5 Проектирование серверной части приложения

Серверная часть приложения выполнена на основе архитектуры REST API. В данном случае взаимодействие с сервером осуществляется по протоколу HTTP при помощи стандартных методов, таких как GET (получение ресурса), PUT (обновление существующего ресурса), POST (создание нового ресурса) и DELETE (удаление ресурса). При этом обращение к ресурсам сервера происходит по их уникальным идентификаторам, называемым URI.

Исходя из выбранных технологий разработки серверной части приложения, для ее реализации было решено использовать шаблон MVC (ModelView-Controller). Схема данного шаблона приведена на рисунке 20.

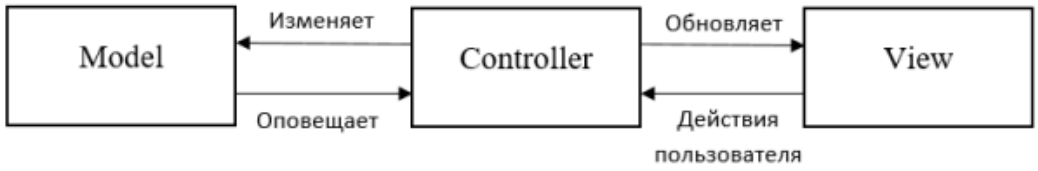


Рисунок 20 – Схема шаблона MVC

Как видно из данной схемы, в системе имеется 3 компонента: модель, представление и контроллер. Классы уровня модели отвечают за организацию доступа к данным (за непосредственное взаимодействие с базой данных). Классы контроллера отвечают за организацию взаимодействия пользователя с клиентом. 39 Они обрабатывают HTTP-запросы клиента и формируют ответы на них. Уровень представления отвечает за взаимодействие с пользователем. Выбор данного шаблона обусловлен использованием в работе технологии NodeJS, которая нацелена на применение данного шаблона.

3.6 Проектирование интерфейса пользователя

Процесс разработки дизайна пользовательского интерфейса включает в себя этап создания вайрфрейов, которые представляют собой набросок или скелет будущего сайта, который имеет минимальную детализацию.

Вайрфрейм должен четко показывать:

– основную группу контента;

– структуру информации;

– описание и базовую визуализацию взаимодействия между интерфейсом и пользователем.

Подобная схема должна быть построена по правилам эстетики, но сильно упрощена. Черный и белый – это стандартные цвета для построения вайрфреймов.

Вайрфреймы обычно используются как документация к проекту, в виду своей статичности и фиксированности в качестве способа отражения интерфейса.

Хотя подобное графическое представление и редко используется как материал для тестирования, в некоторых случаях, оно помогает получить обратную связь на начальном этапе исследования. Для построения вайрфреймов и работы с изображениями на сайте была использована программа Adobe Photoshop.

Вайрфрейм главной страницы сайта, на котором представлено обозначение основных элементов дизайна представлен на рисунке 21.

Вайрфрейм интерфейса страницы авторизации представлен на рисунке 21.

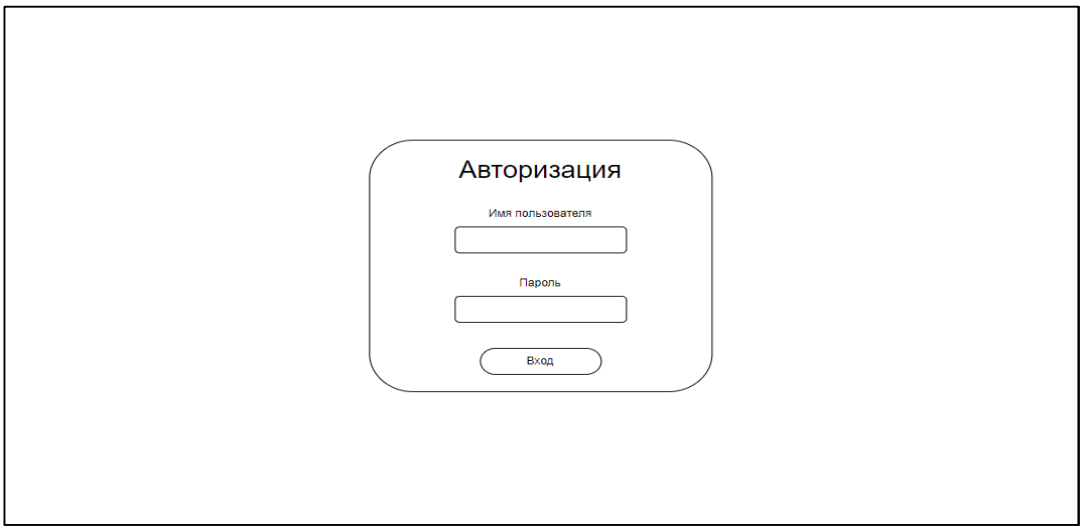


Рисунок 21 – Вайрфрейм окна авторизации

На вайрфрейме страницы авторизации имеются следующие элементы:

– поле ввода имени пользователя;

– поле ввода пароля;

– кнопка входа, при нажатии на которую осуществляется попытка авторизации.

Вайрфрейм интерфейса страницы оценивания представлен на рисунке 22.

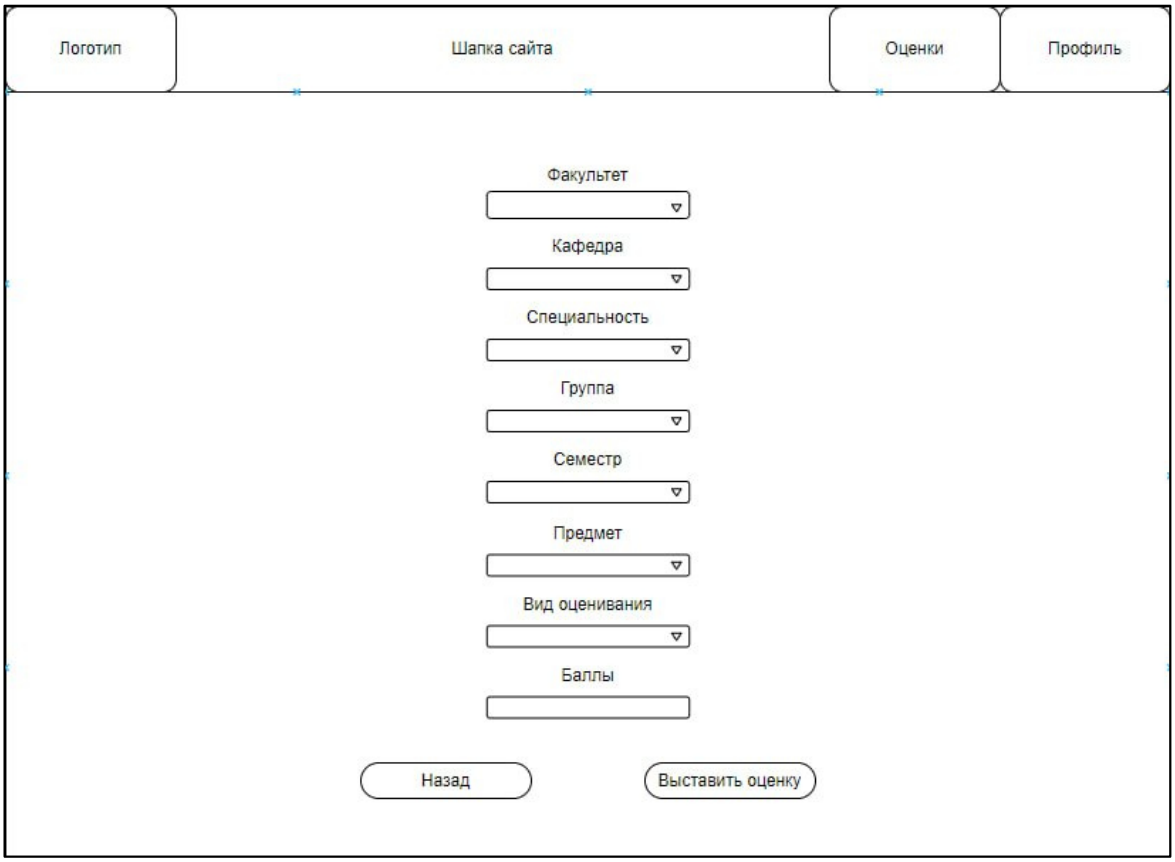


Рисунок 22 – Вайрфрейм страницы оценивания

На макете страницы оценивания имеются следующие элементы:

– выпадающий список с группами;

– выпадающий список с семестрами;

– выпадающий список с предметами;

– выпадающий список с баллами;

– поле ввода баллов;

– кнопка «Выставить оценку», для выполнения выставления оценки;

– кнопке «Назад», чтобы вернуться на предыдущую страницу.

Вайрфрейм интерфейса страницы просмотра оценки представлен на рисунке 23.

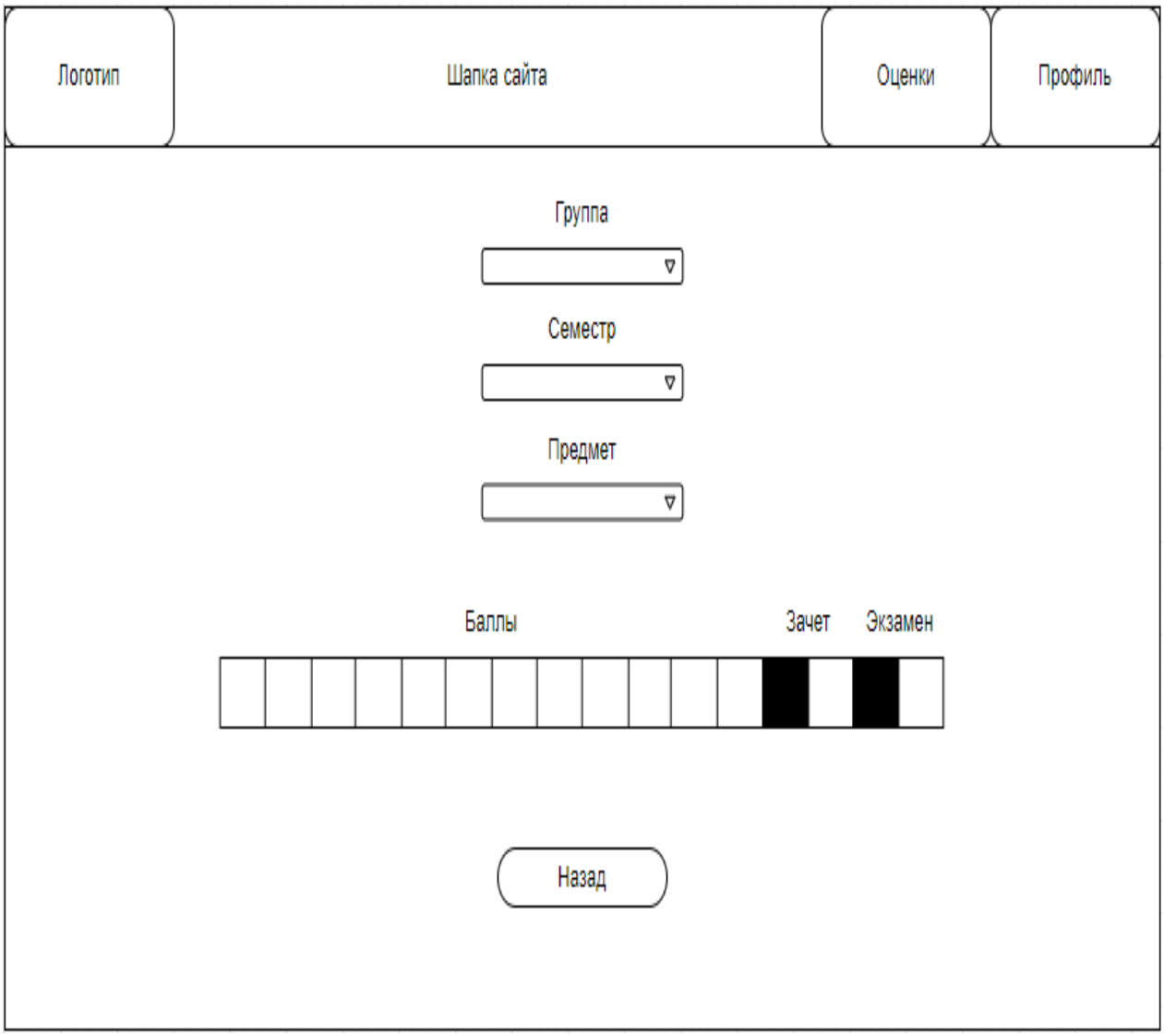


Рисунок 23 – Вайрфрейм страницы просмотра оценки

На макете страницы просмотра оценки имеются следующие элементы:

– выпадающий список с группами;
– выпадающий список с семестрами;
– выпадающий список с предметами;
– кнопке «Назад», чтобы вернуться на предыдущую страницу.

Вайрфрейм интерфейса страницы профиля представлен на рисунке 24.

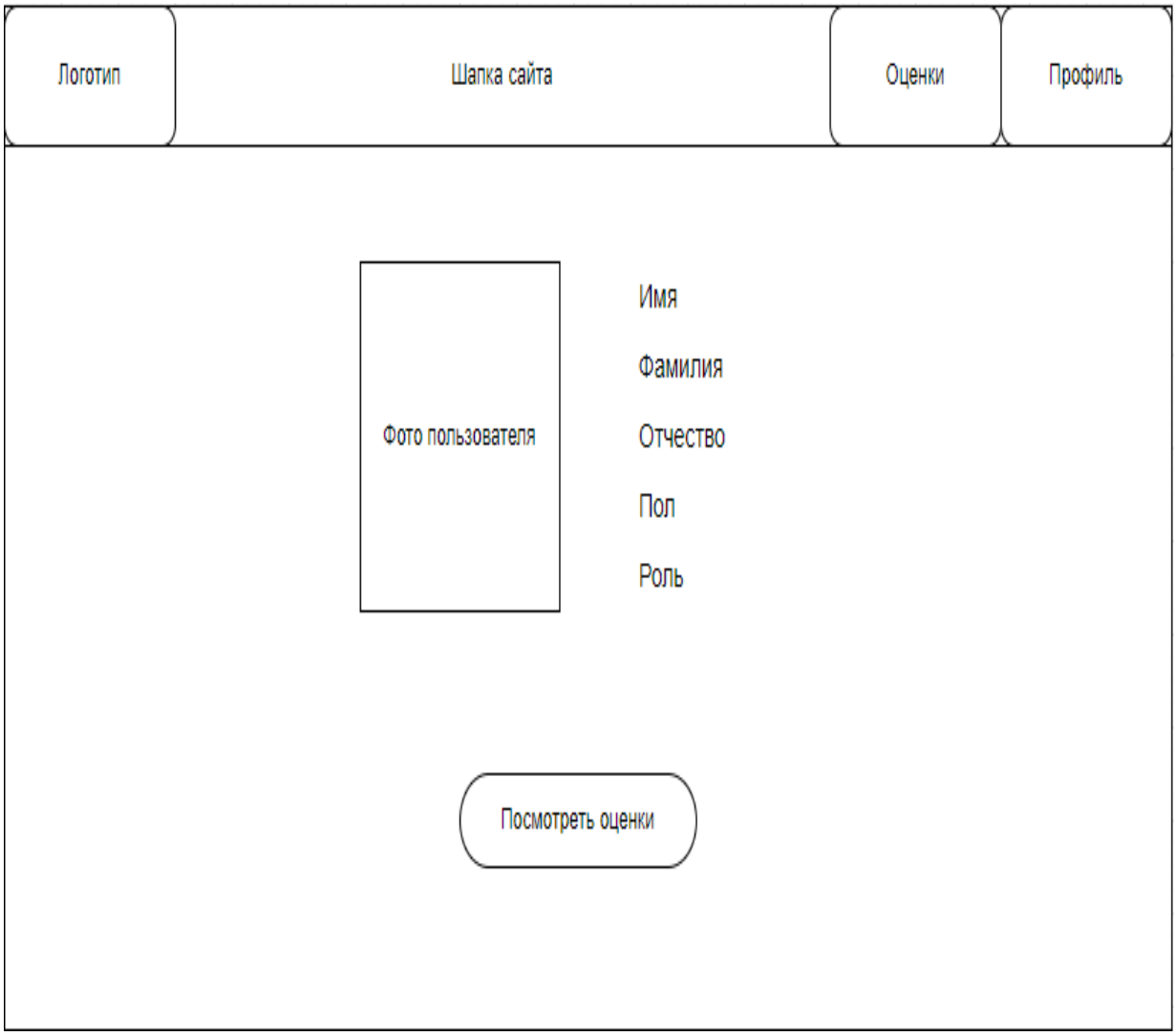


Рисунок 24 – Вайрфрейм страницы профиля

На макете страницы профиля имеются следующие элементы:

– фото пользователя;

– данные пользователя (имя, фамилия, отчество, пол, роль);

– кнопке «Посмотреть оценки», чтобы перейти на страницу.

На этом проектирование можно считать законченным и перейти к реализации.

4 Реализация программного обеспечения

4.1 Реализация базы данных

На основании концептуальной и логической модели для СУБД PostgreSQL 13 была разработана физическая модель базы данных (рисунок 25).

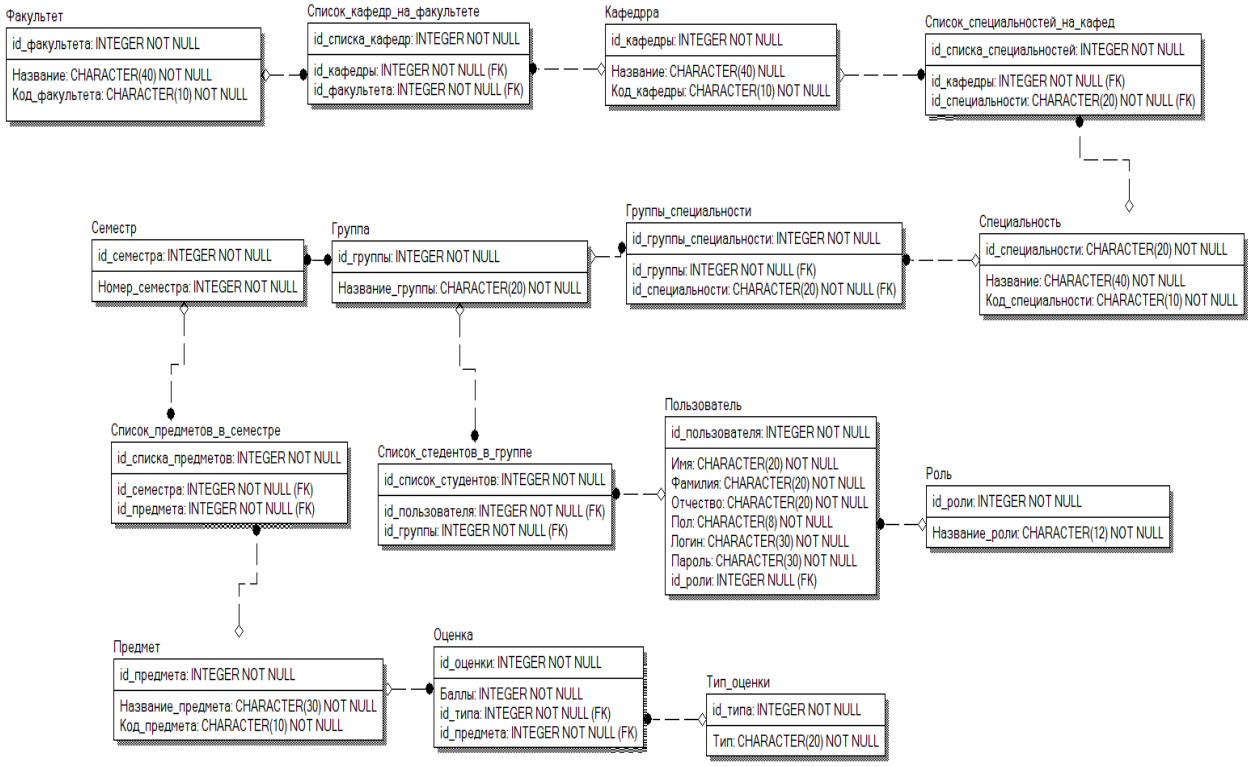


Рисунок 25 – Графическое представление физической модели данных

Описание атрибутов сущности «Факультет» представлено в таблице 7.

Таблица 7 – Описание атрибутов сущности «Факультет»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_факультета	Идентификатор	integer	нет	да	PK
Название	Название факультета	character(40)	нет	да	-
Код факультета	Код факультета	character(10)	нет	да	-

.

Описание атрибутов сущности «Кафедра» представлено в таблице 8

Таблица 8 – Описание атрибутов сущности «Кафедра»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_кафедра	Идентификатор	integer	нет	да	PK
Название	Название кафедры	character (40)	нет	да	-
Код кафедры	Код кафедры	character (10)	нет	да	-

Описание атрибутов сущности «Список кафедр на факультете» представлено в таблице 9.

Таблица 9 – Описание атрибутов сущности «Список кафедр на факультете»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id_списка_кафедр	Идентификатор	integer	нет	да	PK
id_кафедры	Идентификатор кафедры	integer	нет	да	FK
id_факультета	Идентификатор факультета	integer	нет	да	FK

Описание атрибутов сущности «Специальность» представлено в таблице 10.

Таблица 10 – Описание атрибутов сущности «Специальность»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_специальности	Идентификатор	integer	нет	да	PK
Название	Название кафедры	character (40)	нет	да	-
Код специальности	Код специальности	character (10)	нет	да	-

Описание атрибутов сущности «Список специальностей на кафедре» представлено в таблице 11.

Таблица 11 – Описание атрибутов «Список специальностей на кафедре»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id_списка_специальностей	Идентификатор	integer	нет	да	PK
id_специальности	Идентификатор специальности	integer	нет	да	FK
id_кафедры	Идентификатор кафедры	integer	нет	да	FK

Описание атрибутов сущности «Список групп специальности» представлено в таблице 12.

Таблица 12 – Описание атрибутов «Список специальностей на кафедре»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id_группы_специальности	Идентификатор	integer	нет	да	PK
id_группы	Идентификатор группы	integer	нет	да	FK
id_специальности	Идентификатор специальности	integer	нет	да	FK

Описание атрибутов сущности «Группа» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Группа»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_группы	Идентификатор	integer	нет	да	PK
Название_группы	Название группы	character (20)	нет	да	-

Описание атрибутов сущности «Роль» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «Роль»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_роли	Идентификатор	integer	нет	да	PK
Название_роли	Название роли	character (10)	нет	нет	-

Описание атрибутов сущности «Пользователь» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Пользователь»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_пользователя	Идентификатор	integer	нет	да	PK
Имя	Название роли	character (20)	нет	да	-
Фамилия	Фамилия	character (20)	нет	нет	-
Отчество	Отчество	character (20)	да	нет	-
Пол	Пол	character (8)	нет	нет	-
Логин	Логин пользователя	character (20)	нет	да	-
Пароль	Пароль пользователя	character (20)	нет	нет	-
id_роли	Идентификатор роли пользователя	integer	нет	да	FK

Описание атрибутов сущности «Список студентов в группе» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Список студентов в группе»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id_список_студентов	Идентификатор	integer	нет	да	FK

Продолжение таблицы 16

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_группы	Идентификатор группы	integer	нет	да	FK
id_пользователя	Идентификатор пользователя	integer	нет	да	FK

Описание атрибутов сущности «Семестр» представлено в таблице 17.

Таблица 17 – Описание атрибутов сущности «Семестр»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_семестра	Идентификатор	integer	нет	да	PK
Номер семестра	Номер семестра	integer	нет	нет	-

Описание атрибутов сущности «Тип оценивания» представлено в таблице 18.

Таблица 18 – Описание атрибутов сущности «Тип оценивания»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_типа	Идентификатор	integer	нет	да	PK
Тип	Тип оценивания	integer	нет	нет	-

Описание атрибутов сущности «Предмет» представлено в таблице 19.

Таблица 19 – Описание атрибутов сущности «Предмет»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_предмета	Идентификатор	integer	нет	да	PK

Продолжение таблицы 19

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Название_предмета	Название предмета	character (30)	нет	нет	-
Код_предмета	Код предмета	character (10)	нет	да	-

Описание атрибутов сущности «Список предметов в семестре» представлено в таблице 20.

Таблица 20 – Описание атрибутов сущности «Список предметов в семестре»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
id_списка_предметов	Идентификатор	integer	нет	да	PK
id_предмета	Идентификатор предмета	integer	нет	да	FK
id_семестра	Идентификатор семестра	integer	нет	да	FK

Описание атрибутов сущности «Список предметов в семестре» представлено в таблице 21.

Таблица 21 – Описание атрибутов сущности «Оценка»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id_оценки	Идентификатор	integer	нет	да	PK
id_типа_	Идентификатор типа оценивания	integer	нет	да	FK
id_предмета	Идентификатор предммета	integer	нет	да	FK
Points	Баллы	integer	нет	нет	-

На данной физической модели структура базы данных отображена с учетом спецификации типов СУБД PostgreSQL 13.

На основе физической модели БД был сгенерирован скрипт на языке SQL для создания базы данных. Его код представлен в приложении Б. Развертывание спроектированной базы данных было выполнено путем запуска данного скрипта в СУБД PostgreSQL 13.

4.2 Реализация серверной части приложения

Реализация серверной части приложения была произведена в среде программирования Visual Studio Code на языке JavaScript с применением технологии Web API на платформе NodeJS. Для взаимодействия с созданной базой данных была использована технология Express.

На основании используемой технологии вся серверная часть делится на дав слоя: слой контроллеров и роутеров. Структура серверной части приложения представлена на рисунке 26.

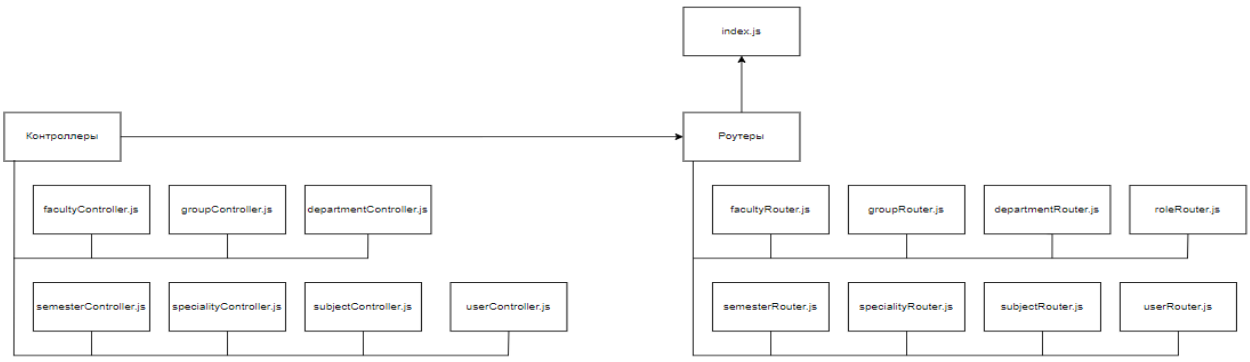


Рисунок 26 – Структура серверной части приложения

Контроллеры обращаются к базе данных с помощью SQL-запросов и получают данные из нее, после чего обрабатывают ее. Функции, которые работают с базой данных создаются в классах, после чего экспортируются в роутеры. В роутерах определяются HTTP-запросы, с помощью которых можно обращаться к базе данных. Далее все роутеры экспортируются в index.js файл, откуда и запускается сам сервер. Описание контролеров приведено в таблице 22.

Таблица 22 – Описание контроллеров

Контроллер	Назначение
facultyController.js	Обрабатывает запросы добавления, удаления, изменения и получения факультетов
departmentController.js	Обрабатывает запросы на добавления новых кафедр и приписывание его к определенному факультету, изменение, удаление, получение кафедр
specialityController.js	Обрабатывает запросы на добавления новых специальностей и приписывание ее к определенной кафедре, изменение, удаление, получение специальностей
groupController.js	Обрабатывает запросы на создание группы, формирование участников группы
userController.js	Обрабатывает запросы на регистрацию и авторизацию пользователя, изменение, получение и удаление данных о пользователе
subjectController.js	Обрабатывает запросы на добавление, удаление, изменение учебных дисциплин
semesterController.js	Обрабатывает запросы на формирование учебный программы на семестр определенной группе

Завершив реализацию серверной части, остается реализовать клиентскую.

4.3 Реализация клиентской части

На данном этапе было реализовано веб-приложение на языке JavaScript в среде разработки Visual Studio Code. Исходный код представлен в приложении Г. Диаграмма компонентов, отражающая структуру клиентской части, представлена на рисунке 27.

Рисунок 27 – Структура клиентской части

Описание компонентов клиентской части представлено в таблице 23.

Таблица 23 – Описание компонентов

Компонент	Назначение
userSlice.ts	Создает объект состояния с информацией об авторизированном пользователе, добавляет функции изменения состояния
loginSlice.ts	Создает объект состояния с информацией о значении полей ввода в окне авторизации, добавляет функции изменения данных значений
marksSlice.ts	Создает объект состояния со значениями, которые нужны для выставления оценки студенту (факультет, кафедра, специальность, группа, семестр, предмет, баллы), данные для просмотра оценое по предмету (группа, семестр, предмет), а также функции для изменения состояния
createUserSlice.ts	Создает объект состояния с информацией о значении полей ввода на странице создания пользователя
store.ts	Создает объект, который объединяет все объекты, создающее состояния и функции, которые изменяет состояние
loginModalWindow.tsx	Компонент модального окна, на котором пользователь может произвести авторизацию в системе
loginModalWindow.module.scss	Файл со стилями для компонента loginModalWindow.tsx
Profile.tsx	Компонент страницы с профилем пользователя
Profile.module.scss	Файл со стилями для компонента Profile.tsx
Marks.tsx	Компонент страницы с выставлением оценок студенту, либо просмотра оценок студентом
Marks.module.scss	Файл со стилями для компонента Marks.tsx
CreateUser.tsx	Компонент для создания пользователя администратором
CreateUser.module.scss	Файл со стилями для компонента CreateUser.tsx
App.tsx	Файл, обеспечивающий роутинг компонентов
index.html	Файл, отображающий компоненты

На этом реализацию проекта можно считать завершенной, но необходимо сделать тесты.

5 Тестирование программного обеспечения

5.1 Тестирование серверной части приложения

На данном этапе было выполнено функциональное тестирование в целях проверки правильности работы серверной части приложения. Результаты функционального тестирования серверной части представлены в таблице 24.

Таблица 24 – Результаты функционального тестирования серверной части

№	Описание теста	Структура запроса	Результат

1	Авторизация пользователя	Метод: POST
		URL: user/login
		Body: {
		“username: wowspecial3”,
		“password: papa8962”
		}
			Тест пройден. Пользователь авторизирован (рисунок Г..1)

2	Регистрация пользователя	Метод: POST
		URL: user/registration
		Body: {
		"name": "Григорий",
		"surname": "Енини",
		"secondName": "Денисович",
		"username": "SouSeaker",
		"password": "p89273698970"
		}
			Тест пройден. Пользователь зарегистрирован (рисунок Г.2)

3	Создание факультета	Метод: POST
		URL: faculty/create
		Body: {
		“name”: “ФИТЭ”
		}
			Тест пройден. Факультет создан (рисунок Г.3)

4	Удаление факультета	Метод: DELETE URL: faculty/delete/3	Тест пройден. Факультет удален (рисунок Г.4)

5	Создание кафедры	Метод: POST
		URL: department/create
		Body: {
		"name": "Прикладная математика"
		}
			Тест пройден. Кафедра создана (рисунок Г.5)

6	Удаление кафедры	Метод: DELETE URL: institute/delete/2	Тест пройден. Кафедра удалена (рисунок Г.6)

7	Создание специальности	Метод: POST
		URL: speciality/create
		Body: {
		"name": "Прикладная математика"
		}
			Тест пройден. Специальность создана (рисунок Г.7)

8	Удаление специальности	Метод: DELETE URL: speciality/delete/2	Тест пройден. Специальность удалена (рисунок Г.8)

Продолжение таблицы 24

№	Описание теста	Структура запроса	Результат

9	Создание группы	Метод: POST
		URL: group/create
		Body: {
		"name": "20ВИ1"
		}
			Тест пройден. Группа пройдена (рисунок Г.9)

10	Удаление группы	Метод: DELETE URL: group/delete/2	Тест пройден. Группа удалена (рисунок Г.10)

11	Создание предмета	Метод: POST
		URL: subject/create
		Body: {
		“name”: “Математика”
		}
			Тест пройден. Предмет создан (рисунок Г.11)

12	Удаление предмета	Метод: DELETE URL: subject/delete/4	Тест пройден. Предмет создан (рисунок Г.12)

13	Добавление кафедры в факультет	Метод: POST
		URL: faculty/add_institute
		Body: {
		“facultyid”: 1,
		“departmentid”: 1
		}
			Тест пройден. Кафедра добавлена в факультет (рисунок Г.13)

14	Добавление специальности в кафедру	Метод: POST
		URL: institute/add_speciality
		Body: {
		"instituteID": 1,
		"specialityID": 1
		}
			Тест пройден. Специальность добавлена в кафедру (рисунок Г.14)

15	Добавление группы в специальность	Метод: POST
		URL: institute/add_speciality
		Body: {
		"specialityID": 1,
		"groupID": 1
		}
			Тест пройден. Группа добавлена в специальность (рисунок Г.15)

16	Создание семестра	Метод: POST
		URL: semester/create
		Body: {
		"number": 1,
		"groupid": 1
		}
			Тест пройден. Семестр создан (рисунок Г.16)

17	Получение информации о пользователе	Метод: GET URL: user/get_one/2	Тест пройден. Информация о пользователе получена (рисунок Г.17)
18	Получение информации о факультете	Метод: GET URL: faculty/get_one/1	Тест пройден. Информация о факультете получена (рисунок Г.18)
19	Получение списка кафедр на факультете	Метод: GET URL: institute/getByFacultyName/ФВТ	Тест пройден. Список кафедр на факультете получен (рисунок Г.19)

Окончание таблицы 24

№	Описание теста	Структура запроса	Результат
20	Получение списка специальностей на кафедре	Метод: GET URL:speciality/getByDepartmentName/ИВС	Тест пройден. Список специальностей на факультете получен (рисунок Г.20)
21	Получение списка групп у специальности	Метод: GET URL: group/getGroupsBySpecialtyName/ Прикладная информатика	Тест пройден. Список групп у специальности получен (рисунок Г.21).
22	Получение списка семестров группы	Метод: GET URL: semester/getByGroupId/1	Тест пройден. Список семестров группы получен (рисунок Г.22)

Все тесты были пройдены. Значит можно говорить о том, что серверная часть приложения работает корректно.

5.2 Тестирование клиентской части приложения

Также было произведено функциональное тестирование клиентской части web-приложения, результаты которого приведены в таблице 25.

Таблица 25 – Результаты функционального тестирования web-приложения

Вариант использования	Параметры теста	Результат
Авторизация	Авторизация пользователя с корректными данными. Логин: wowspecial3, Пароль: papa8962	Тест пройден. Авторизация выполнена (рисунок Д.1)
	Авторизация пользователя с некорректными данными. Логин: qweqweqwe, Пароль: qweqweqwe	Тест пройден. Показано сообщение с ошибкой (рисунок Д.2)
	Авторизация пользователя с незаполненным полем. Логин: wowspecial3, Пароль:	Тест пройден. Показано сообщение с ошибкой (рисунок Д.3)
Выставление баллов студенту	Выставление баллов студенту	Тест пройден. Авторизация выполнена (рисунок Д.4)
Просмотр успеваемости	Просмотр выставленных баллов	Тест пройден. Успеваемость выведена на экран (рисунок Д.5)
Создание пользователя	Создание пользователя	Тест пройден. Пользователь создан (рисунок Д.6)

Все тесты клиентской части были пройдены, что свидетельствует о правильной работе клиентской части.

5.3 Метрики кода

Метрики кода позволяют получить количественные величины, характеризующее программное обеспечение по тем или иным свойствам. Существует большое количество разнообразных метрик. Одной из самых простых метрик кода является количество строк кода (SLOC-метрик). Также одной из самых распространенных метрик, основанных на подсчете некоторых единиц в коде программы, относят метрики Холстеда.

Рассмотрим функцию, отвечающую за отрисовку страницы с выставлением баллов (листинг 1).

Листинг 1 – Код функции, отвечающий за страницу с выставлением баллов

```
const Marks: React.FC = () => {

    // получение данных о пользователе
    const {user} = useAppSelector(state => state.userReducer)
    const dispatch = useAppDispatch()

    //получение списка факультетов при изменени данных о пользователе
    useEffect(() => {
        if (user) {
            // функция получения списка факльтетов
           // Параметрами функции являются id пользователя и диспатч
            getFaculties(user!.id, dispatch).then()
        }
    }, [user]);

    // объекты с переменными состояния с информацией о содержании вкладок
    const {professorInputs, studentInputs} = useAppSelector(state =>
state.marksReducer)

    const changeFaculty = (e: ChangeEvent<HTMLSelectElement>) => {
        getFaculties(e.currentTarget.options.selectedIndex,
dispatch).then()
    }

    // заголовки и содержание выплывающих списков
    let professorInputsInfo = [
        { name: 'Факультет', options: professorInputs.faculties,
onChangeFunction: changeFaculty},
        { name: 'Кафедра', options: professorInputs.departments},
        { name: 'Специальность', options: professorInputs.specialities},
        { name: 'Группа', options: professorInputs.groups},
        { name: 'Семестр', options: professorInputs.semester},
        { name: 'Студент', options: professorInputs.student},
        { name: 'Предмет', options: professorInputs.subject},
    ]
    // Возвращаемая JSX разметка
    return (
```

```
        <div className={style.marks_wrapper}>
            {user?.role === 2 && (
                <div>
                    <h1>Выставить оценку студенту</h1>
                    {professorInputsInfo.map(input => (
                        <div>
                            <h2>{input.name}</h2>
                            <select className={style.option} onChange={e
=> input.onChangeFunction!(e)}>
                                {input.options.map(option => (
                                    <option
value={option}>{option}</option>
                                ))}
                            </select>
                        </div>
                    ))}
                    <h2>{'Оценка'}</h2>
                    <input type="number" className={style.option}/>
                    <div className={style.validation}>Баллы
выставлены</div>
                    <div>
                        <button className={style.marks_button}>Выставить
оценку</button>
                    </div>

                </div>
        </div>
    );
};
```

Для данной функции будут вычислены две группы метрик: метрики размера программы и метрики Холстеда.

5.3.1 Метрики размера программы

Данная группа метрик используется для измерения объема программы с помощью подсчета количества строк в тексте исходного кода. При этом часто выделяют методику подсчета физических строк кода и строк комментариев. Вычислим метрики размера программы для кода, приведенного в листинге 1 (таблица 26).

Таблица 26 – Метрики размера программы

Метрика	Количество
Количество строк кода	40
Количество строк комментариев	7

Теперь вычислим процент комментариев. Он равен (5/40)*100%=17,5%. Это означает, что количество комментариев в коде достаточно для его хорошего понимания, и при этом не слишком велико и не переполняет программу.

5.3.2 Метрики Холстеда

Метрика Холстеда вычисляется на основе анализа числа строк и синтаксических элементов исходного кода программы, а именно используются характеристики: число уникальных операторов, число уникальных операндов, общее число операторов и общее число операндов [30 – 32]. Для измерения данных характеристик необходимо составить словарь операторов (таблица 27) и словарь операндов (таблица 28)

Таблица 27 – Словарь операторов

Оператор	Число вхождений
=	25
.	27
if	1
return	1
!	2
,	23
{}	30
const	5
let	1

Таблица 28 – Словарь операндов

Операнд	Число вхождений
user	6
professorInputs	9
professorInputsInfo	2
dispatch	3
useEffect	1
useAppDispatch	1
useAppSelector	1
name	8
options	8

Из вышеописанных таблиц получаем следующие данные: число уникальных операторов программы n_1 = 9, число уникальных операндов программы n_2 = 9, общее число операторов N_1 = 115, общее число операндов в программе N_2 = 23.

Вычисли некоторые из существенных характеристик на основании полученных данных:

Расчёт длины программы рассчитывается по формуле:

$$N = N1 + N2 = 138,$$

где N1 – это общее количество операторов и операндов в программе (42);

N2 – это количество уникальных операторов и операндов (14).

Словарь программы (n) представляет собой количество уникальных операторов и операндов, представляющий расчёт суммы по формуле:

$$n = n1 + n2 = 18,$$

где n1 – это количество уникальных операторов (в данном случае 17);

n2 – количество уникальных операндов (в данном случае 7).

Объем программы рассчитывается по формуле

$$V = N \cdot \log_2 N \approx 980,$$

где N – длина программы;

n – словарь программы.

Сложность программы определяется как произведение двух частей n1 / 2 и N2 / n2 по формуле:

$$D = \left(\frac{n1}{2}\right) \cdot \left(\frac{N2}{n2}\right) \approx 11.5,$$

где n1 – количество уникальных операторов;

N2 – общее количество операндов;

n2 – количество уникальных операндов.

Вычисленные характеристики показывают, что программный код имеет достаточно высокий уровень качества и удобства для дальнейшего усовершенствования и поддержки.

Заключение

В результате выполнения работы над выпускной квалификационной работой бакалавра была изучена предметная область сферы web-приложений систем контроля успеваемости студентов.

На этапе проектирования была разработана концептуальная, логическая, функциональная модели базы данных для веб-приложения системы контроля успеваемости СПО. Было произведено проектирование серверной и клиентской частей приложения.

При реализации web-приложения для учета успеваемости студентов СПО была реализована база данных на основе физической модели данных, разработанной на этапе проектирования. В качестве СУБД была использована PostgreSQL 13. Для реализации серверной части была использована технология NodeJS, которая запускает JavaScript код на стороне сервера. Клиентская часть была создана с использование библиотеки React, которая предназначена для упрощения реализации сложных процессов при разработке клиентской части web-приложения.

Были подсчитаны следующие метрики: размера программы, Холстеда. Данные метрики показали, что написанный программный код является читабельным для разработчиков, а также код имеет достаточно высокий уровень качества и удобства для дальнейшего усовершенствования и поддержки.

Список используемых источников

1. ГОСТ Р ИСО 9241-151-2014. Руководство по проектированию пользовательских интерфейсов сети Интернет – [Электронный ресурс]. URL: https://docs.cntd.ru/document/1200113012 (дата обращения 12.05.2024).

2. ГОСТ Р 52872-2019. Интернет-ресурсы и другая информация, представленная в электронно-цифровой форме – [Электронный ресурс]. URL: https://tiflocentre.ru/download/gost-r-52872-2019.pdf (дата обращения: 07.05.2024).

3. ГОСТ 34.602-89. Техническое задание на создание автоматизированной системы. – [Электронный ресурс]. URL: https://docs.cntd.ru/document/1200006924 (дата обращения: 14.05.2024).

4. РД 50-34.698-90. Методические указания. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов. – [Электронный ресурс]. URL: https://docs.cntd.ru/document/1200006978 (дата обращения 08.05.2024).

5. Дональд Норман. Дизайн привычных вещей – [Электронный ресурс]. URL: https://www.mann-ivanov-ferber.ru/assets/files/bookparts/the-design-of-everyday-things/dpv_read.pdf (дата обращения: 06.05.2024).

6. Майкл Бейрут. Теперь вы это видите – [Электронный ресурс]. URL: https://www.litmir.me/br/?b=629186&p=1 (дата обращения: 07.05.2024).

7. Адам Д. Скотт. Разработка на JavaScript. Построение кроссплатформенных приложений с помощью GraphQL, React, React Native и Electron – [Электронный ресурс]. URL: https://ru.pk1lib.org/ireader/16855434 (дата обращения: 08.05.2024).

8. Фаулер Мартин. Рефакторинг кода на JavaScript. Улучшение проекта существующего кода (2019) – [Электронный ресурс] URL: https://fktpm.ru/file/61-fauler-refactoring-refaktoring-ulucsenie-sushhestvuyushhego-koda.pdf (дата обращения: 07.05.2024).

9. Джон Резиг, Биер Бибо. Секреты JavaScript ниндзя – [Электронный ресурс].URL: http://www.habucenter.ru/upload/iblock/76a/76a1fda775f9d864.pdf (дата обращения: 08.05.2024).

10. Эрик Фримэн, Элизабет Робсон. Изучаем программирование на JavaScript. – [Электронный ресурс]. URL: https://pdf.11klasov.net/16658-izuchaem-programmirovanie-na-javascript-frimen-je-robson-je.html (дата обращения: 08.05.2024).

11. Нейт Купер, Ким Джи. Как создать сайт. Комикс-путеводитель по HTML, CSS и WordPress – [Электронный ресурс]. URL: https://ftp.zhirov.website/books (дата обращения: 08.05.2024).

12. Джон Дакетт. HTML и CSS. Разработка и дизайн веб-сайтов – [Электронный ресурс]. URL: https://library-it.com/web/html/html-i-css-razrabotka-i-dizajn-veb-sajtov-2013/ (дата обращения: 09.05.2024).

13. Стефан Загмайстер, Джессика Уолш. О Красоте – [Электронный ресурс]. URL: https://www.labirint.ru/books/709227/#pdf (дата обращения: 09.04.2024).

14. Ричард Пулин. Школа дизайна: макет. Практическое руководство для студентов и дизайнеров – [Электронный ресурс]. URL: https://fileskachat.com/view (дата обращения: 09.05.2024).

15. Илья Сидоренко Дизайнер интерфейсов. Принципы работы и построение карьеры – [Электронный ресурс]. URL: https://iskros.com/book (дата обращения: 09.04.2024)

16. Ajax навигация с изменением http url. – [Электронный ресурс]. URL: https://ru.stackoverflow.com/questions (дата обращения: 10.05.2024).

17. Современный учебник JavaScript. – [Электронный ресурс]. URL: https://learn.javascript.ru/ (дата обращения 12.05.2024).

18. HTML5 reference – [Электронный ресурс]. URL: https://dev.w3.org/html5/html-author/ (дата обращения: 05.05.2024).

19. Справочник JavaScript – [Электронный ресурс]. URL: https://javascript.ru/manual (дата обращения: 09.05.2024).

Приложение А
(рекомендуемое)
Модели данных

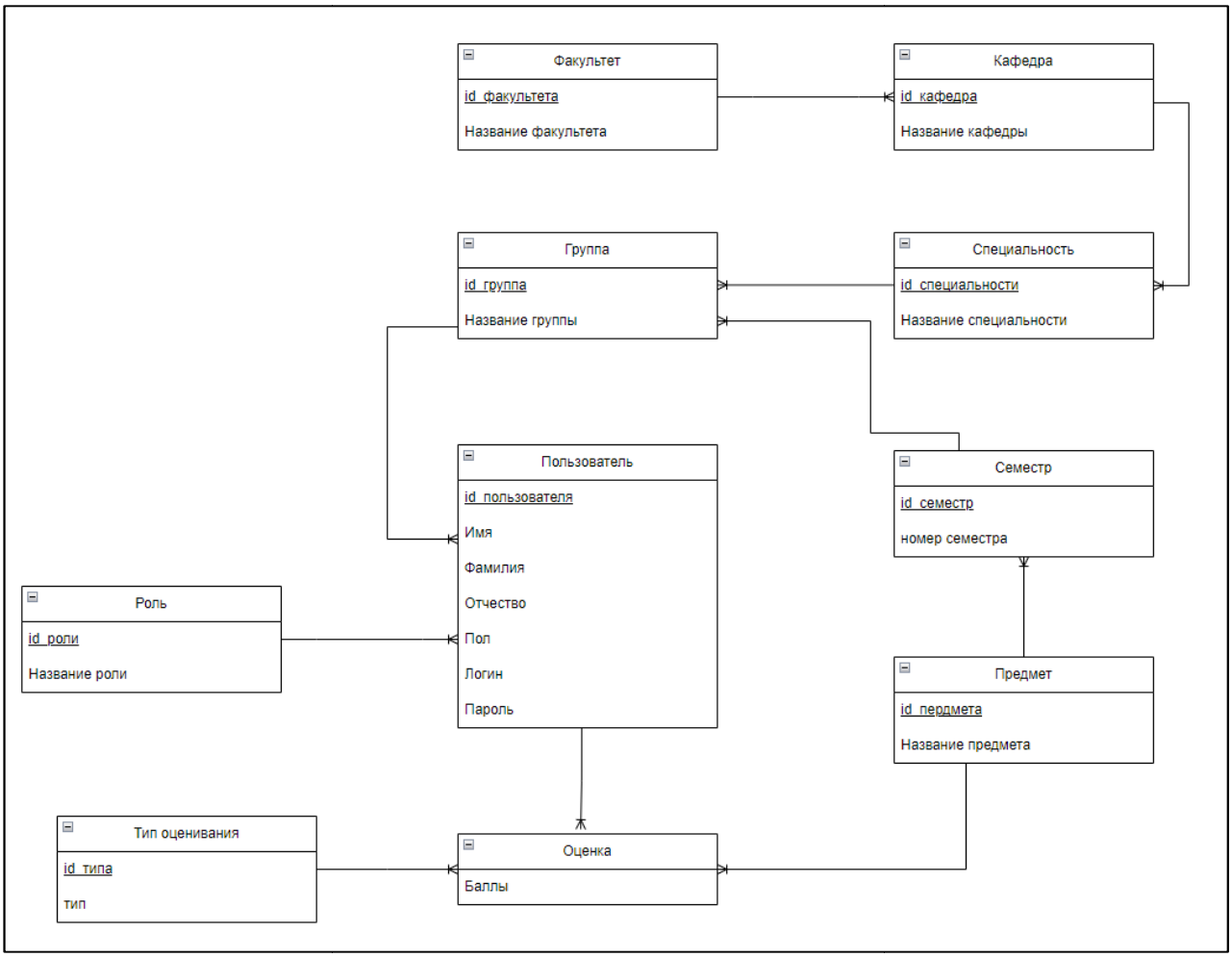


Рисунок А.1 – Концептуальная модель

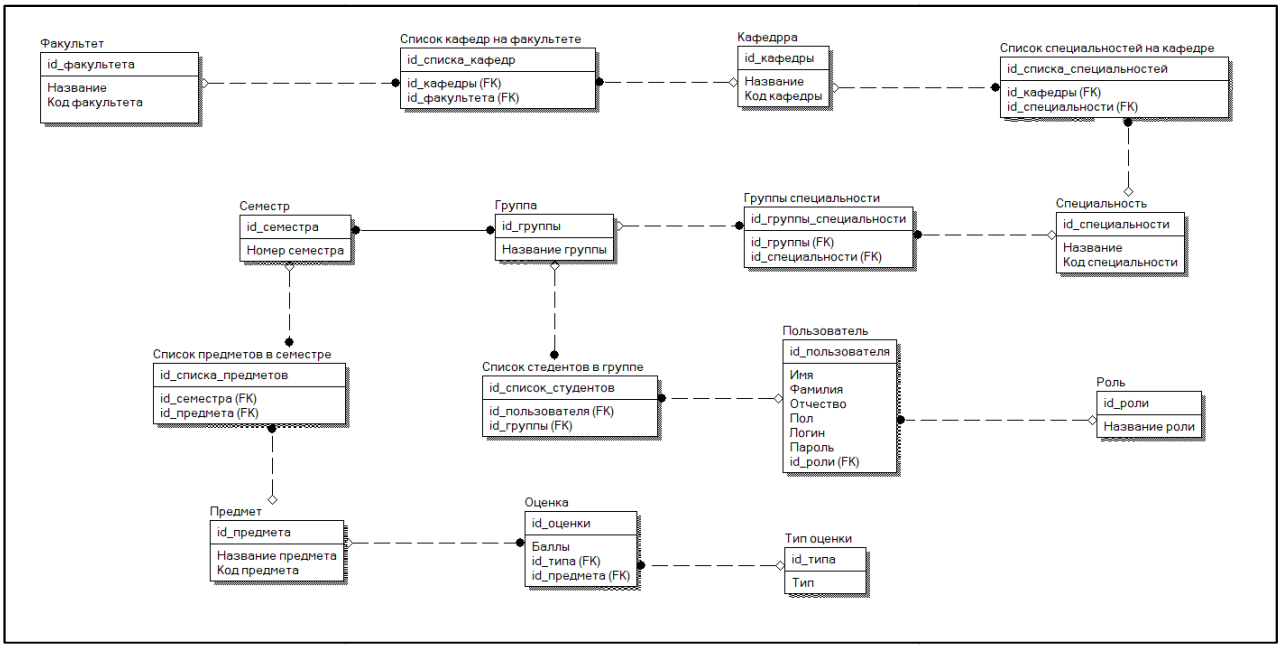


Рисунок А.2 – Логическая модель

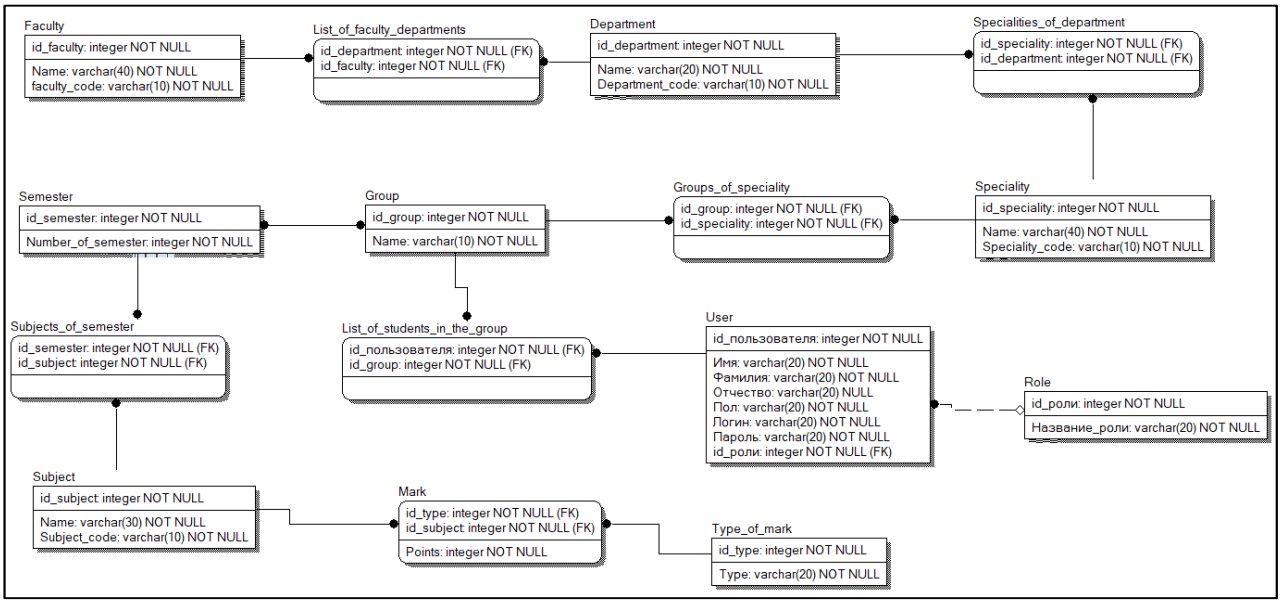


Рисунок А.3 – Физическая модель данных

Приложение Б

(рекомендуемое)

Код клиентской части

```
import React from 'react';
import './App.css';
import {BrowserRouter, Routes, Route} from "react-router-dom";
import LoginModalWindow from
"./components/loginModalWindow/loginModalWindow";
import Header from "./components/header/header";
import Marks from "./components/marks/marks";
import Profile from "./components/profile/profile";
import CreateUser from "./components/createUser/createUser";

function App() {

    return (
        <div className="App">
            <BrowserRouter>
                <LoginModalWindow />
                <Header />
                <Routes>
                    <Route path={`/marks`} element={<Marks />}></Route>
                    <Route path={`/profile`} element={<Profile
/>}></Route>
                    <Route path={`/create_user`} element={<CreateUser
/>}></Route>
                </Routes>
            </BrowserRouter>
        </div>
    );
}

export default App;

import React from 'react';
import style from './createUser.module.scss'

const CreateUser: React.FC = () => {
    return (
        <div className={style.createUser_wrapper}>
            <h1>Зарегистрировать пользователя</h1>
            <div>
                <h2>Имя пользователя</h2>
                <input type="text" className={style.option}/>

                <h2>Пароль</h2>
                <input type="password" className={style.option}/>

                <h2>Имя</h2>
                <input type="text" className={style.option}/>

                <h2>Фамилия</h2>
                <input type="text" className={style.option}/>

                <h2>Отчество</h2>
                <input type="text" className={style.option}/>

                <h2>Роль</h2>
                <select name="" id="" className={style.option}>
                    <option value="Студент">Студент</option>
                    <option
```

```
value="Преподаватьель">Преподаватьель</option>
                </select>
                <div className={style.validation}>Пользователь
зарегистрирован</div>
                <div>
                    <button
className={style.button}>Зарегистрировать</button>
                </div>

            </div>

        </div>
    );
};

export default CreateUser;
```

```
.createUser_wrapper {
  background: rgb(43, 43, 43);
  width: 900px;
  margin: 50px auto;
  padding: 50px;
  border: 3px solid #d3b9b9;
  border-radius: 30px;
  color: rgb(238, 238, 238);
}

.option {
  width: 250px;
  padding: 5px;
  border: 1px solid black;
  border-radius: 7px;
}

.button {
  padding: 7px;
  border: 1px solid black;
  border-radius: 7px;
  width: 150px;
  margin-top: 20px;
  font-size: 15px;
  transition: box-shadow 0.3s ease-in-out;
  &:hover {
    box-shadow: 0 0 5px rgb(220, 203, 203);
    cursor: pointer;
  }
}

.validation {
  margin-top: 20px;
  color: #f10404;
  font-size: 20px;
}
```

```
import React, {useEffect} from 'react';
import style from "./header.module.scss"
import {ReactComponent as Logo} from "../../svg/logo.svg";
import {NavLink} from "react-router-dom";
```

```
import {authorization} from "../../store/actionCreators/UserAC";
import Cookies from "universal-cookie";

const Header = () => {
    return (
        <header className={style.header}>
            <Logo width={'100'}/>
            <nav className={style.header_navigation_wrapper}>
                <ul className={style.header_navigation}>
                    <NavLink to={'/marks'}>
                        <li>Оценки</li>
                    </NavLink>
                    <NavLink to={'/profile'}>
                        <li>Профиль</li>
                    </NavLink>
                    <li>Выход</li>
                </ul>
            </nav>
        </header>
    );
};

export default Header;
```

```
.header {
  background: rgb(43, 43, 43);
  color: rgb(238, 238, 238);
  display: flex;
  flex-direction: row;
  justify-content: space-between;
}

.header_navigation_wrapper{
  display: flex;
  justify-content: center;
  .header_navigation {
    display: flex;
    flex-direction: row;
    list-style-type: none;
    margin: 0 20px 0 0;
    gap: 20px;
    font-size: 30px;
    align-items: center;
    a {
      text-decoration: none;
      color: rgb(238, 238, 238);
      cursor: pointer;
    }
  }
}
import React, {useState} from 'react';
import style from './loginModalWindow.module.scss'
import {useAppDispatch, useAppSelector} from "../../hooks/redux";
import {loginSlice} from "../../store/reducers/LoginSlice";
import {IValidation} from "../../models/IValidation";
import Cookies from "universal-cookie";
import {authorization, getUserById} from
"../../store/actionCreators/UserAC";
```

```
const LoginModalWindow: React.FC = () => {

    const dispatch = useAppDispatch()

    const [validation, setValidation] = useState<IValidation>({
        isInvalid: false,
        text: "Не работает",
    })

    const cookies = new Cookies()

    const isLogged = cookies.get("jwt") ? true : false
    console.log(isLogged)

    if (isLogged) {
        const jwt = cookies.get("jwt")
        const userId = Number(JSON.parse(atob(jwt.split('.')[1])).id)
        getUserById(Number(userId), dispatch).then()
    }

    const {username, password} = useAppSelector(state =>
state.loginReducer)
    const {setPasswordText, setUsernameText} = loginSlice.actions

    const inputs = [
        {header: "Имя пользователя", text: username, changeTextFunction:
setUsernameText, type: "text"},
        {header: "Пароль", text: password, changeTextFunction:
setPasswordText, type: "password"},
    ]

    const loginButtonHandler = async () => {

        const user_info = [username, password]
        const emptyInput = user_info.some(element => element === '' ?
true : false)

        if (emptyInput) {
            setValidation({
                isInvalid: true,
                text: "Не все поля были заполнены",
            })
            return
        }

        try {
            let [message] = await Promise.all([authorization(username,
password, dispatch)])

            if (message === undefined) {
                setPasswordText("")
            }
            else if (message === 400) {
                setValidation({
                    isInvalid: true,
                    text: "Неверный логин или пароль"
                })
```

```
            }
        } catch (e) {
            console.log(e)
            setValidation({
                isInvalid: true,
                text: "Неверный логин или пароль"
            })
        }
    }

    const validationReset = () => {
        setValidation({
            isInvalid: false,
            text: "Не работает"
        })
    }

    return (
        <div className={isLogged ? `${style.login_wrapper}
${style.closed}` : `${style.login_wrapper} ${style.open}`}>
            <div className={style.login_content}>
                <h2 className={style.login_header}>Авторизация</h2>
                {inputs.map(input => (
                    <div key={input.header}
className={style.login_inputs_block}>
                        <div
className={style.login_input_header}>{input.header}</div>
                        <input type={input.type}
className={style.login_input} value={input.text}
                               onClick={validationReset}
                               onChange={(e) =>
dispatch(input.changeTextFunction(e.target.value))}/>
                    </div>
                ))}
                {!validation.isInvalid && (
                    <div
className={style.validation}>{validation.text}</div>
                )}
                {validation.isInvalid && (
                    <div className={`${style.validation}
${style.validation_error}`}>{validation.text}</div>
                )}

                <button className={style.login_button}
onClick={loginButtonHandler}>Вход</button>
            </div>
        </div>
    );
};

export default LoginModalWindow;
```

```
.login_wrapper {
  position: fixed;
  background-color: rgba(0, 0, 0, .4);
  -webkit-backdrop-filter: blur(10px);
  backdrop-filter: blur(10px);
  width: 100%;
```

```
  height: 100%;
  left: 0;
  top: 0;
  right: 0;
  bottom: 0;
  opacity: 1;
  transition: opacity 0.2s;
}

.login_wrapper.open {
  pointer-events: all;
  opacity: 1;
}

.login_wrapper.closed {
  pointer-events: none;
  opacity: 0;
}

.login_content {
  background: rgb(238, 238, 238);
  color: rgba(0, 0, 0, 1);
  opacity: 1;
  max-width: 20%;
  min-width: 400px;
  margin: auto;
  margin-top: 15%;
  margin-bottom: 5%;
  padding: 20px;
  border-radius: 40px;
  transition: opacity 0.4s;
}

.login_button {
  background: rgb(238, 238, 238);
  display: flex;
  justify-content: center;
  align-items: center;
  margin: 0 auto;
  border: 1px solid black;
  border-radius: 40px;
  width: 200px;
  height: 40px;
  font-size: 20px;
  cursor: pointer;
  transition: box-shadow 0.3s ease-in-out;
  padding: 5px;
  &:hover {
    box-shadow: 0 0 5px rgba(0,0,0,1);
    cursor: pointer;
  }
}

.login_header {
  margin-bottom: 40px;
}

.login_inputs_block {
```

```
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding-bottom: 20px;
  .login_input_header {
    margin-bottom: 5px;
  }
  .login_input {
    padding: 7px;
    width: 300px;
    border-radius: 10px;
    border: 1px solid black;
  }
}

.validation {
  user-select: none;
  color: rgb(238, 238, 238);
  margin-bottom: 20px;
}

.validation_error {
  color: rgb(224, 19, 19)
}
```

```
import React, {ChangeEvent, useEffect} from 'react'
import {useAppDispatch, useAppSelector} from "../../hooks/redux";
import {getFaculties} from "../../store/actionCreators/MarksAC";
import style from './marks.module.scss'


const Marks: React.FC = () => {

    // получение данных о пользователе
    const {user} = useAppSelector(state => state.userReducer)
    const dispatch = useAppDispatch()

    //получение списка факультетов при изменени данных о пользователе
    useEffect(() => {
        if (user) {
            getFaculties(user!.id, dispatch).then()
        }
    }, [user]);

    // объекты с переменными состояния с информацией о содержании вкладок
    const {professorInputs, studentInputs} = useAppSelector(state =>
state.marksReducer)

    const changeFaculty = (e: ChangeEvent<HTMLSelectElement>) => {
        getFaculties(e.currentTarget.options.selectedIndex,
dispatch).then()
    }

    // заголовки и содержание выплывающих списков
    let professorInputsInfo = [
        { name: 'Факультет', options: professorInputs.faculties,
onChangeFunction: changeFaculty},
```

```
        { name: 'Кафедра', options: professorInputs.departments},
        { name: 'Специальность', options: professorInputs.specialities},
        { name: 'Группа', options: professorInputs.groups},
        { name: 'Семестр', options: professorInputs.semester},
        { name: 'Студент', options: professorInputs.student},
        { name: 'Предмет', options: professorInputs.subject},
    ]

    return (
        <div className={style.marks_wrapper}>
            {user?.role === 2 && (
                <div>
                    <h1>Выставить оценку студенту</h1>
                    {professorInputsInfo.map(input => (
                        <div>
                            <h2>{input.name}</h2>
                            <select className={style.option} onChange={e
=> input.onChangeFunction!(e)}>
                                {input.options.map(option => (
                                    <option
value={option}>{option}</option>
                                ))}
                            </select>
                        </div>
                    ))}
                    <h2>{'Оценка'}</h2>
                    <input type="number" className={style.option}/>
                    <div className={style.validation}>Баллы
выставлены</div>
                    <div>
                        <button className={style.marks_button}>Выставить
оценку</button>
                    </div>

                </div>
            )}
            {user?.role === 1 && (
                <div>
                    <div>
                        <h2>Группа</h2>
                        <select className={style.option}>
                            <option value="20ВЭ1">20ВЭ1</option>
                        </select>

                        <h2>Семестр</h2>
                        <select className={style.option}>
                            <option value="2">2</option>
                        </select>

                        <h2>Прпедмет</h2>
                        <select className={style.option}>
                            <option value="Объектно-ориентированное
программирование">Объектно-ориентированное программирование</option>
                        </select>
                    </div>
                    <table className={style.marks_table}>
                        <thead>
                        </thead>
```

```
                        <tbody>
                        <tr>
                            <td>Лаб. работы</td>
                            <td>12</td>
                            <td></td>
                            <td></td>
                            <td></td>
                            <td></td>
                            <td></td>
                            <td></td>
                            <td></td>
                        </tr>
                        <tr>
                            <td>Зачет</td>
                            <td></td>
                        </tr>
                        <tr>
                            <td>Экзамен</td>
                            <td></td>
                        </tr>
                        <tr>
                            <td>Курсовой проект</td>
                            <td></td>
                        </tr>
                        </tbody>
                    </table>
                </div>
            )}
        </div>
    );
};

export default Marks;
.option {
  width: 250px;
  padding: 5px;
  border: 1px solid black;
  border-radius: 7px;
}

.marks_wrapper {
  background: rgb(43, 43, 43);
  width: 900px;
  margin: 50px auto;
  padding: 50px;
  border: 3px solid #d3b9b9;
  border-radius: 30px;
  color: rgb(238, 238, 238);

  .marks_button {
    padding: 7px;
    border: 1px solid black;
    border-radius: 7px;
    width: 150px;
    margin-top: 20px;
    font-size: 15px;
    transition: box-shadow 0.3s ease-in-out;
    &:hover {
```

```
      box-shadow: 0 0 5px rgb(220, 203, 203);
      cursor: pointer;
    }
  }
}

.validation {
  margin-top: 20px;
  color: #f10404;
  font-size: 20px;
}

.marks_table {

  margin: 40px auto 0;
  table-layout: fixed;
  border: 1px rgb(238, 238, 238) solid;
  border-collapse: collapse;
  td, th {
    border: 1px rgb(238, 238, 238) solid;
    width: 60px;
    height: 60px;
  }
}

import React from 'react';
import style from './profile.module.scss'
import {useAppSelector} from "../../hooks/redux";
const Profile: React.FC = () => {

    const {user} = useAppSelector(style => style.userReducer)
    console.log(user)
    return (
        <div className={style.profile_wrapper}>
            <h1>{user?.secondname} {user?.name} {user?.surname}</h1>
        </div>
    );
};

export default Profile;

import {createSlice, PayloadAction} from "@reduxjs/toolkit";

export interface LoginState {
    username: string,
    password: string,
}

const initialState: LoginState = {
    username: '',
    password: '',
}

export const loginSlice = createSlice({
    name: 'login',
    initialState,
    reducers: {
        setUsernameText(state, action: PayloadAction<string>) {
```

```
            state.username = action.payload
        },
        setPasswordText(state, action: PayloadAction<string>) {
            state.password = action.payload
        },
    }
})

export default loginSlice.reducer

import {createSlice, PayloadAction} from "@reduxjs/toolkit";

interface IMarksState {
    professorInputs: {
        faculties: string[],
        departments: string[],
        specialities: string[],
        groups: string[],
        semester: string[],
        student: string[],
        subject: string[],
        markType: string[],
        points: string,
    },
    studentInputs: {
        groups: string[],
        semester: string[],
        subject: string[],
    },
}

const initialState: IMarksState = {
    professorInputs: {
        faculties: [],
        departments: [],
        specialities: [],
        groups: [],
        semester: [],
        student: [],
        subject: [],
        markType: [],
        points: '',
    },
    studentInputs: {
        groups: [],
        semester: [],
        subject: [],
    }
}

export const marksSlice = createSlice({
    name: 'professorMarks',
    initialState,
    reducers: {
        facultiesFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.faculties = action.payload
        },
        departmentsFetching(state, action: PayloadAction<string[]>) {
```

```
            state.professorInputs.departments = action.payload
        },
        specialitiesFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.specialities = action.payload
        },
        groupsFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.groups = action.payload
        },
        inputsSemestersFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.semester = action.payload
        },
        studentsFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.student = action.payload
        },
        subjectsFetching(state, action: PayloadAction<string[]>) {
            state.professorInputs.subject = action.payload
        },
        setPointsValue(state, action: PayloadAction<string>) {
            state.professorInputs.points = action.payload
        },
    }
})

export default marksSlice.reducer
import {IUser} from "../../models/IUser";
import {createAsyncThunk, createSlice, PayloadAction} from
"@reduxjs/toolkit";

interface UserState {
    user: IUser | null,
    isLoading: boolean,
    error: string,
}

const initialState: UserState = {
    user: null,
    isLoading: false,
    error: '',
}

export const userSlice = createSlice({
    name: 'user',
    initialState,
    reducers: {
        usersFetching(state) {
            state.isLoading = true
        },
        usersFetchingSuccess(state, action: PayloadAction<IUser>) {
            state.isLoading = false
            state.error = ''
            state.user = action.payload
        },
        usersFetchingError(state, action: PayloadAction<string>) {
            state.isLoading = false
            state.error = action.payload
        }
    },
```

```
})

export default userSlice.reducer
import {combineReducers, configureStore} from "@reduxjs/toolkit";
import userReducer from './reducers/UserSlice'
import loginReducer from './reducers/LoginSlice'
import marksReducer from './reducers/MarksSlice'

const rootReducer = combineReducers({
    userReducer,
    loginReducer,
    marksReducer,
})

export const setupStore = () => {
    return configureStore({
        reducer: rootReducer
    })
}

export type RootState = ReturnType<typeof rootReducer>
export type AppStore = ReturnType<typeof setupStore>
export type AppDispatch = AppStore['dispatch']
import {AppDispatch} from "../store";
import axios from "axios";
import {IUser} from "../../models/IUser";
import {api_url} from "../../api_url";
import {userSlice} from "../reducers/UserSlice";
import Cookies from "universal-cookie";
const cookies = new Cookies()
const {usersFetchingSuccess} = userSlice.actions
export const authorization = async (username: string, password: string,
dispatch: AppDispatch) => {
    //const {usersFetchingSuccess} = userSlice.actions
    let user_id

    try {
        await axios.post(`${api_url}/api/login`, {
            username: username,
            password: password
        }).then(async res => {
            cookies.set("jwt", res.data, {path: '/'})
            user_id = JSON.parse(atob(res.data.split('.')[1]));
            console.log(user_id)
            const user = await
axios.get<IUser>(`${api_url}/api/user/get_one/${user_id.id}`)
            dispatch(usersFetchingSuccess(user.data))
        }).catch(error => {
            if (error.response) {
                throw (error.response.status)
            }
        })
    } catch (e) {
        throw e
    }
}

export const getUserById = async (userID: number, dispatch: AppDispatch)
```

```
=> {
    try {
        await
axios.get<IUser>(`${api_url}/api/user/get_one/${userID}`).then(res => {
            dispatch(usersFetchingSuccess(res.data))
        })
    } catch (e) {
        throw e
    }
}

import {AppDispatch} from "../store";
import {marksSlice} from "../reducers/MarksSlice";
import axios from "axios";
import {api_url} from "../../api_url";

export const getFaculties = async (userId: number, dispatch: AppDispatch)
=> {
    const {facultiesFetching} = marksSlice.actions

    try {
        const faculties = await
axios.get(`${api_url}/api/faculty/get_professor_faculties/${userId}`).the
n(res => {
            dispatch(facultiesFetching(res.data))
            getDepartments(res.data[0], dispatch).then()
        })
    } catch (e) {
        throw e
    }
}

export const getDepartments = async (facultyName: string, dispatch:
AppDispatch) => {
    const {departmentsFetching} = marksSlice.actions

    try {
        await
axios.get(`${api_url}/api/institute/getByFacultyName/${facultyName}`).the
n(res => {
            dispatch(departmentsFetching(res.data))
            getSpecialities(res.data[0], dispatch).then()
        })
    } catch (e) {
        throw (e)
    }
}

export const getSpecialities = async (departmentName: string, dispatch:
AppDispatch) => {
    const {specialitiesFetching} = marksSlice.actions

    try {
        await
axios.get(`${api_url}/api/speciality/getByDepartmentName/${departmentName
}`).then(res => {
            dispatch(specialitiesFetching(res.data))
            getGroups(res.data[0], dispatch).then()
```

```
        })
    } catch (e) {
        throw (e)
    }
}

export const getGroups = async (specialityName: string, dispatch:
AppDispatch) => {
    const {groupsFetching} = marksSlice.actions

    try {
        await
axios.get(`${api_url}/api/group/getGroupsBySpecialtyName/${specialityName
}`).then(res => {
            dispatch(groupsFetching(res.data))
            getSemesters(res.data[0], dispatch).then()
            getStudents(res.data[0], dispatch)
        })
    } catch (e) {
        throw (e)
    }
}

export const getSemesters = async (groupName: string, dispatch:
AppDispatch) => {
    const {inputsSemestersFetching, subjectsFetching} =
marksSlice.actions

    try {
        await
axios.get(`${api_url}/api/semester/getByGroupId/${groupName}`).then(async
res => {
            dispatch(inputsSemestersFetching(res.data))
            await
axios.get(`${api_url}/api/semester/getSubjects/${res.data[0]}_${groupName
}`).then(res => {
                dispatch(subjectsFetching(res.data))
            })
        })
    } catch (e) {
        throw e
    }
}

export const getStudents = async (groupName: string, dispatch:
AppDispatch) => {
    const {studentsFetching} = marksSlice.actions

    try {
        await
axios.get(`${api_url}/api/user/get_students_by_group/${groupName}`).then(
res => {
            dispatch(studentsFetching(res.data))
        })
    } catch (e) {
        throw e
    }
}
```

Приложение В

(рекомендуемое)

Код серверной части

```
const express = require('express')
const cors = require('cors')

const facultyRouter = require('./routes/facultyRouter')
const instituteRouter = require('./routes/instituteRouter')
const specialityRouter = require('./routes/specialityRouter')
const groupRouter = require('./routes/groupRouter')
const roleRouter = require('./routes/roleRouter')
const userRouter = require('./routes/userRouter')
const subjectRouter = require('./routes/subjectRouter')
const semesterRouter = require('./routes/semesterRouter')

const PORT = process.env.PORT || 5000

const app = express()

app.use(cors())
app.use(express.json())

app.use('/api', facultyRouter)
app.use('/api', instituteRouter)
app.use('/api', specialityRouter)
app.use('/api', groupRouter)
app.use('/api', roleRouter)
app.use('/api', userRouter)
app.use('/api', subjectRouter)
app.use('/api', semesterRouter)

const start = () => {
    try {
        app.listen(PORT, () => console.log(`server started on port
${PORT}`))
    } catch (e) {
        console.log(e)
    }
}

start()
const db = require('../db')

class facultyController {
    async createFaculty (req, res) {
        const {name} = req.body
        const faculty = await db.query(`INSERT INTO faculty (name) VALUES
($1) RETURNING *`, [name])
        res.json(faculty.rows[0])
    }

    async getOneFaculty (req, res) {
        const id = req.params.id
        const faculty = await db.query(`SELECT * FROM faculty WHERE id =
$1`, [id])
        res.json(faculty.rows[0])
    }

    async getAllFaculty(req, res) {
        const faculties = await db.query(`SELECT * FROM faculty`)
        res.json(faculties.rows)
```

```
    }

    async updateFaculty (req, res) {
        const {id, name} = req.body
        const updatedFaculty = await db.query(`UPDATE faculty SET name =
$2 WHERE id = $1 RETURNING *`, [id, name])
        res.json(updatedFaculty.rows[0])
    }

    async deleteFaculty (req, res) {
        const id = req.params.id
        const deletedFaculty = await db.query(`DELETE FROM faculty WHERE
id = $1`, [id])
        res.json(deletedFaculty.rows[0])
    }

    async addInstitute (req, res) {
        const {facultyID, instituteID} = req.body
        const addedInstitute = await db.query(`INSERT INTO
faculty_department (facultyid, instituteid) VALUES ($1, $2) RETURNING *`,
[facultyID, instituteID])
        res.json(addedInstitute.rows[0])
    }

    async removeInstitute (req, res) {
        const {facultyID, instituteID} = req.body
        const removedInstitute = await db.query(`DELETE FROM
faculty_institute WHERE facultyid = $1 AND instituteid = $2`, [facultyID,
instituteID])
        res.json(removedInstitute.rows[0])
    }

    async getProfessorFaculties (req, res) {
        const professorID = req.params.id
        console.log(professorID)
        const professorDepartments = await db.query(`SELECT * FROM
professors_in_department WHERE id_user = $1`, [professorID])

        let facultiesID = []
        for (let i = 0; i < professorDepartments.rows.length; i++) {
            const facultyID = await db.query(`SELECT * FROM
faculty_department WHERE instituteid = $1`,
[professorDepartments.rows[i].id_department])
            for (let j = 0; j < facultyID.rows.length; j++) {
                facultiesID.push(facultyID.rows[j].facultyid)
            }
        }
        console.log(facultiesID)

        let facultiesNames = []
        for (let i = 0; i < facultiesID.length; i++) {
            const faculty = await db.query(`SELECT * FROM faculty WHERE
id = $1`, [facultiesID[i]])
            facultiesNames.push(faculty.rows[0].name)
        }
        console.log(facultiesNames)
        res.json(facultiesNames)
    }
```

```
}

module.exports = new facultyController()

const db = require('../db')

class groupController {
    async createGroup (req, res) {
        const {name} = req.body
        const group = await db.query(`INSERT INTO class (name) VALUES
($1) RETURNING *`, [name])
        res.json(group.rows[0])
    }

    async getOneGroup (req, res) {
        const id = req.params.id
        const group = await db.query(`SELECT * FROM class WHERE id = $1`,
[id])
        res.json(group.rows[0])
    }

    async getAllGroup (req, res) {
        const groups = await db.query(`SELECT * FROM class`)
        res.json(groups.rows)
    }

    async updateGroup (req, res) {
        const {id, name} = req.body
        const updatedGroup = await db.query(`UPDATE class SET name = $2
WHERE id = $1 RETURNING *`, [id, name])
        res.json(updatedGroup.rows[0])
    }

    async deleteGroup (req, res) {
        const id = req.params.id
        const deletedGroup = await db.query(`DELETE FROM class WHERE id =
$1`, [id])
        res.json(deletedGroup.rows[0])
    }

    async getSpecialityGroups (req, res) {
        const specialityName = req.params.id
        console.log(specialityName)
        const speciality = await db.query(`SELECT * from specialty WHERE
name = $1`, [specialityName])

        const groupId = await db.query(`SELECT * from speciality_calss
WHERE specialityid = $1`, [speciality.rows[0].id])

        let groupsName = []

        for (let i = 0; i < groupId.rows.length; i++) {
            const group = await db.query(`SELECT * from class WHERE id =
$1`, [groupId.rows[i].classid])
            groupsName.push(group.rows[0].name)
        }
        console.log(groupsName)
        res.json(groupsName)
```

```
    }
}

module.exports = new groupController()

const db = require('../db')

class instituteController {
    async createInstitute (req, res) {
        const {name} = req.body
        const sport = await db.query(`INSERT INTO department (name)
VALUES ($1) RETURNING *`, [name])
        res.json(sport.rows[0])
    }

    async getOneInstitute (req, res) {
        const id = req.params.id
        const institute = await db.query(`SELECT * FROM institute WHERE
id = $1`, [id])
        res.json(institute.rows[0])
    }

    async getAllInstitute (req, res) {
        const institutions = await db.query(`SELECT * FROM institute`)
        res.json(institutions.rows)
    }

    async updateInstitute (req, res) {
        const {id, name} = req.body
        const updatedInstitute = await db.query(`UPDATE institute SET
name = $2 WHERE id = $1 RETURNING *`, [id, name])
        res.json(updatedInstitute.rows[0])
    }

    async deleteInstitute (req, res) {
        const id = req.params.id
        const deletedInstitute = await db.query(`DELETE FROM department
WHERE id = $1`, [id])
        res.json(deletedInstitute.rows[0])
    }

    async addSpeciality (req, res) {
        const {instituteID, specialityID} = req.body
        const addedInstitute = await db.query(`INSERT INTO
department_specialty (department, specialtyid) VALUES ($1, $2) RETURNING
*`, [instituteID, specialityID])
        res.json(addedInstitute.rows[0])
    }

    async removeSpeciality (req, res) {
        const {facultyID, instituteID} = req.body
        const removedInstitute = await db.query(`DELETE FROM
institute_speciality WHERE instituteid = $1 AND specialityid = $2`,
[instituteID, specialityID])
        res.json(removedInstitute.rows[0])
    }

    async getInstitutesInFaculty (req, res) {
```

```
        const facultyName = req.params.id
        console.log(facultyName)
        const faculty = await db.query(`SELECT * from faculty WHERE name
= $1`, [facultyName])
        const institutesId = await db.query(`SELECT * from
faculty_department WHERE facultyid = $1`, [faculty.rows[0].id])

        let institutesName = []
        for (let i = 0; i < institutesId.rows.length; i++) {
            const institute = await db.query(`SELECT * from department
WHERE id = $1`, [institutesId.rows[i].instituteid])
            institutesName.push(institute.rows[0].name)
        }
        console.log(institutesName)
        res.json(institutesName)
    }
}

module.exports = new instituteController()

const db = require('../db')

class roleController {
    async getRole (req, res) {
        const id = req.params.id
        const role = await db.query(`SELECT * FROM role WHERE id = $1`,
[id])
        res.json(role.rows[0])
    }
}

module.exports = new roleController()

const db = require('../db')

class semesterController {
    async createSemester (req, res) {
        const {number, groupid} = req.body
        const group = await db.query(`INSERT INTO semester (number,
groupid) VALUES ($1, $2) RETURNING *`, [number, groupid])
        res.json(group.rows[0])
    }

    async getSemestersByGroupId (req, res) {
        const name = req.params.id
        const group = await db.query(`SELECT * FROM class WHERE name =
$1`, [name])
        const semesters = await db.query(`SELECT * FROM semester WHERE
groupid = $1`, [group.rows[0].id])

        let semesterNumbers = []
        for (let i = 0; i < semesters.rows.length; i++) {
            semesterNumbers.push(semesters.rows[i].number)
        }

        res.json(semesterNumbers)
    }
```

```
    async getSubjects (req, res) {
        const semesterNumber_groupName = req.params.id
        const semesterNumber = semesterNumber_groupName.split('_')[0]
        const groupName = semesterNumber_groupName.split('_')[1]

        const group = await db.query(`SELECT * FROM class WHERE name =
$1`, [groupName])

        const semester = await  db.query(`SELECT * FROM semester WHERE
groupid = $1 AND number = $2`, [group.rows[0].id, semesterNumber])

        const semesterSubjects = await db.query(`SELECT * FROM
semester_subject WHERE semesterid = $1`, [semester.rows[0].id])

        const subjects = []
        for (let i = 0; i < semesterSubjects.rows.length; i++) {
            const subject = await db.query(`SELECT * FROM subject WHERE
id = $1`, [semesterSubjects.rows[i].subject])
            subjects.push(subject.rows[0].name)
        }

        res.json(subjects)

    }
}

module.exports = new semesterController()

const db = require('../db')

class specialityController {
    async createSpeciality (req, res) {
        const {name} = req.body
        const speciality = await db.query(`INSERT INTO specialty (name)
VALUES ($1) RETURNING *`, [name])
        res.json(speciality.rows[0])
    }

    async getOneSpeciality (req, res) {
        const id = req.params.id
        const speciality = await db.query(`SELECT * FROM speciality WHERE
id = $1`, [id])
        res.json(speciality.rows[0])
    }

    async getAllSpeciality (req, res) {
        const specialties = await db.query(`SELECT * FROM speciality`)
        res.json(specialties.rows[0])
    }

    async updateSpeciality (req, res) {
        const {id, name} = req.body
        const updatedSpecialties = await db.query(`UPDATE speciality SET
name = $2 WHERE id = $1 RETURNING *`, [id, name])
        res.json(updatedSpecialties.rows[0])
    }

    async deleteSpeciality (req, res) {
```

```
        const id = req.params.id
        const deletedSpeciality = await db.query(`DELETE FROM specialty
WHERE id = $1`, [id])
        res.json(deletedSpeciality.rows[0])
    }

    async addGroup (req, res) {
        const {specialityID, groupID} = req.body
        const addedInstitute = await db.query(`INSERT INTO
speciality_calss (specialityid, classid) VALUES ($1, $2) RETURNING *`,
[specialityID, groupID])
        res.json(addedInstitute.rows[0])
    }

    async removeGroup (req, res) {
        const {specialityID, groupID} = req.body
        const removedInstitute = await db.query(`DELETE FROM
speciality_class WHERE specialityid = $1 AND groupid = $2`,
[specialityID, groupID])
        res.json(removedInstitute.rows[0])
    }

    async getSpecialitiesInInstitute (req, res) {
        const instituteName = req.params.id
        console.log(instituteName)
        const institute = await db.query(`SELECT * from department WHERE
name = $1`, [instituteName])

        const specialitiesId = await db.query(`SELECT * from
department_specialty WHERE specialtyid = $1`, [institute.rows[0].id])

        let specialitiesName = []

        for (let i = 0; i < specialitiesId.rows.length; i++) {
            const speciality = await db.query(`SELECT * from specialty
WHERE id = $1`, [specialitiesId.rows[i].specialtyid])
            specialitiesName.push(speciality.rows[0].name)
        }
        console.log(specialitiesName)
        res.json(specialitiesName)
    }
}

module.exports = new specialityController()

const db = require('../db')

class subjectController {
    async createSubject (req, res) {
        const {name} = req.body
        const subject = await db.query(`INSERT INTO subject (name) VALUES
($1) RETURNING *`, [name])
        res.json(subject.rows[0])
    }

    async getSubject (req, res) {
        const id = req.params.id
        const subject = await db.query(`SELECT * FROM subject WHERE id =
```

```
$1`, [id])
        res.json(subject.rows[0])
    }

    async getAllSubjects (req, res) {
        const subjects = await db.query(`SELECT * FROM subject`)
        res.json(subjects.rows[0])
    }

    async deleteSubject (req, res) {
        const id = req.params.id
        const deletedSubject = await db.query(`DELETE FROM subject WHERE
id = $1`, [id])
        res.json(deletedSubject.rows[0])
    }
}

module.exports = new subjectController()

const db = require('../db')
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {validationResult} = require('express-validator')
const {secret} = require('../config')
const generateAccessToken = (id, roles) => {
    const payload = {
        id: id,
        role: roles,
    }
    return jwt.sign(payload, secret, {expiresIn: "24h"})
}
class userController {

    async registration (req, res) {
        try {
            const errors = validationResult(req)
            if(!errors.isEmpty()) {
                return res.status(400).json({message: 'Ошибка при
регистрации', errors})
            }
            const {name, surname, secondName, username, password} =
req.body
            const user = await db.query(`SELECT * FROM users WHERE
username = $1`, [username])
            if (user.rows.length !== 0) {
                console.log(user.rows)
                return res.status(400).json({message: 'Пользователь уже
существует'})
            }
            const hashPassword = bcrypt.hashSync(password, 7)
            const new_user = await db.query('INSERT INTO users (name,
surname, secondName, role, username, password) VALUES ($1, $2, $3, $4,
$5, $6)',
                [name, surname, secondName, 1, username, hashPassword])
            res.json("пользователь зарегистрирован")
            res.json(new_user.rows[0])
        } catch (e) {
            console.log(e)
```

```
            return res.status(400).json({message: "Registration error"})
        }
    }

    async login (req, res) {
        try {
            const {username, password} = req.body
            const user = await db.query(`SELECT * FROM users WHERE
username = $1`, [username])
            if (user.rows.length === 0) {
                return res.status(400).json({message: `Пользователь
'${username}' не существует`})
            }
            const validPassword = bcrypt.compareSync(password,
user.rows[0].password)
            if (!validPassword) {
                return res.status(400).json({message: `Введен неверный
пароль`})
            }

            const token = generateAccessToken(user.rows[0].id, "user")
            console.log(token)
            return res.json(token)
        } catch (e) {
            console.log(e)
            res.status(400).json({message: "Login error"})
        }
    }
    async createStudent (req, res) {
        const {name, surname, secondName, groupID} = req.body
        const student = await db.query(`INSERT INTO users (name, surname,
secondname, role) VALUES ($1, $2, $3, 1) RETURNING *`, [name, surname,
secondName])
        const addToGroup = await db.query(`INSERT INTO class_users
(classid, studentid) VALUES ($1, $2)`, [groupID, student.rows[0].id])
        res.json(student.rows[0])
    }

    async createProfessor (req, res) {
        const {name, surname, secondName} = req.body
        const professor = await db.query(`INSERT INTO users (name,
surname, secondname, role) VALUES ($1, $2, $3, 2) RETURNING *`, [name,
surname, secondName])
        res.json(professor.rows[0])
    }

    async getOneUser (req, res) {
        const id = req.params.id
        const user = await db.query(`SELECT * FROM users WHERE id = $1`,
[id])
        res.json(user.rows[0])
    }

    async getAllUsers(req, res) {
        const users = await db.query(`SELECT * FROM users`)
        res.json(users.rows)
    }
```

```
    async getStudentsByGroup (req, res) {
        const groupName = req.params.id
        const group = await db.query(`SELECT * FROM class WHERE name =
$1`, [groupName])
        const class_students = await db.query(`SELECT * FROM class_users
WHERE classid = $1`, [group.rows[0].id])

        let students = []
        for (let i = 0; i < class_students.rows.length; i++) {
            const student = await db.query(`SELECT * FROM users WHERE id
= $1`, [class_students.rows[i].userid])
            console.log(student.rows)
            students.push(student.rows[0].surname + ' ' +
student.rows[0].name + ' ' + student.rows[0].secondname)
        }
        //console.log(students)
        res.json(students)
    }

    async updateUser (req, res) {
        const {name, surname, secondName, studentID} = req.body
        const updatedUser = await db.query(`UPDATE users SET name = $1,
surname = $2, secondName = $3 WHERE id = $4 RETURNING *`, [name, surname,
secondName, studentID])
        res.json(updatedUser.rows[0])
    }

    async updateUserRole (req,res) {
        const {userID, role} = req.body
        const updatedUser = await db.query(`UPDATE users SET role = $1
WHERE id = $2`, [role, userID])
    }

    async deleteStudent (req, res) {
        const id = req.params.id
        await db.query(`DELETE FROM class_users WHERE studentid = $1`,
[id])
        const deletedStudent = await db.query(`DELETE FROM users WHERE id
= $1`, [id])
        res.json(deletedStudent.rows[0])
    }

    async deleteUser (req, res) {
        const id = req.params.id
        const deletedUsers = await db.query(`DELETE FROM users WHERE id =
$1`, [id])
        res.json(deletedUsers.rows[0])
    }

}


module.exports = new userController()

const Router = require('express')
const router = new Router()
const facultyController = require('../controller/facultyController')
```

```
router.post('/faculty/create', facultyController.createFaculty)
router.get('/faculty/get_all', facultyController.getAllFaculty)
router.get('/faculty/get_one/:id', facultyController.getOneFaculty)
router.put('/faculty/update', facultyController.updateFaculty)
router.delete('/faculty/delete/:id', facultyController.deleteFaculty)

router.post('/faculty/add_institute', facultyController.addInstitute)
router.delete('/faculty/remove_institute',
facultyController.removeInstitute)

router.get('/faculty/get_professor_faculties/:id',
facultyController.getProfessorFaculties)

module.exports = router

const Router = require('express')
const router = new Router()
const groupController = require('../controller/groupController')

router.post('/group/create', groupController.createGroup)
router.get('/group/get_all', groupController.getAllGroup)
router.get('/group/get_one/:id', groupController.getOneGroup)
router.put('/group/update', groupController.updateGroup)
router.delete('/group/delete/:id', groupController.deleteGroup)

router.get('/group/getGroupsBySpecialtyName/:id',
groupController.getSpecialityGroups)

module.exports = router

const Router = require('express')
const router = new Router()
const instituteController = require('../controller/instituteController')


router.post('/institute/create', instituteController.createInstitute)
router.get('/institute/get_all', instituteController.getAllInstitute)
router.get('/institute/get_one/:id', instituteController.getOneInstitute)
router.put('/institute/update', instituteController.updateInstitute)
router.delete('/institute/delete/:id',
instituteController.deleteInstitute)

router.post('/institute/add_speciality',
instituteController.addSpeciality)
router.delete('/institute/remove_speciality',
instituteController.removeSpeciality)

router.get('/institute/getByFacultyName/:id',
instituteController.getInstitutesInFaculty)


module.exports = router

const Router = require('express')
const router = new Router()
const roleController = require('../controller/roleController')

router.get('/role/get/:id', roleController.getRole)
```

```
module.exports = router

const Router = require('express')
const router = new Router()
const semesterController = require('../controller/semesterController')

router.post('/semester/create', semesterController.createSemester)
router.get('/semester/getByGroupId/:id',
semesterController.getSemestersByGroupId)
router.get('/semester/getSubjects/:id', semesterController.getSubjects)

module.exports = router

const Router = require('express')
const router = new Router()
const specialityController =
require('../controller/specialityController')

router.post('/speciality/create', specialityController.createSpeciality)
router.get('/speciality/get_all', specialityController.getAllSpeciality)
router.get('/speciality/get_one/:id',
specialityController.getOneSpeciality)
router.put('/speciality/update', specialityController.updateSpeciality)
router.delete('/speciality/delete/:id',
specialityController.deleteSpeciality)

router.post('/speciality/add_group', specialityController.addGroup)
router.delete('/speciality/remove_group',
specialityController.removeGroup)

router.get('/speciality/getByDepartmentName/:id',
specialityController.getSpecialitiesInInstitute)

module.exports = router
const Router = require('express')
const router = new Router()
const subjectController = require('../controller/subjectController')

router.post('/subject/create', subjectController.createSubject)
router.get('/subject/get_all', subjectController.getAllSubjects)
router.get('/subject/get_one/:id', subjectController.getSubject)
router.delete('/subject/delete/:id', subjectController.deleteSubject)

module.exports = router

const Router = require('express')
const router = new Router()
const userController = require('../controller/userController')
const {check} = require('express-validator')

router.post('/user/create_student', userController.createStudent)
router.post('/user/create_professor', userController.createProfessor)
router.get('/user/get_all', userController.getAllUsers)
router.get('/user/get_one/:id', userController.getOneUser)
router.get('/user/get_students_by_group/:id',
userController.getStudentsByGroup)
router.put('/user/update/:id', userController.updateUser)
```

```
router.put('/user/update_role', userController.updateUserRole)
router.delete('/user/delete_student/:id', userController.deleteStudent)
router.delete('/user/delete_user/:id', userController.deleteUser)

router.post('/registration',
    check('username','Имя пользователя не может быть пустым').notEmpty(),
    check('password', 'Пароль должен иметь больше 4 и меньше 30
символов').isLength({min:4, max:30}),
    userController.registration)
router.post('/login', userController.login)

module.exports = router
```

Приложение Г

(рекомендуемое)

Результаты тестирования серверной части

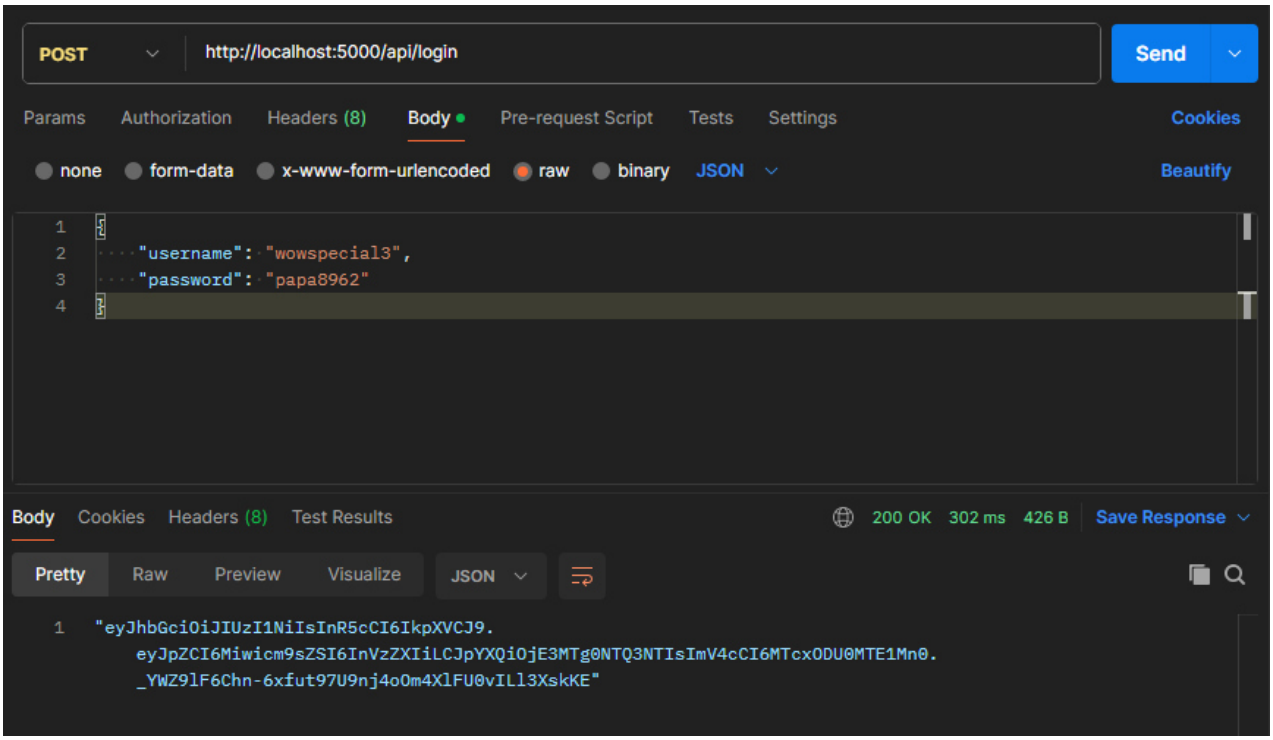


Рисунок Г.1 – Авторизация пользователя

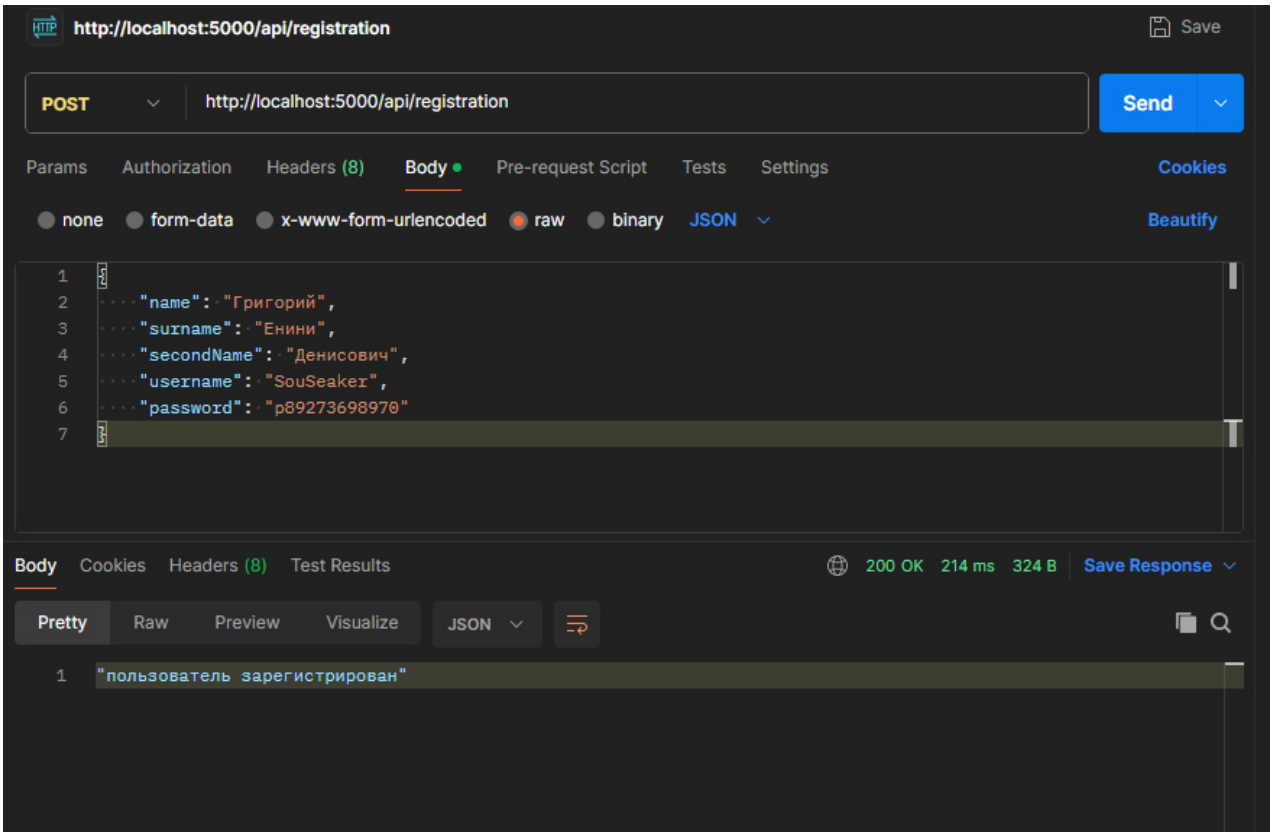


Рисунок Г.2 – Регистрация пользователя

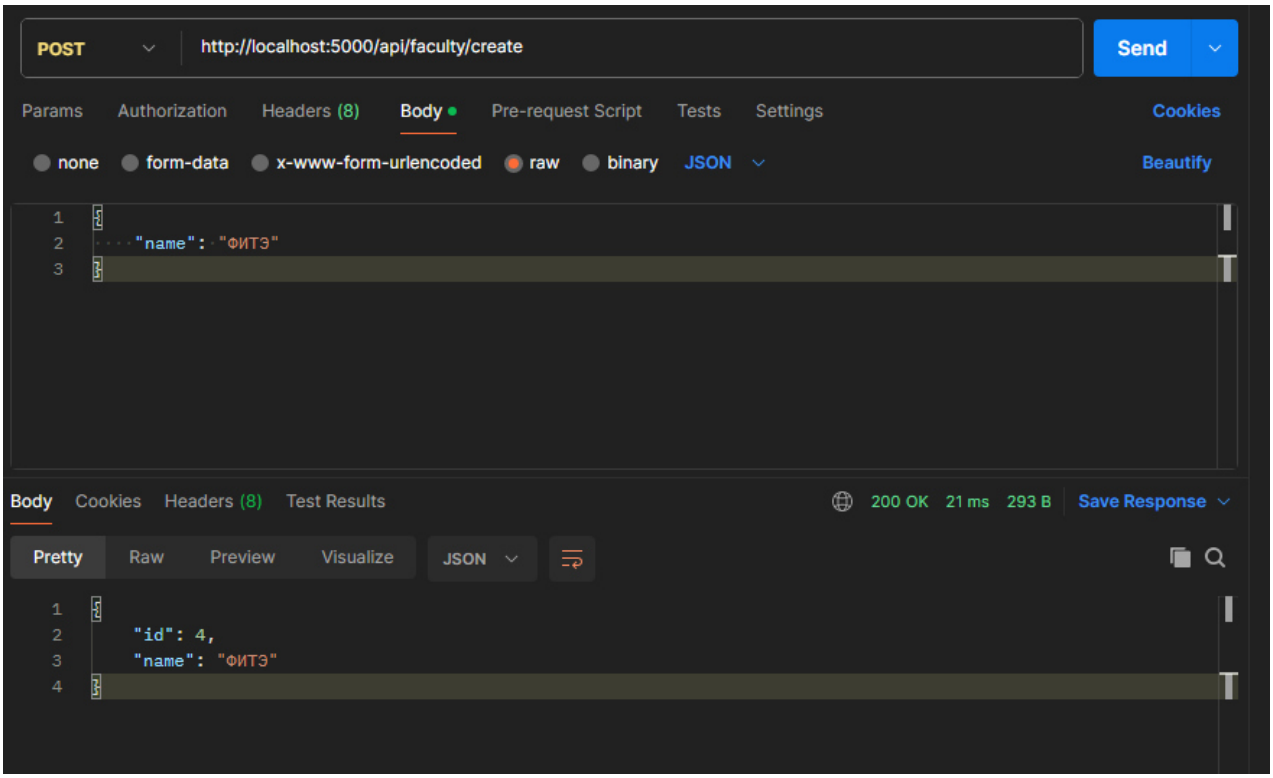


Рисунок Г.3 – Создание факультета

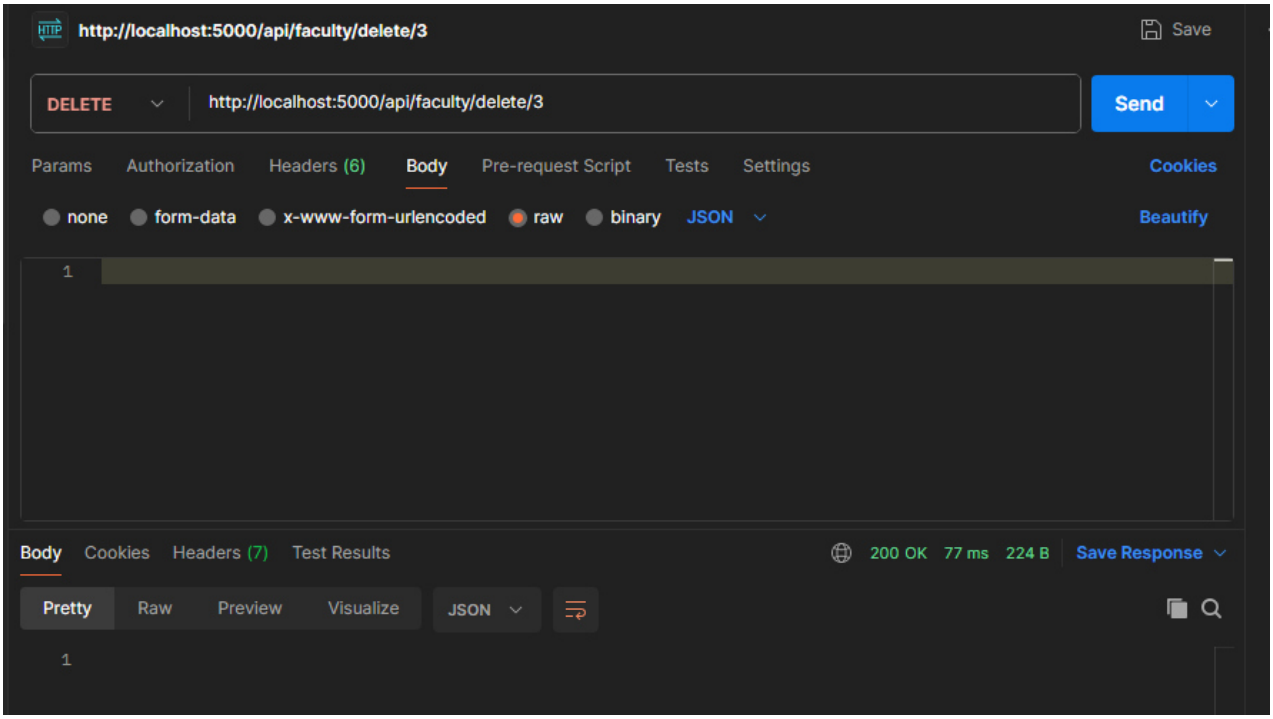


Рисунок Г.4 – Удаление факультета

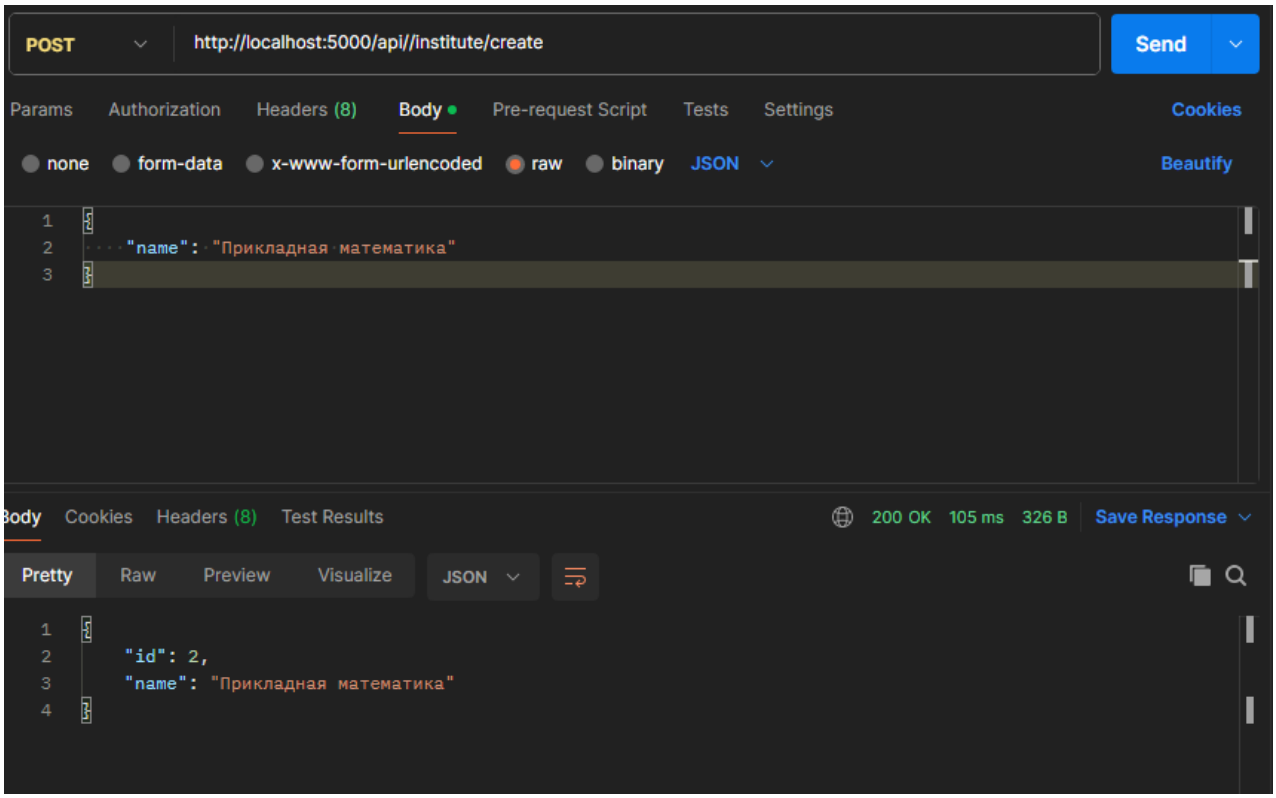


Рисунок Г.5 – Создание кафедры

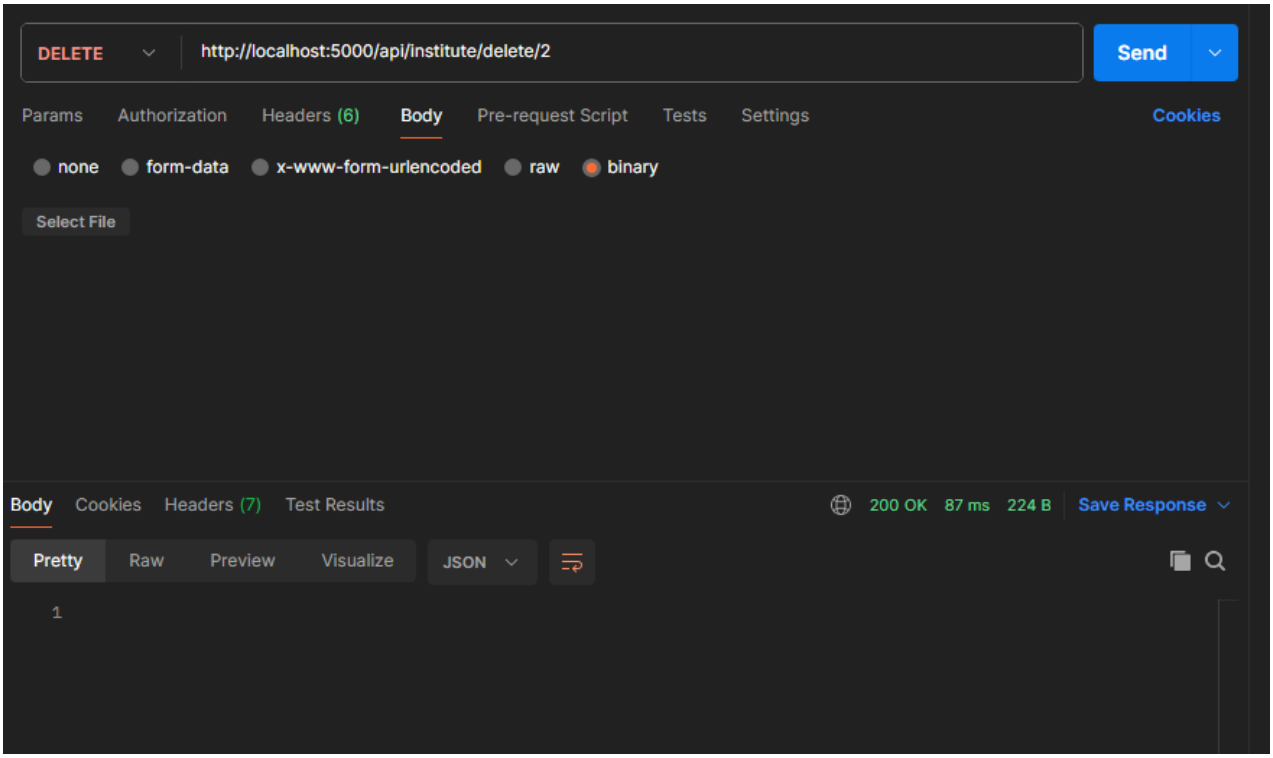


Рисунок Г.6 – Удаление кафедры

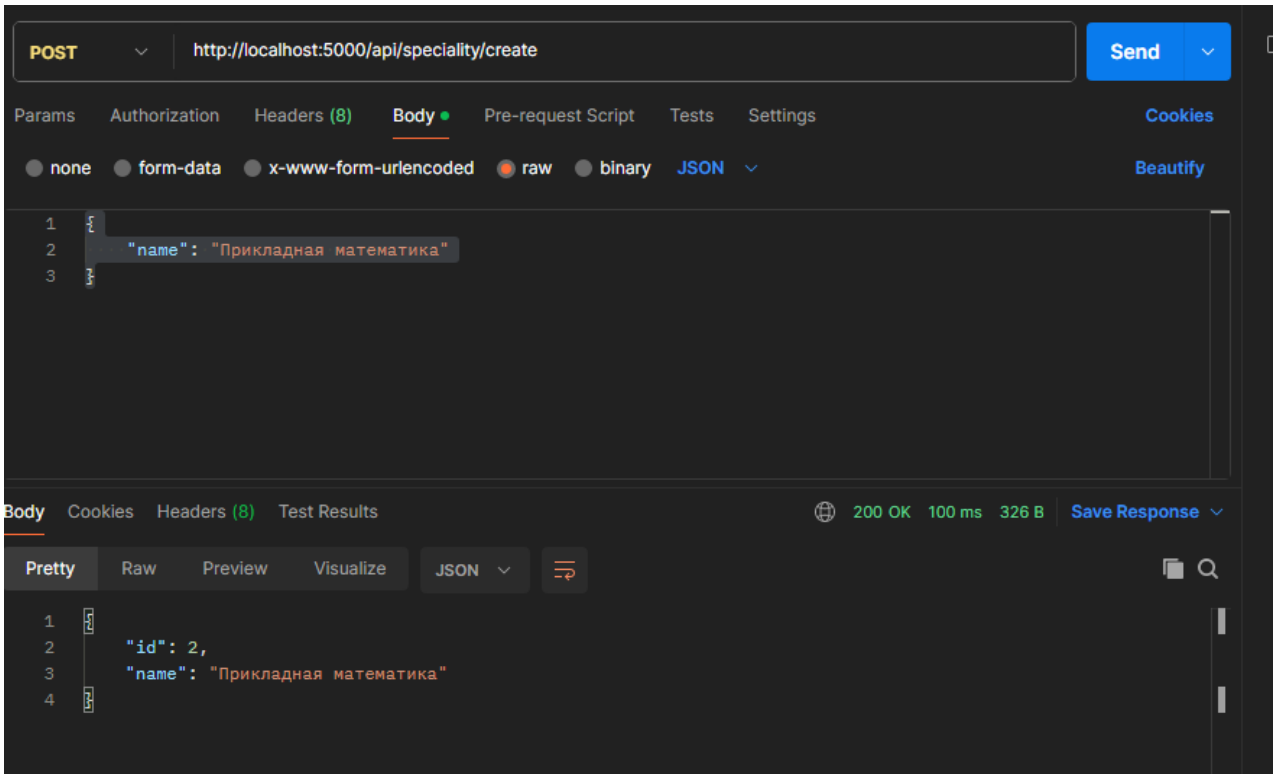


Рисунок Г.7 – Создание специальности

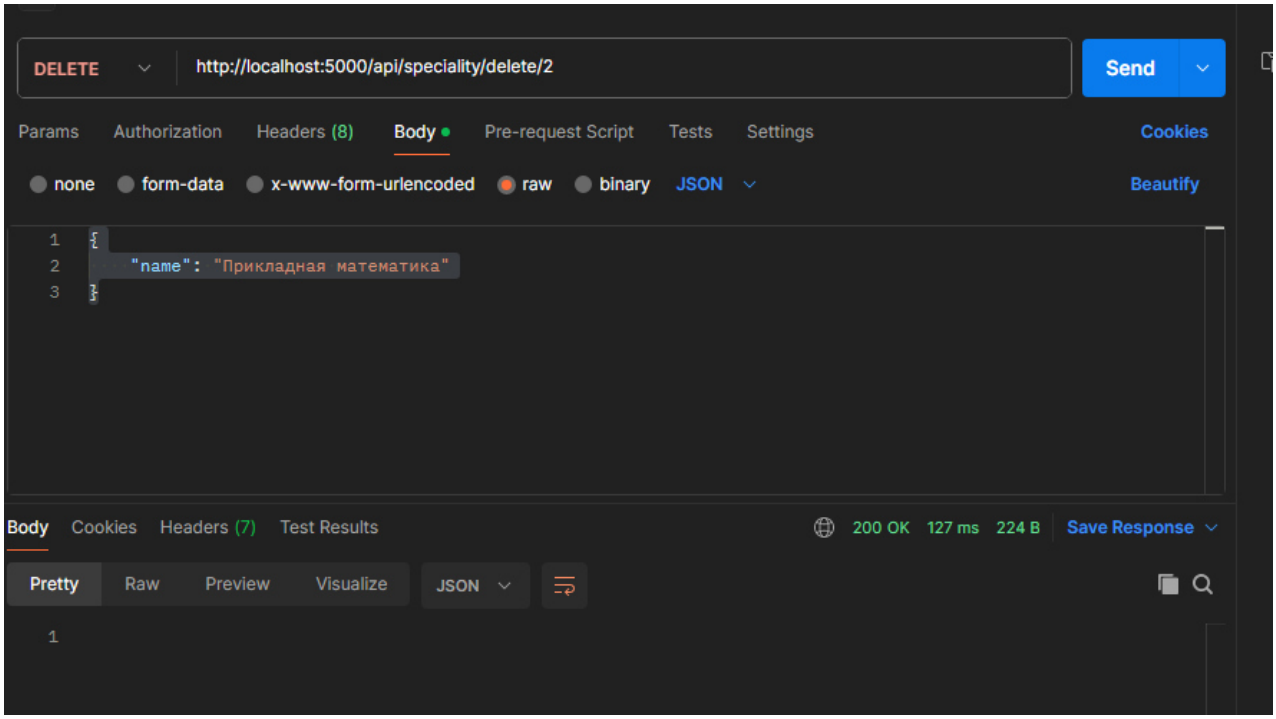


Рисунок Г.8 – Удаление специальности

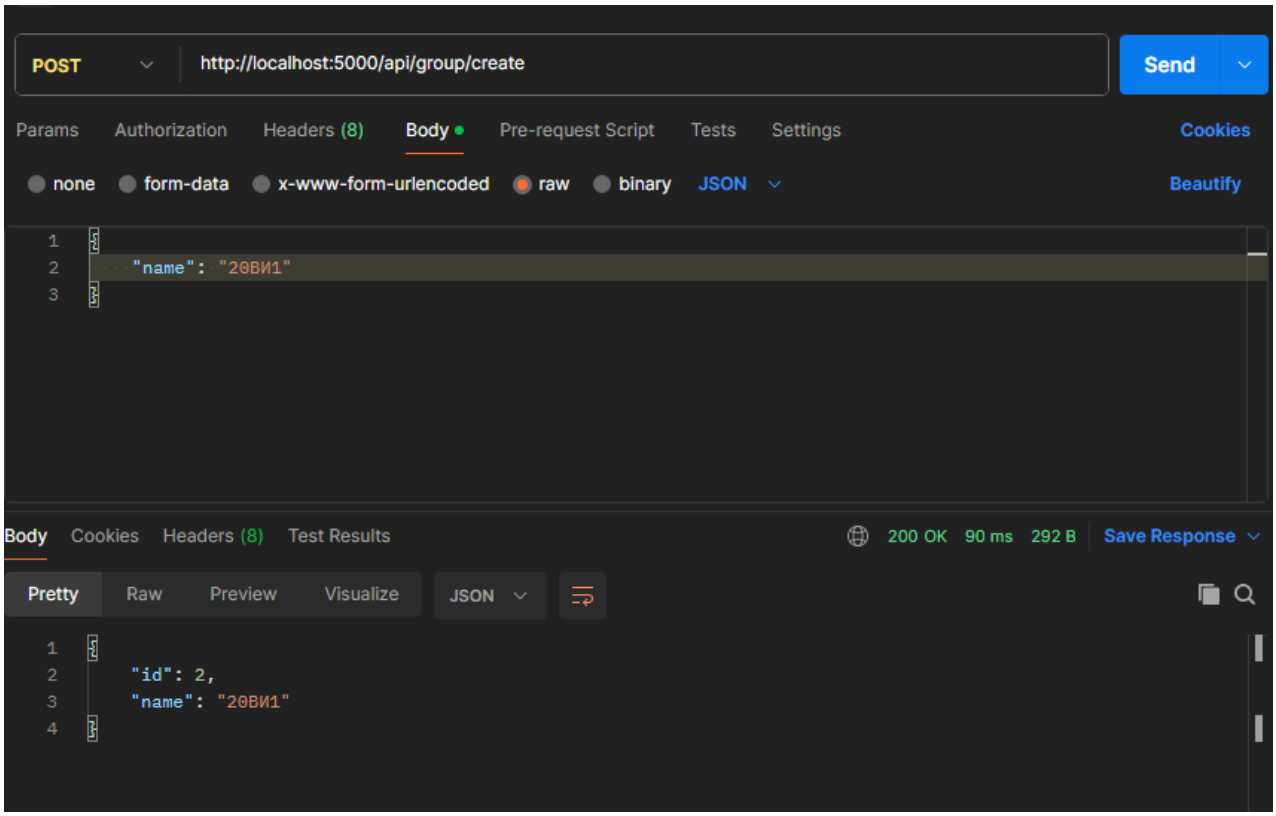


Рисунок Г.9 – Создание группы

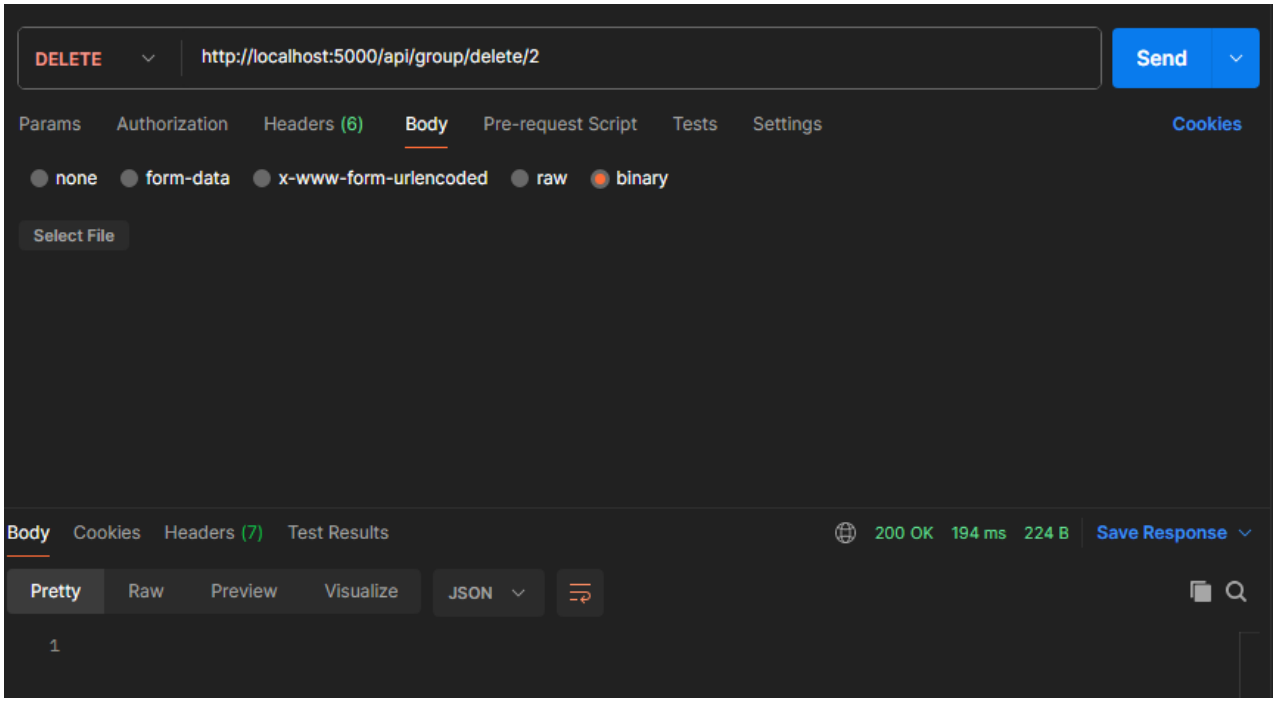


Рисунок Г.10 – Удаление группы

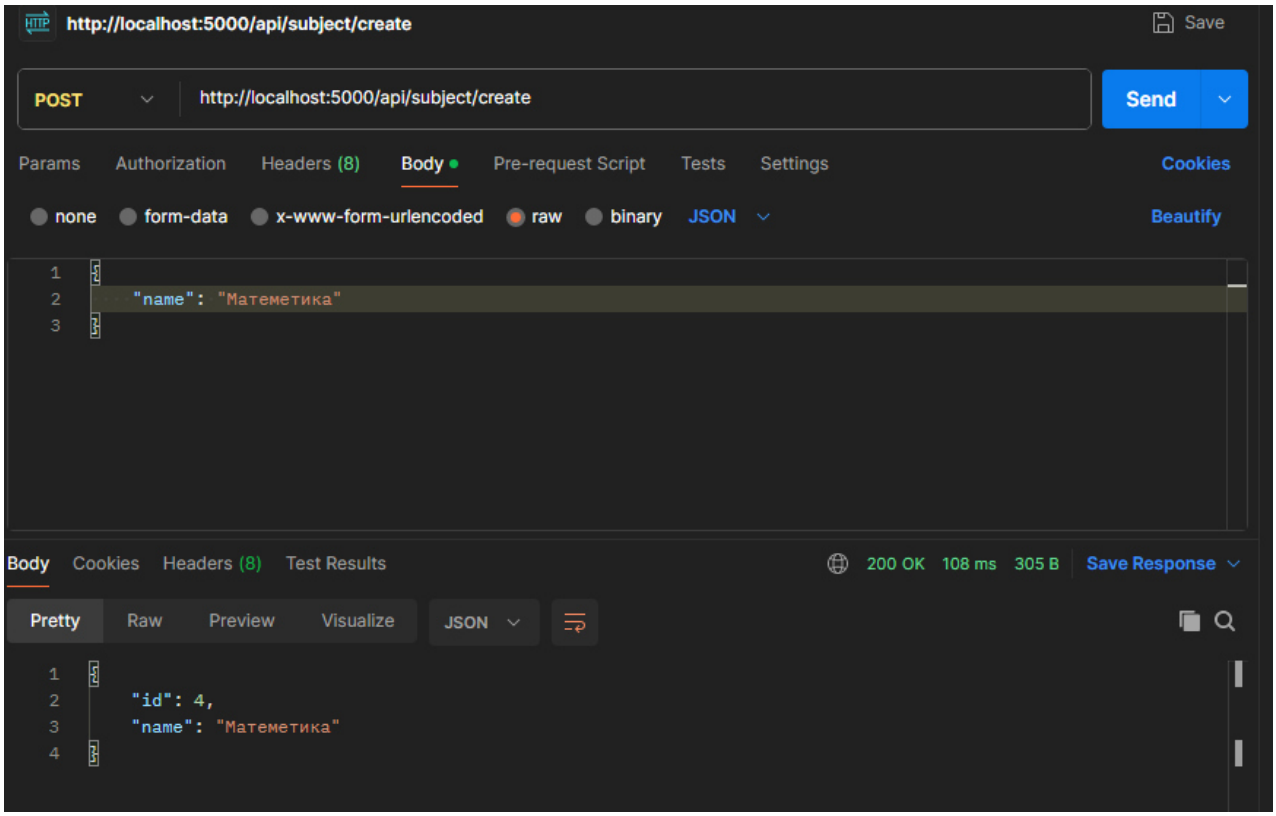


Рисунок Г.11 – Создание предмета

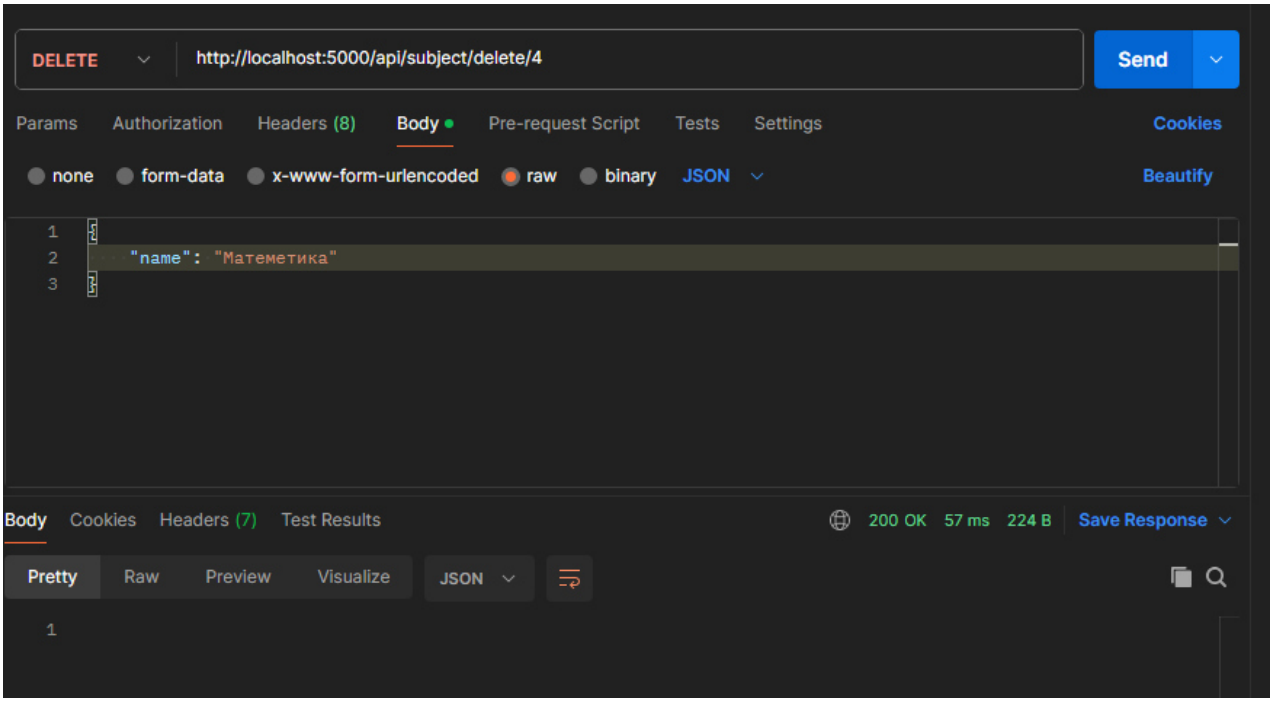


Рисунок Г.12 – Удаление предмета

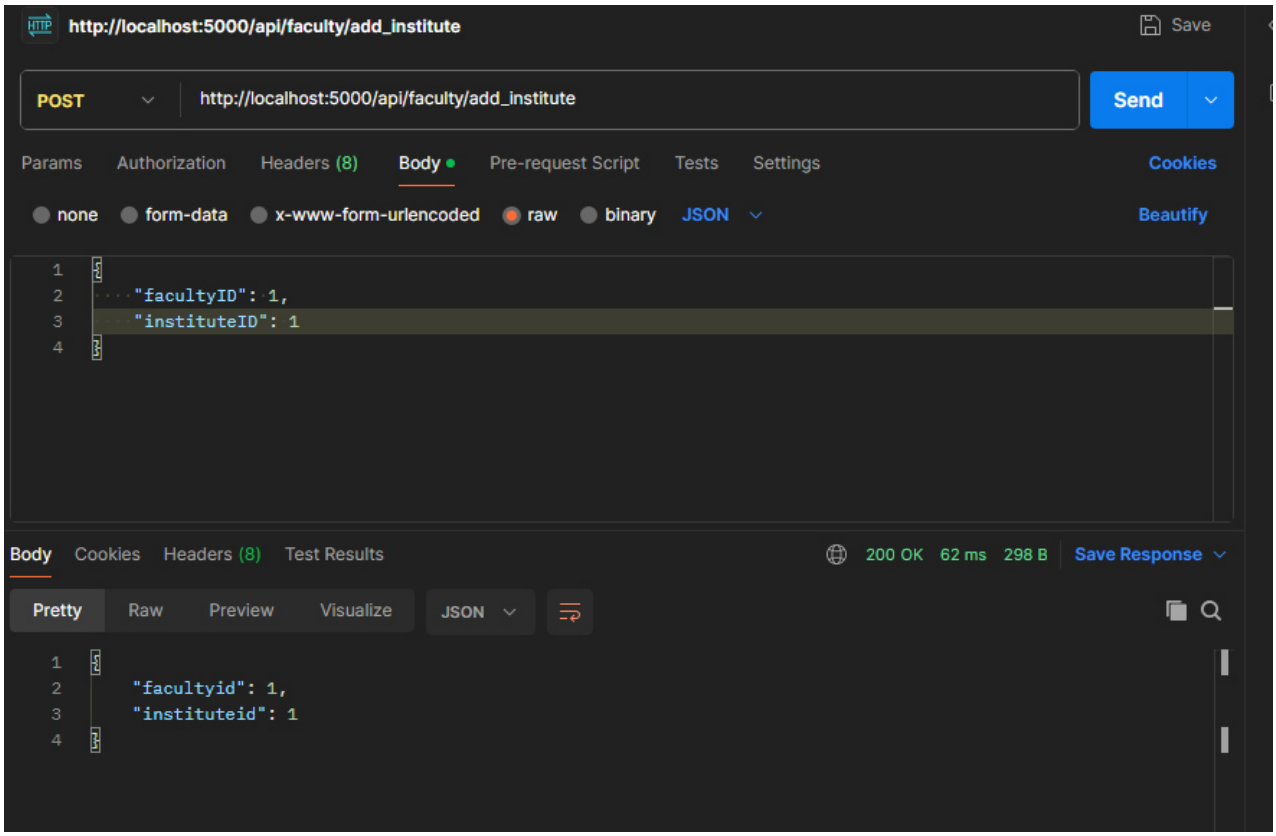


Рисунок Г.13 – Добавление кафедры в факультет

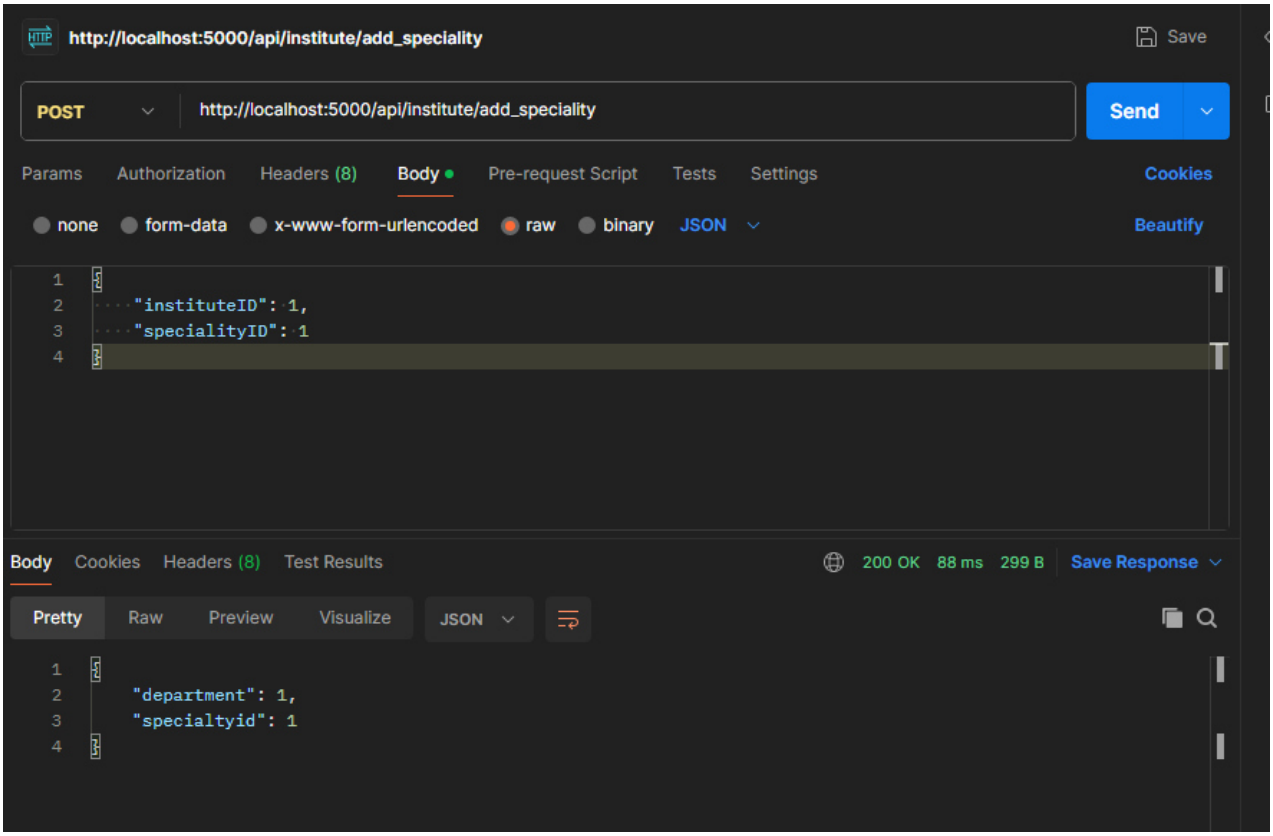


Рисунок Г.14 – Добавление специальность на кафедру

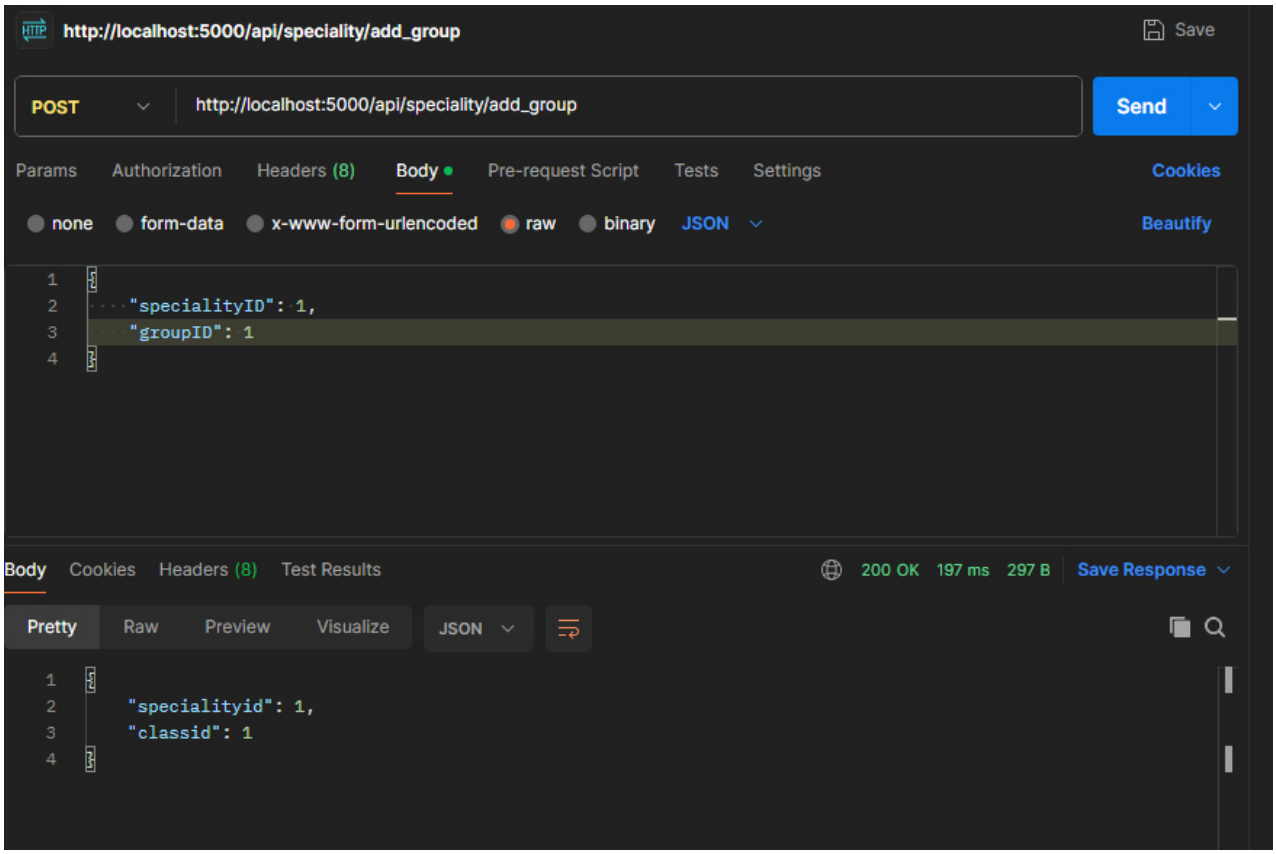


Рисунок Г.15 – Добавление группы на специальность

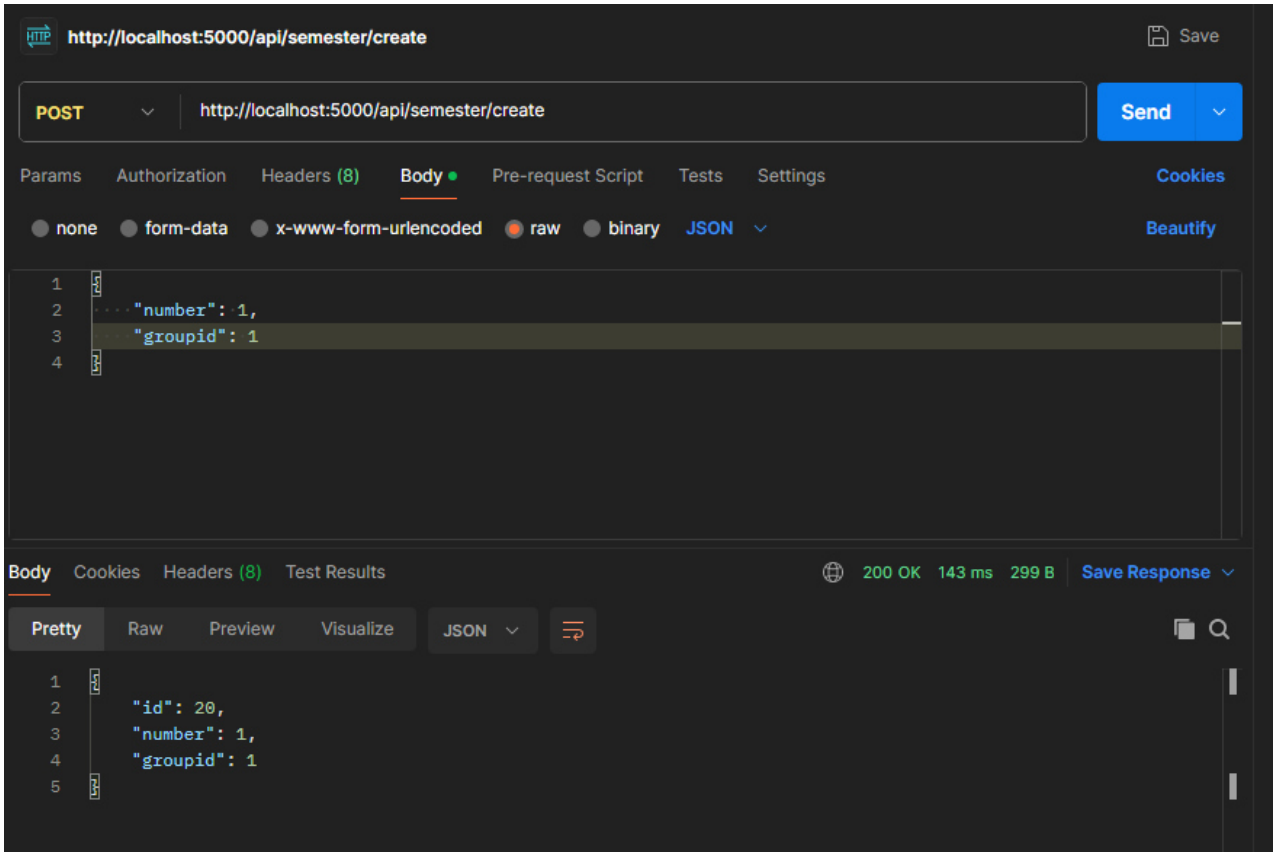


Рисунок Г.16 – Создание семестра

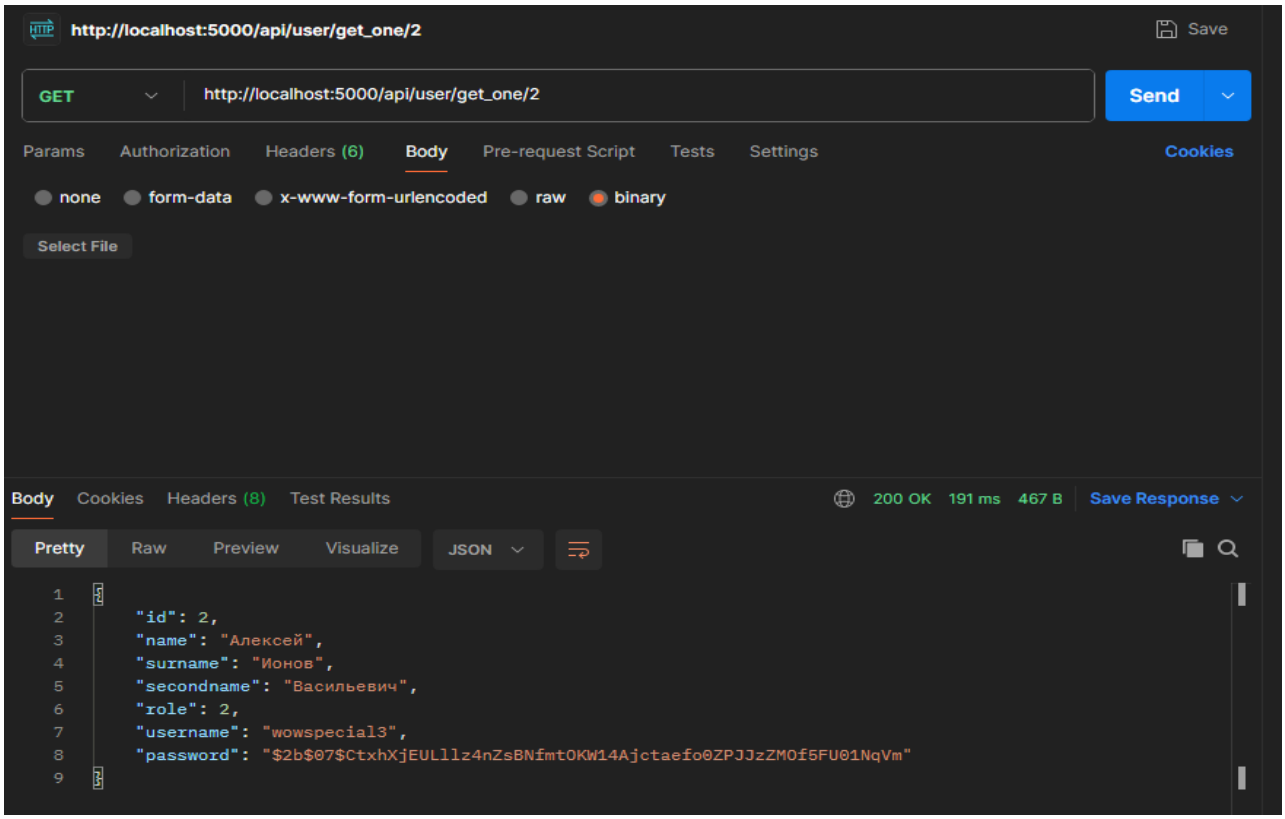


Рисунок Г.17 – Получение информации о пользователе

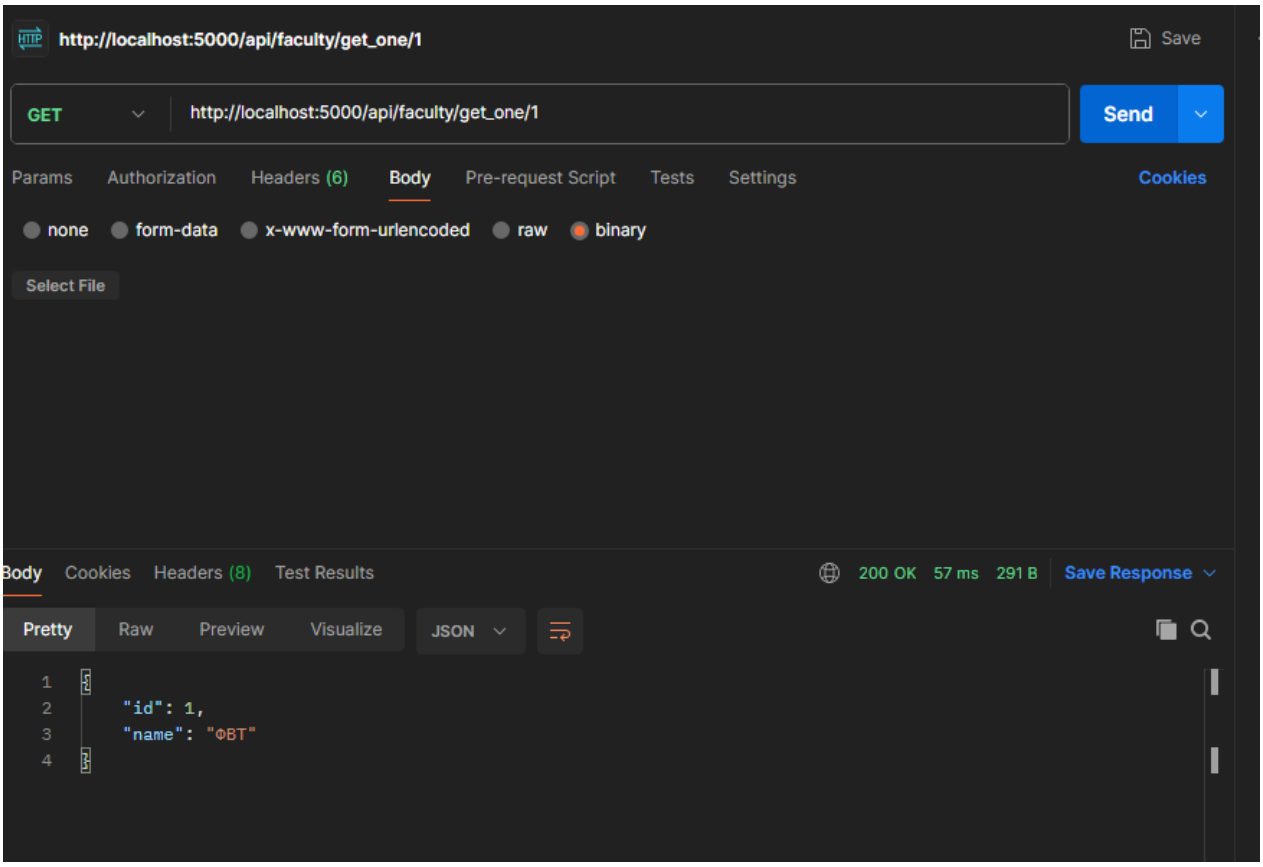


Рисунок Г.18 – Получение информации о факультете

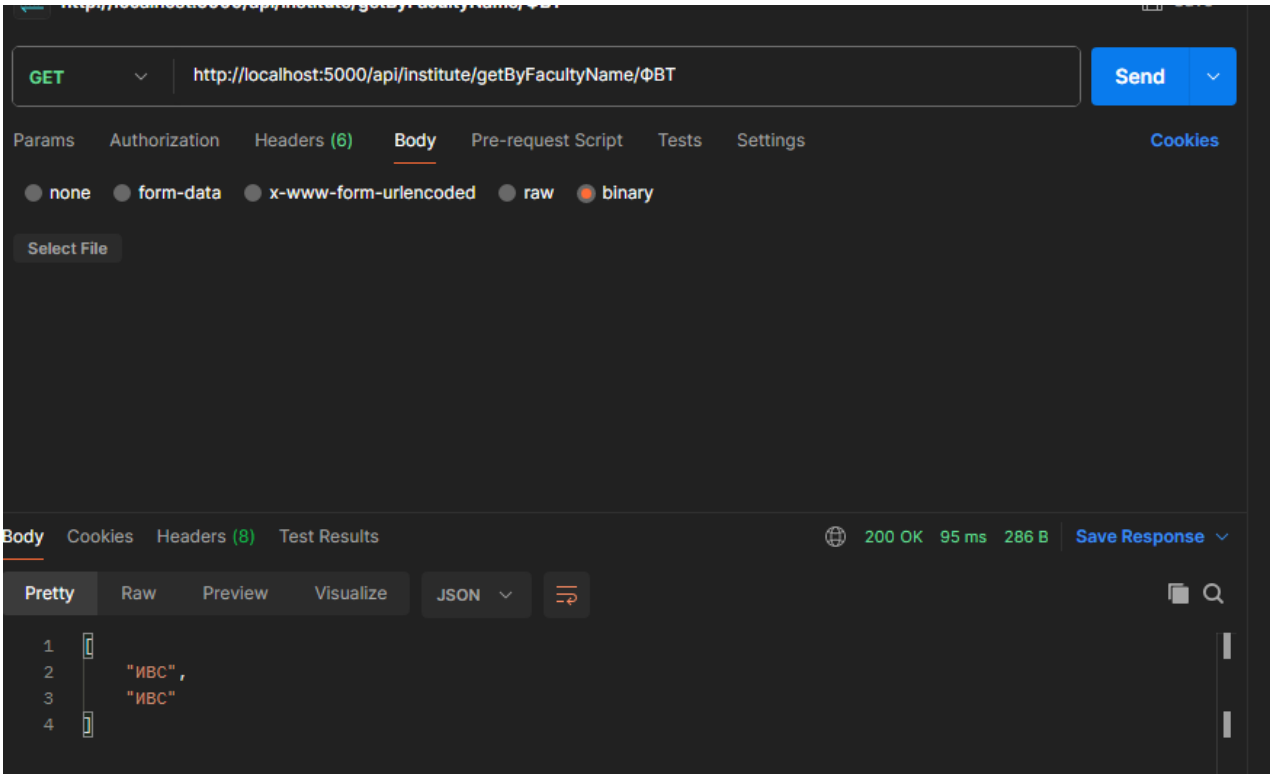


Рисунок Г.19 – Получение списка кафедр по названию факультета

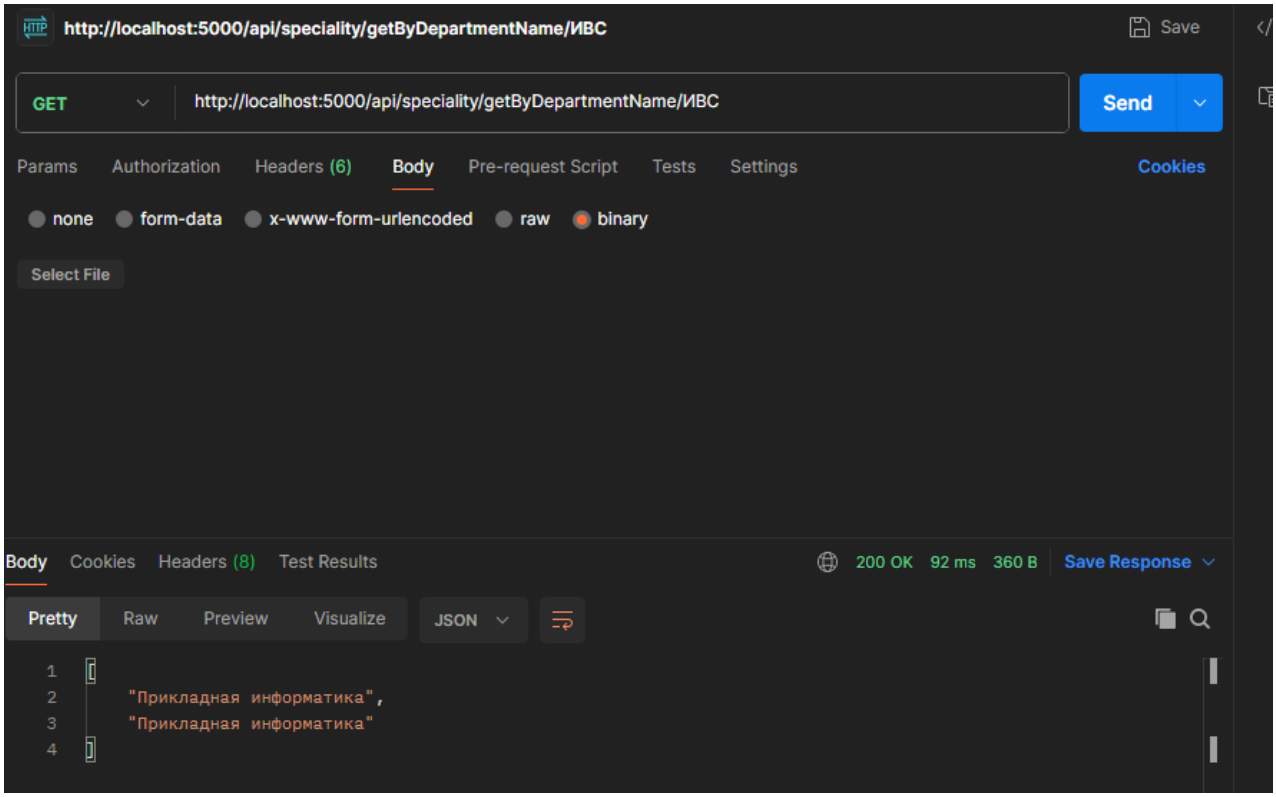


Рисунок Г.20 – Получение списка специальностей по названию кафедры

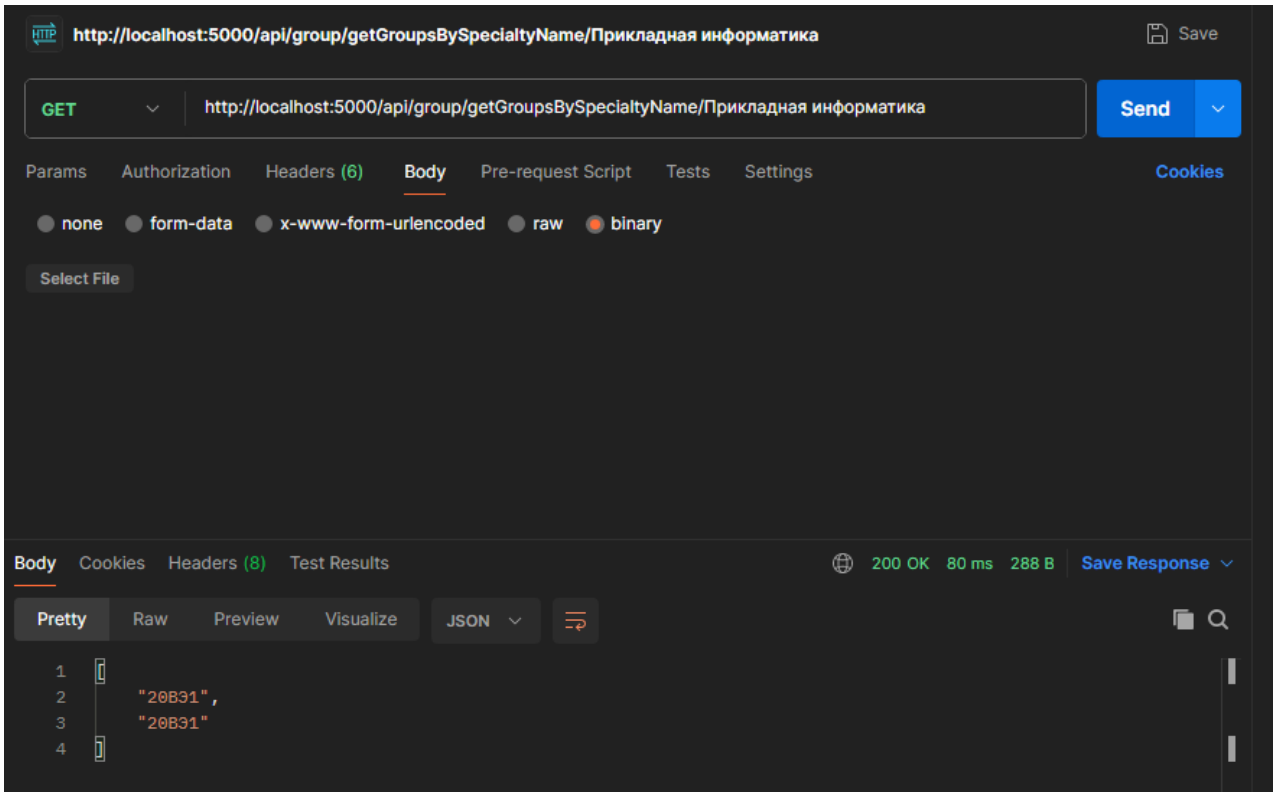


Рисунок Г.21 – Получение списка групп по названию специальности

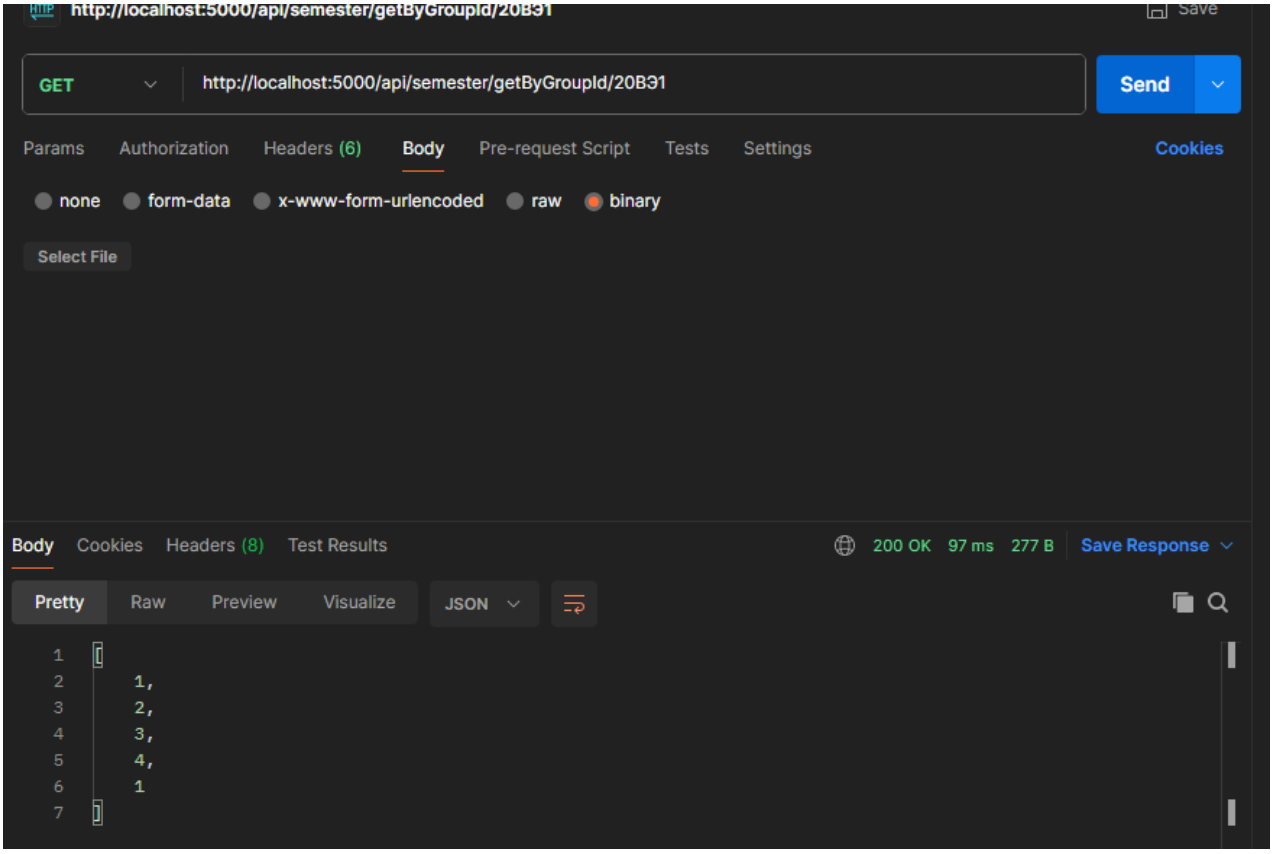


Рисунок Г.22 – Получение списка семестров группы

Приложение Д

(рекомендуемое)

Результаты тестирования серверной части

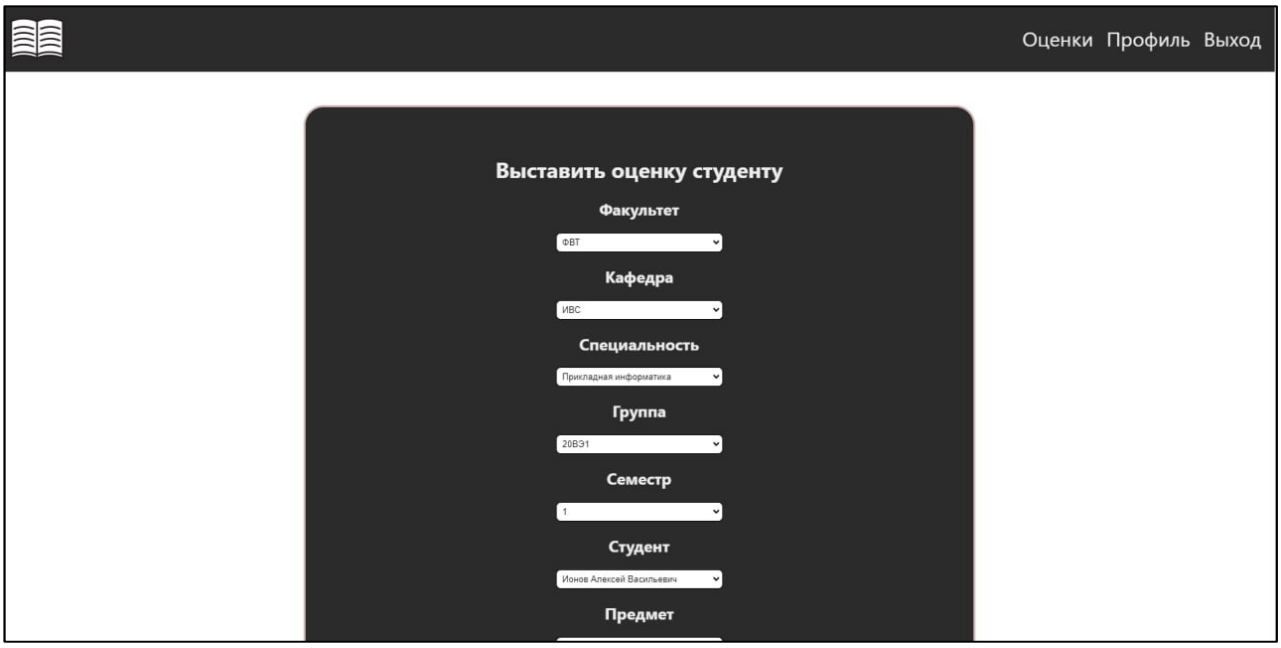


Рисунок Д.1 – Успешная авторизация

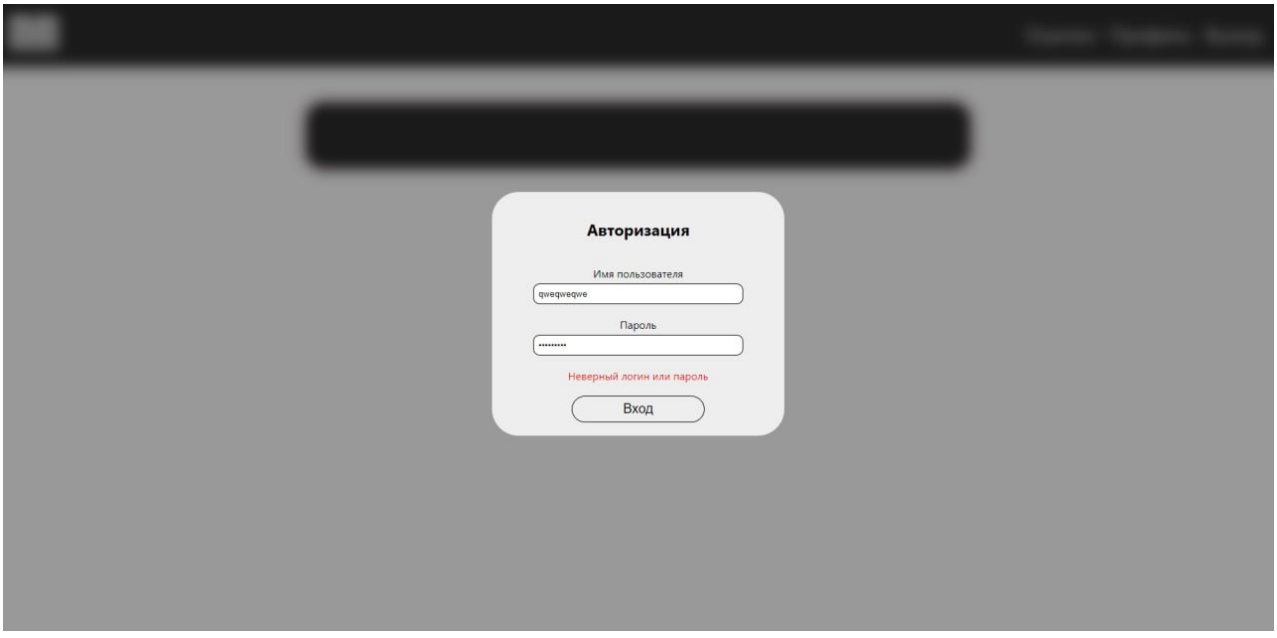


Рисунок Д.2 – Ошибка авторизации при неверно введенных данных

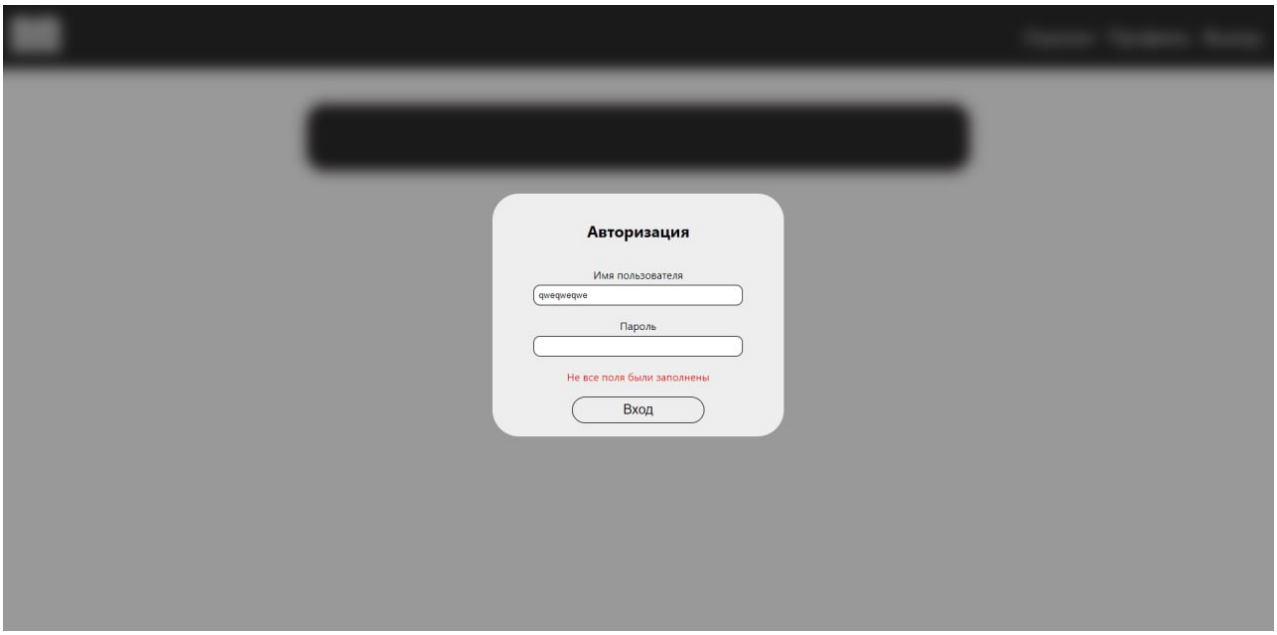


Рисунок Д.3 – Ошибка авторизации при незаполненных полях

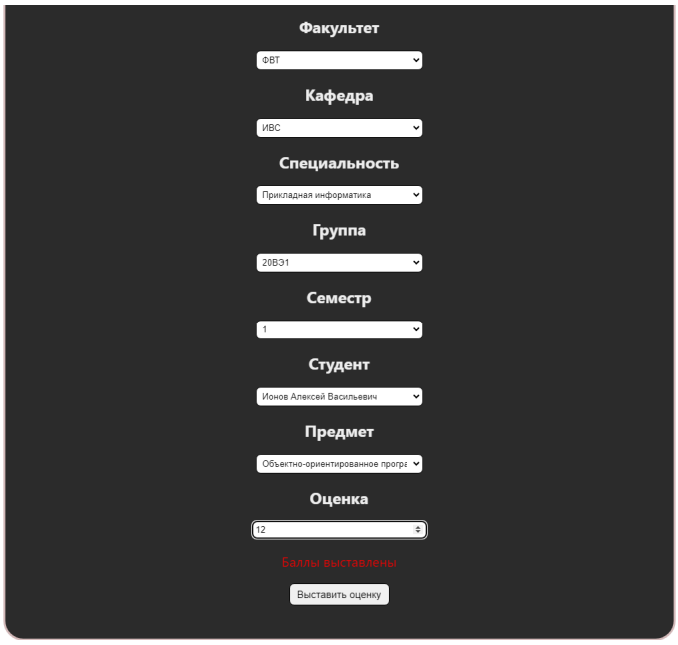


Рисунок Д.4 – Успешное выставление баллов студенту

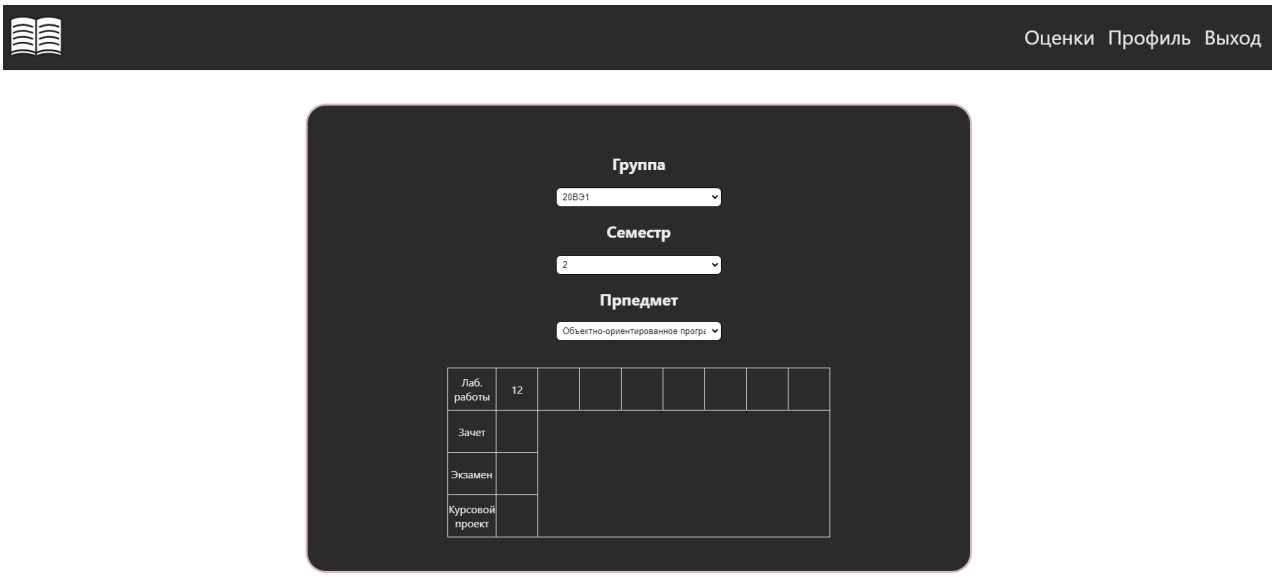


Рисунок Д.5 – Просмотр успеваемости

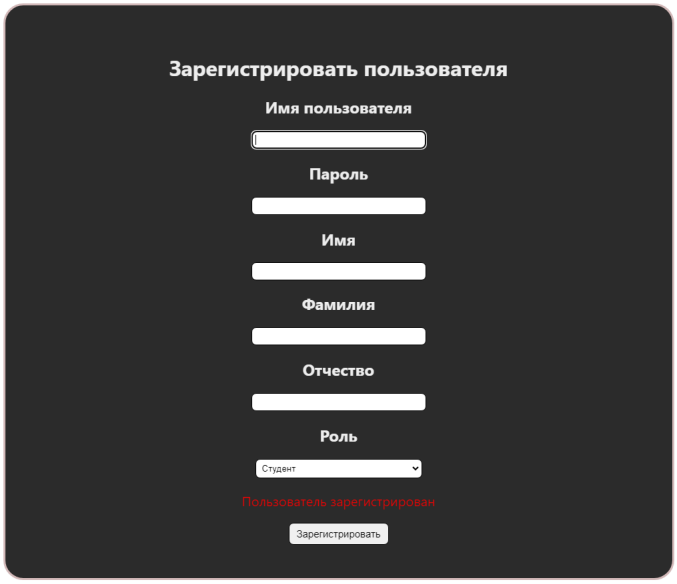


Рисунок Д.6 – Регистрация пользователя