МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 03 «Информационные системы и технологии»

Специализация «Издательско-полиграфический комплекс»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ**

по дисциплине «Программирование в Internet»

Тема: Web-приложение «МедТестЧек»

Исполнитель

|  |  |  |
| --- | --- | --- |
| cтудент 4 курса группы 1  Руководитель |  | \_\_\_\_\_\_\_\_\_\_\_\_ Д. C. Шкабров  подпись, дата |
| ассистент |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. С. Кантарович  подпись, дата |
| Курсовой проект защищен с оценкой |  | \_\_\_\_\_\_\_\_\_\_\_\_ |
| Руководитель \_\_\_\_\_\_\_\_\_\_ |  | В. С. Кантарович |

подпись

Минск 2023

Содержание

[Введение 4](#_Toc155588150)

[1 Постановка задач и обзор аналогов 5](#_Toc155588151)

[1.1 Обзор аналогов 5](#_Toc155588152)

[1.1.1 Web-приложение «Kahoot!»..............................................................................5](#_Toc155588153)

[1.1.2 Web-приложение «Quizizz»...............................................................................6](#_Toc155588154)

[1.2 Постановка задач 8](#_Toc155588155)

[1.3 Вывод........................................................................................................................9](#_Toc155588156)

[2 Проектирование web-приложения 10](#_Toc155588157)

[2.1 Выбор инструментов разработки 10](#_Toc155588158)

[2.2 Разработка модели базы данных 10](#_Toc155588159)

[2.3 Проектирование UML-диаграммы 15](#_Toc155588160)

[2.4 Проектирование алгоритмов 17](#_Toc155588161)

[2.5 Вывод......................................................................................................................21](#_Toc155588162)

[3 Разработка web-приложения 22](#_Toc155588163)

[3.1 Разработка серврной части 22](#_Toc155588164)

[3.1.1 Хэширование паролей......................................................................................32](#_Toc155588165)

[3.1.2 Валидация данных............................................................................................33](#_Toc155588166)

[3.2 Разработка клиентской части 34](#_Toc155588167)

[3.3 Развертывание приложения на базе Docker-Desktop 36](#_Toc155588167)

[3.4 Вывод......................................................................................................................37](#_Toc155588168)

[4 Тестирование 38](#_Toc155588169)

[4.1 Ручное тестирование 38](#_Toc155588170)

[4.2 Вывод......................................................................................................................44](#_Toc155588171)

[5 Руководство пользователя 45](#_Toc155588172)

[5.1 Запуск проекта 45](#_Toc155588173)

[5.2 Руководство администратора 46](#_Toc155588174)

[5.3 Руководство преподавателя 48](#_Toc155588175)

[5.4 Руководство студента 50](#_Toc155588176)

[5.4 Вывод......................................................................................................................52](#_Toc155588177)

[Заключение 53](#_Toc155588178)

[Список используемых источников 54](#_Toc155588179)

[Приложение А 55](#_Toc155588180)

[Приложение Б 60](#_Toc155588181)

[Приложение В 65](#_Toc155588182)

[Приложение Г 67](#_Toc155588183)

[Приложение Д 70](#_Toc155588184)

[Приложение Е 73](#_Toc155588185)

[Приложение Ж 76](#_Toc155588186)

# **Введение**

В современной медицине постоянно появляются новые открытия и возможности, поэтому очень важно идти в ногу со временем и для этого надо постоянно обновлять свои знания. Тесты помогают восполнить пробелы и повысить уровень знаний, в соответсвии с современенностью.

Но возникает проблема, когда похожие средства оказываются недоступными или неэфективными, поэтому возникает потребность для альтернативы, которая дает простой и доступный в использовании функционал для медиков. Поэтому для составляния тестов, важно учесть базовые знания и новые тенденции, появляющиеся в медицине. Так пользователи приложения вспомнят то, что учили и узнают о своих пробелах.

Цель этого курсового проекта заключена в создании системы закрепления знаний и их управления в web-приложении. Надо предоставить преподователям и студентам возможность проверить студентов и самих себя.

Для выполнения поставленных целей, надо выполнить задачи:

- сделать анализ web-приложений, применяющиеся для закрепления знаний;

- выявить основные достоинства и недостатки существующих аналогов;

- разработать web-приложение, которое позволяет создавать разные виды закрепления знаний и их управление по предиетам и их темам.

По итогу, разрабатывание системы будет давать пользователям возможность прверить знания студентов и свои.

# **1 Постановка задач и обзор аналогов**

## **1.1 Обзор аналогов**

### **1.1.1 Web-приложение «Kahoot!»**

Kahoot! — игровая обучающая платформа, используемая в школах и других учебных заведениях. На сайте предоставлен каталог игр — «Kahoots» — каждая из которых является викториной, содержащей вопросы с несколькими вариантами ответов. Kahoot! был основан в 2012 сообща студентами и преподавателями Норвежского университета естественных и технических наук [1].

Kahoot! был разработан для групповых занятий. Учащиеся собираются вокруг общего экрана — например, интерактивной доски, проектора, монитора или экрана, передаваемого виртуально, например, через VNC, Jitsi, Skype или Google Hangouts. Игровой процесс простой: все игроки одновременно отвечают на вопросы на своих устройствах. Вопросы выводятся ученикам на экран по одному. Участники набирают очки за каждый правильный ответ. В конце викторины на экран выводится количество очков всех участников, набранных во время ответа на её вопросы.

Главная страница сайта «Kahoot!» представлена на рисунке 1.1.

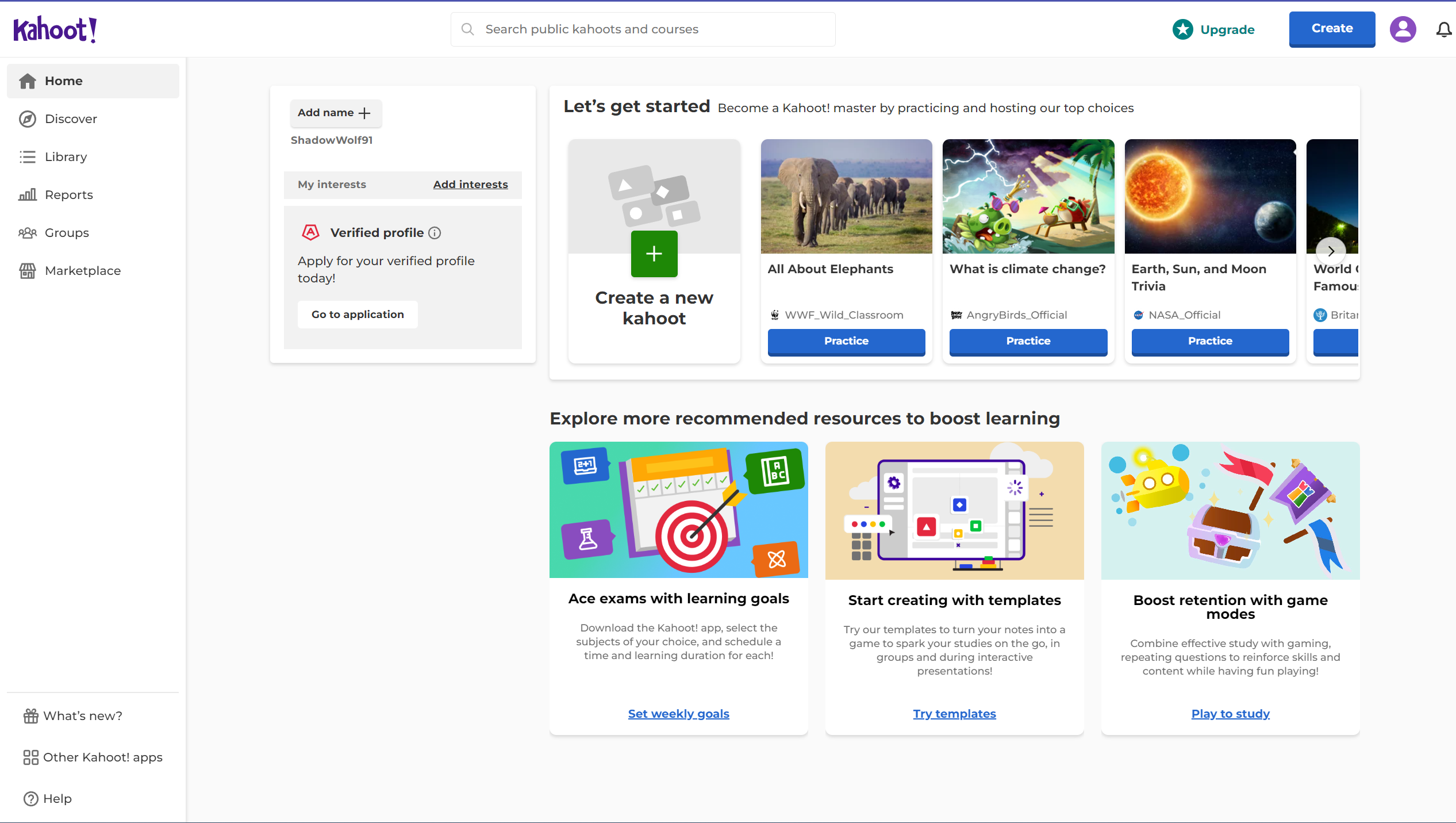


Рисунок 1.1 – Интерфейс «Kahoot!»

Страница с видом теста «Kahoot!» представлена на рисунке 1.2.



Рисунок 1.2 – Решение теста в «Kahoot!»

Для участия в игре не требуется обязательной регистрации. Викторины можно выбрать из каталога на сайте. Также можно создавать новые викторины — часто эту задачу выполняет ведущий урок, например, учитель. Время ответа на каждый вопрос ограничено примерно 30-60 секундами.

### **1.1.2 Web-приложение «Quizizz»**

Quizizz — это игровая платформа для взаимодействия со студентами, предлагающая множество функций, которые сделают занятия в классе веселыми, интерактивными и увлекательными [2].

Как учитель, вы можете проводить формативное оценивание, задавать домашние задания и осуществлять другие увлекательные взаимодействия со своими учениками (для всех классов).

Главная страница сайта «Quizizz» представлена на рисунке 1.3.

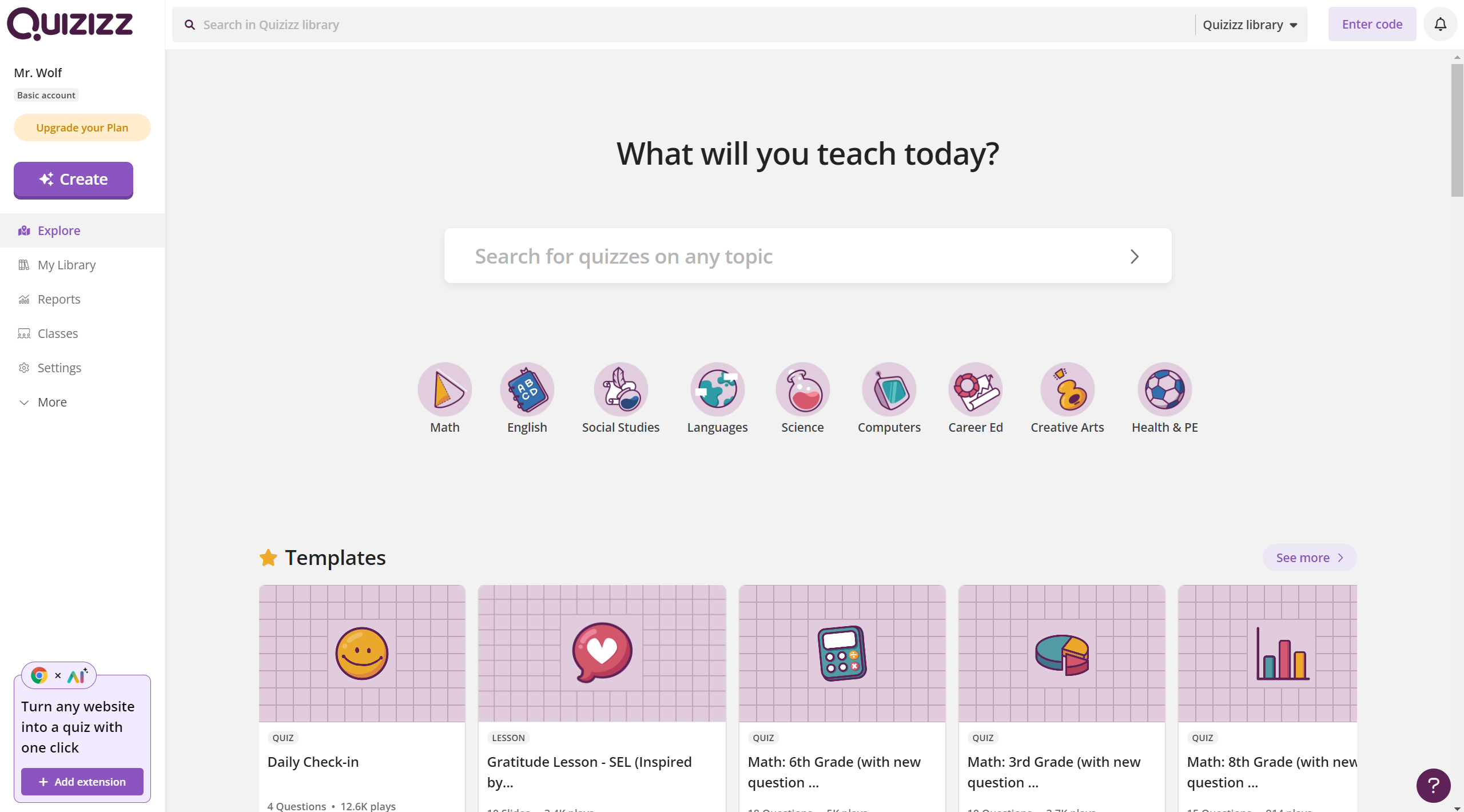


Рисунок 1.3 – Главная страница «Quizizz»

Страница сайта перед выполнением теста «Quizizz» представлена на рисунке 1.4.

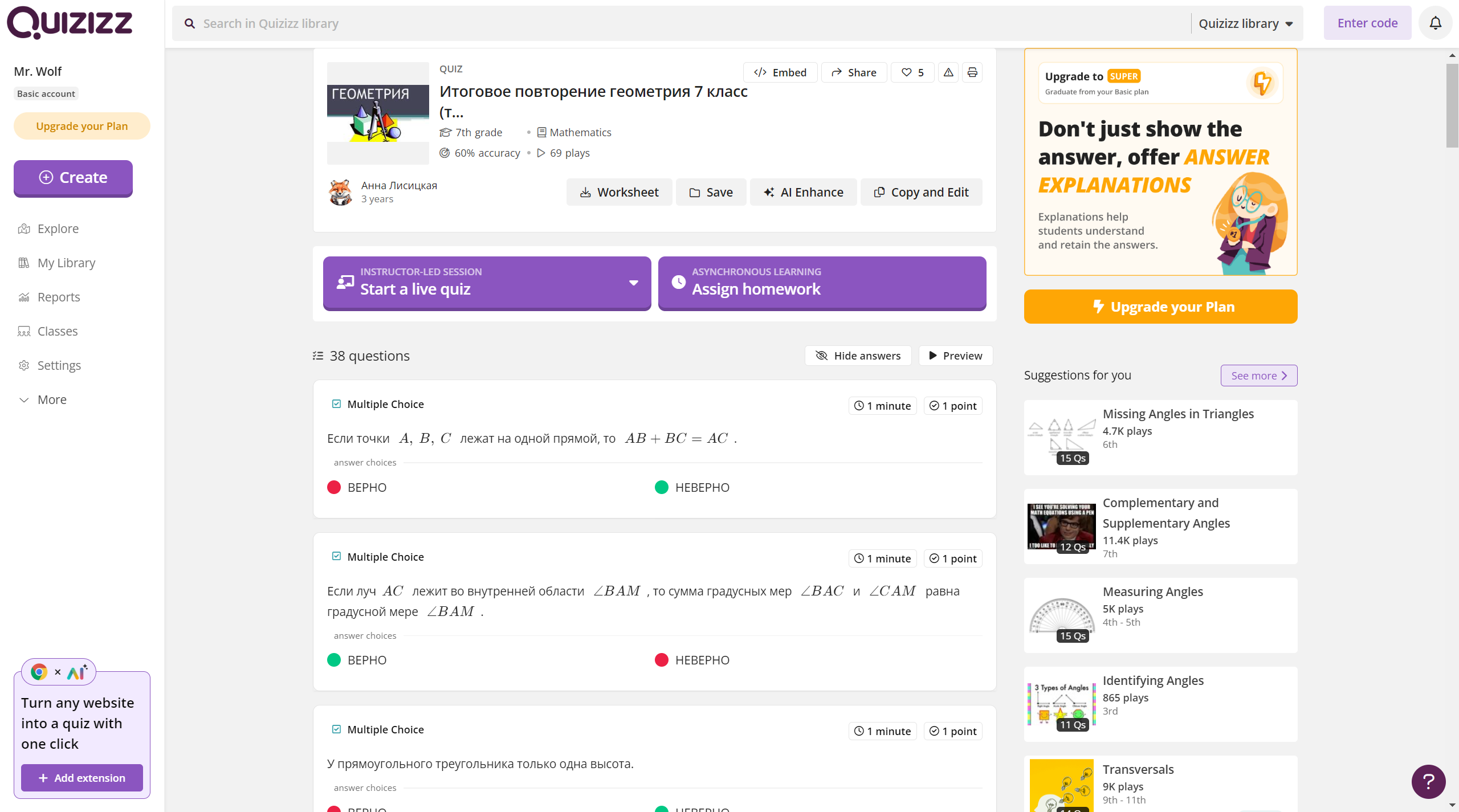


Рисунок 1.4 – Страница сайта с ответами перед выполнением теста «Quizizz»

Возможности викторины:

• Уроки/викторины, проводимые инструктором: Учителя контролируют темп; весь класс вместе просматривает каждый вопрос;

• Уроки/тесты, ориентированные на учащихся: учащиеся продвигаются в своем темпе, и вы видите таблицу лидеров и живые результаты по каждому вопросу или уроку;

• Доступ к миллионам викторин: импортируйте любые общедоступные викторины как есть, редактируйте и настраивайте их;

• (Викторина + урок) Редактор: выберите один из 5 различных типов вопросов, чтобы добавлять изображения, видео и аудио к телепортируемым вопросам из других викторин и уроков;

• Отчеты: получайте подробную информацию на уровне класса и учащихся по каждому тесту.

## **1.2 Постановка задач**

Целью данного курсового проекта является создание веб-приложения для проверки знаний в виде тестов и других вариантов. Для реализации проекта необходимо установить список задач, выполнение которых будет отображать прогресс:

* обеспечивать реализацию 3 ролей: администратора, преподавателя и студента;
* обеспечивать регистрацию и авторизацию пользователей;
* обеспечивать возможность администратору управлять пользователями и предметами;
* обеспечивать преподавателю управление тестами: добавление, удаление, редактирование тестов;
* обеспечивать преподавателю и студенту результаты тестов;
* обеспечивать студенту выбор предмета, а затем выбор вида теста;
* обеспечивать студенту возможность создания или удаления теста по темам;
* обеспечивать возможность студенту видеть таймер во время ответов на вопросы.
  1. **Вывод**

В ходе анализа существующих аналогов, была выявлена целесообразность разработки и определен основной спектр функциональных возможностей, которые необходимо реализовать при разработке данного программного продукта. Также, на основании использования приложений-аналогов были выявлены основные критерии для построения дизайна приложения, позволяющего осуществлять наиболее простое и интуитивное его использование.

# **2 Проектирование web-приложения**

После того как основной функционал был определен, можно начинать непосредственно проектирование приложения. Проектированию подлежит база данных приложения, а также UML-диаграмма и блок-схема алгоритмов работы.

**2.1 Выбор инструментов разработки**

Следующим логическим шагом следует выбор инструментов для разработки.

В качестве языка программирования для создания веб-приложения и интерфейса будет использоваться язык разработки TypeScript, также программное средство должно взаимодействовать с базой данных.

В ходе работы, вся логика, которая предусмотрена тематикой приложения, должна быть реализована на средствах, поставляемых выбранной СУБД и её стандартом языка SQL. В качестве основных инструментов, была выбрана СУБД PostgreSQL среда разработки для неё Prisma[3]. Разработка самого веб-приложения осуществлялась на платформе node.js посредством модели MRCS.

PostgreSQL — свободная объектно-реляционная система управления базами данных. Существует в реализациях для множества UNIX-подобных платформ. Функции являются блоками кода, исполняемыми на сервере, а не на клиенте БД.

Node.js представляет среду выполнения кода на JavaScript, которая построена на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код.

MRCS — cтруктура каталогов Модель-Маршруты-Контроллеры-Сервисы. Для модульной структуры кода логика должна быть разделена на эти каталоги и файлы.

* Модели (Models) — определение схемы модели;
* Маршруты (Routes) — маршруты API сопоставляются с контроллерами[4];
* Контроллеры (Controllers) - контроллеры обрабатывают всю логику проверки параметров запроса, запроса и отправки ответов с правильными кодами;
* Службы (Services) - службы содержат запросы к базе данных и возвращают объекты или выдают ошибки[5].

## **2.2 Разработка модели базы данных**

Как отмечалось в предыдущих разделах, одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя. В частности, для приложения по организации обучающих курсов, можно выделить набор таких операций, как:

* возможность администратору управлять пользователями и предметами;
* возможность преподавателю управлять тестами: добавление, удаление, редактирование тестов;
* возможность студенту выбор предмета, а затем выбор вида теста;
* возможность студенту создания или удаления теста по темам.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

В базе данных реализовано девять таблиц: «*User*», «*UserToken*», «*Statistics*», «*Subjects*», «*Themes*», «*Tests*», «*OpenQuestions*», «*Cards*», «*TOCSession*». Между таблицами также настроены связи, отражающие их взаимосвязь.

Таблица «*User*» предназначена для хранения данных после прохождения регистрации пользователем. Именно из этой таблицы берутся данные для разграничения ролей и в зависимости от этого, от возможностей, выполняемых в приложении для разных пользователей.

Таблица «*UserToken*» предназначена для хранения токенов.

Таблица «*Statistics*» предназначена для хранения данных о статистике пользователя (набранные баллы, количество правильно отвеченных)

Таблица «*Subjects*» предназначена для хранения информации об учебном предмете: название учебного предмета.

Таблица «*Themes*» предназначена для хранения информации о темах учебного предмета, как название темы, выбор режима прохождения

Таблица «*Tests*» предназначена для хранения данных о тесте и там содержиться такая информация как: вопрос, ответы (A, B, C, D), правильный ответ, какой теме они принадлежат.

Таблица «*OpenQuestions*» предназначена для хранения данных об открытом вопросе и там содержиться такая информация как: вопрос, ответ, правильный ответ, какой теме они принадлежат.

Таблица «*Cards*» предназначена для хранения данных о карточке и там содержиться такая информация как: вопрос, ответ, правильный ответ, какой теме они принадлежат.

Таблица «*TOCSession*» предназначена для хранения сессии прохождения видов закрепления знаний.

Общая структура базы данных представлена на рисунке 2.1 ниже.



Рисунок 2.1 – Структура базы данных

В качестве примера, того, на чём основывались решения построения тех или иных связей, рассмотрим связи сущностей «*Tests*»/«*OpenQuestions*»/«*Cards*» и «*Statistics*», а также «*Tests*»/«*OpenQuestions*»/«*Cards*» и «*Themes*».

«*Themes*» и «*Tests*»/«*OpenQuestions*»/«*Cards*» являются центральными понятиями в проектируемой системе, так как в совокупности они представляют собой, в некотором смысле, информационное ядро системы, отражающее её основное предназначение – управление видами закрепления знаний и создания для них определенных типов. Связь между этими сущностями определена как «один ко многим». Это объясняется тем, что один тип курсов может содержать некоторое множество конкретных курсов, например, по разным направлениям математики.

Связь между сущностями «*Statistics*» и«*Tests*»/«*OpenQuestions*»/«*Cards*» отражает факт участия в решениях тестов/вопросов/карточек студентов – один студент может как проходить несколько тестов/вопросов/карточек. Соответственно, связь между ними должна быть отражена отношением «один ко многим».

Далее будут расписана каждая таблица для общего понимания, какие данные хранятся в них. Таблица «*Users*» содержит следующие поля:

- *id\_user* – идентификационный номер пользователя;

- *username* – логин пользователя для авторизации в приложении;

- *password* – хешированный пароль пользователя;

- *role* – тип пользователя.

Таблица «*UserToken*» содержит следующие поля:

- *device\_id* – идентификационный номер устройства;

- *refreshToken* – токен пользователя;

- *user\_id* – идентификационный номер пользователя.

Таблица «*Statistics*» основная информация о статистиках пользователей:

- *id* – идентификационный номер статистики пользователя;

- *rightAnswered* – количество правильно отвеченных;

- *mark* – оценка за ответы.

Таблица «*Subjects*» содержит в себе конкретные курсы:

- *id\_subject* – идентификационный номер предмета;

- *subjectName* – название предмета.

Таблица «*Themes*» включает в себя информацию о темах предметов:

- *id\_theme* – идентификационный номер темы;

- *subject\_id* – идентификационный номер предмета;

- *themeName* – название темы;

- *mode* – режим проверки знаний;

- *questionAmount* – количество вопросов.

Таблица «*Tests*» нужна для получения тестов:

- *id\_test* – идентификационный номер теста;

- *theme\_id* – идентификационный номер темы;

- *question* – вопрос;

- *optionA* – ответ A;

- *optionB* – ответ B;

- *optionC* – ответ C;

- *optionD* – ответ D;

- *correctAnswer* – правильный ответ;

- *testName* – название теста;

- *statistic\_id* – идентификационный номер статистики пользователя.

Таблица «*OpenQuestions*» нужна для получения вопросов:

- *id\_openQuestion* – идентификационный номер руководителя;

- *theme\_id* – идентификационный номер темы;

- *question* – вопрос;

- *correctAnswer* - правильный ответ;

- *openQuestionName* - название открытого вопроса;

- *statistic\_id* - идентификационный номер статистики пользователя.

Таблицы «*Cards*» нужна для получения карточек:

- *id\_card* – идентификационный номер студента;

- *theme\_id* – идентификационный номер темы;

- *word* - слово;

- *correctAnswer* - правильный ответ;

- *cardName* – название карточки;

- *statistic\_id* - идентификационный номер статистики пользователя.

Таблица «*TOCSession*» содержит в себе данные о сессиях прохождения:

- *id* - идентификационный номер сессии;

- *userId* - идентификационный номер пользователя;

- *statisticId* - идентификационный номер предмета;

- *themeId* - идентификационный номер темы.

Остальные примеры связей между сущностями являются довольно тривиальными в сравнении с вышеуказанными, и логика их проектирования не должна вызывать вопросов, исходя только из семантического отражения любой из сущностей.

На рисунке 2.1 проиллюстрированы следующие связи:

* связь «*User-UserToken*»: один ко многим;
* связь «*User-TOCSession*»: один ко многим;
* связь « *TOCSession-Themes*»: один к одному;
* связь «*TOCSession -Cards*»: один ко многим;
* связь «*Subjects-Themes*»: один ко многим;
* связь «*Themes-Tests*»: один ко многим;
* связь «*Themes-OpenQuestions*»: один ко многим;
* связь «*Themes-Cards*»: один ко многим.

## **2.3 Проектирование UML-диаграммы**

При разработке программного обеспечения производится определение схем взаимодействия для представления, какие воздействия на систему смогут оказывать разные пользователи. Схемы взаимодействия — это описание возможного поведения пользователя и определение того, как система будет реагировать на его поведение и приспосабливаться к нему. И для этого используются UML-диаграммы.

Основная задача приложения — представлять собой единое средство, дающее возможность конечному пользователю использовать всю функциональность приложения и отслеживать поведение системы.

Данный этап является одним из самых значимых по время проектирования, так как именно тут решается, кто и какими ролями будет обладать в данной системе.

В данном приложении существует три роли: администратор, преподователь и студент. Но перед тем, как быть в трех ролях, пользователям нужно пройти авторизацию, а регистрация создает нового пользователя студента.

После регистрации, пользователь переходит в роль студента, а при авторизации пользователь переходит в роль либо студента, либо преподователя, либо администратора, которым доступны функционалы. В данной диаграмме показывается, что пользователь, который не вошел в систему должен пройти авторизацию и регистрацию, и после этого будет доступен функционал приложения, как показано на рисунке 2.2.

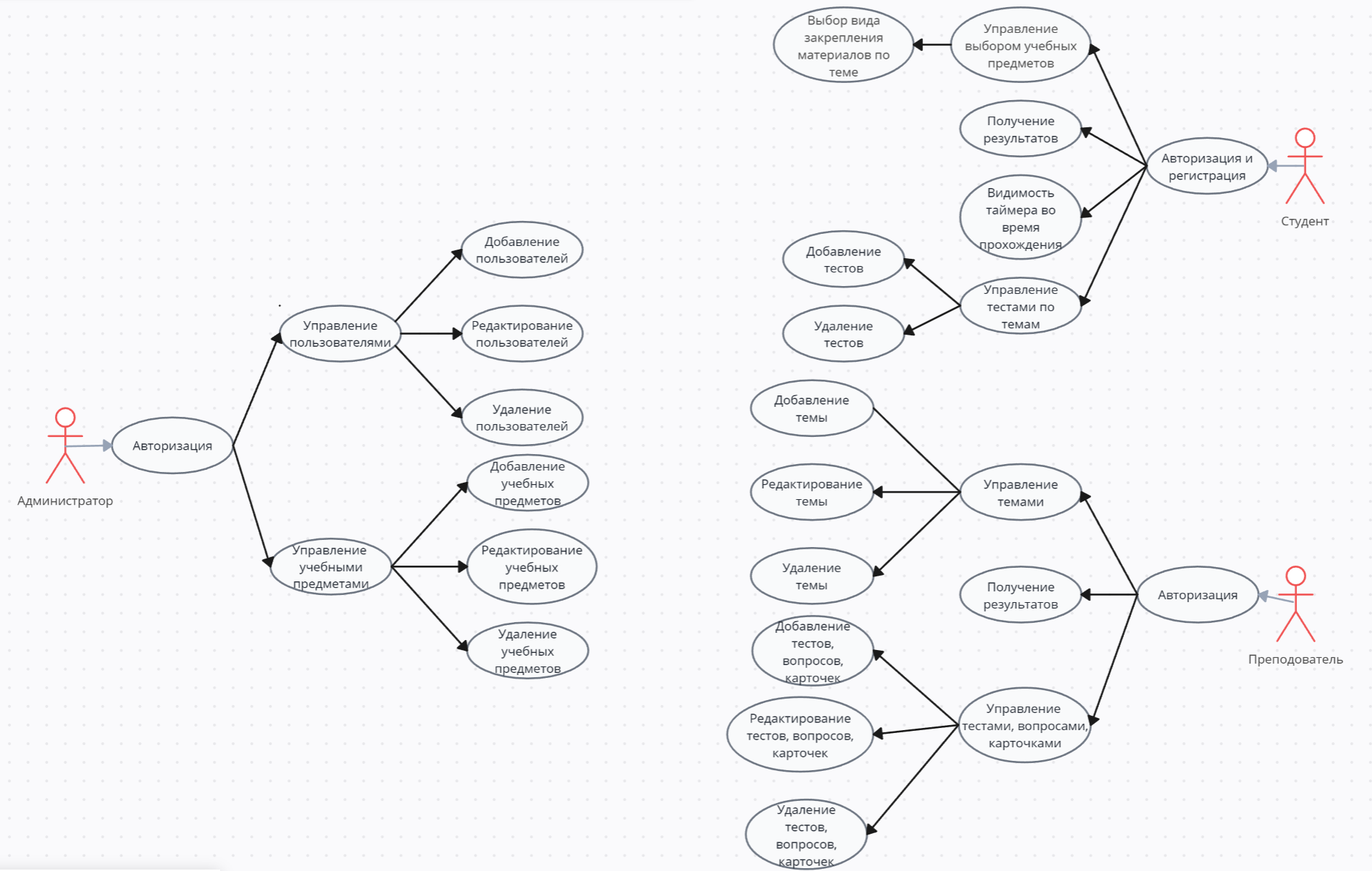


Рисунок 2.2 – UML-диаграмма всех пользователей

Для начала рассмотрим пользователя студента. Он может выполнять выбор уже из готового контента и с ним взатмодействовать.

Как видно из диаграммы преподователь может выполнять множество функций в приложении по созданию разных видов проверки знаний, так же ему доступно и редактирование данных, и их удаление.

Если пользователь в системе авторизируются под особыми правами, детали чего мы опустим, то пользователь будет в роли администратора, которому доступно еще больше функционала.

Как видно, администратор наделен наибольшими правами по сравнению с другими пользователями системы, он может создавать все предметы с темами приложения, изменять их и в случае, если необходимо, то и удалять.

Каждая роль имеет своих участников, которые должны мочь выполнять только то, что предусмотрено для них системой, любое нарушение пределов данных и оговоренных в этом приложении ролей может привести к непредвиденным последствиям, что означает важность правильной реализации данный ролей. Реализация, таким образом, должна предусматривать, что каждый пользователь может совершать определенные действия, согласно его роли. Если пользователь пытается сделать что-то, что находиться вне его полномочий, система должна пресечь его действия, иначе они пагубно отразятся на системе.

## **2.4 Проектирование алгоритмов**

Важным этапом при взаимодействии с системой является регистрация пользователя, так как далее происходит процесс авторизации, который позволит использовать весь функционал приложения. Поэтому, на мой взгляд, важным этапом будет являться проверка правильности ввода данных пользователя при регистрации, таких как проверка на неповторимость логина, проверка правильно ли пользователь ввел пароль с первого раза. Блок-схема данного алгоритма изображена на рисунке 2.3.

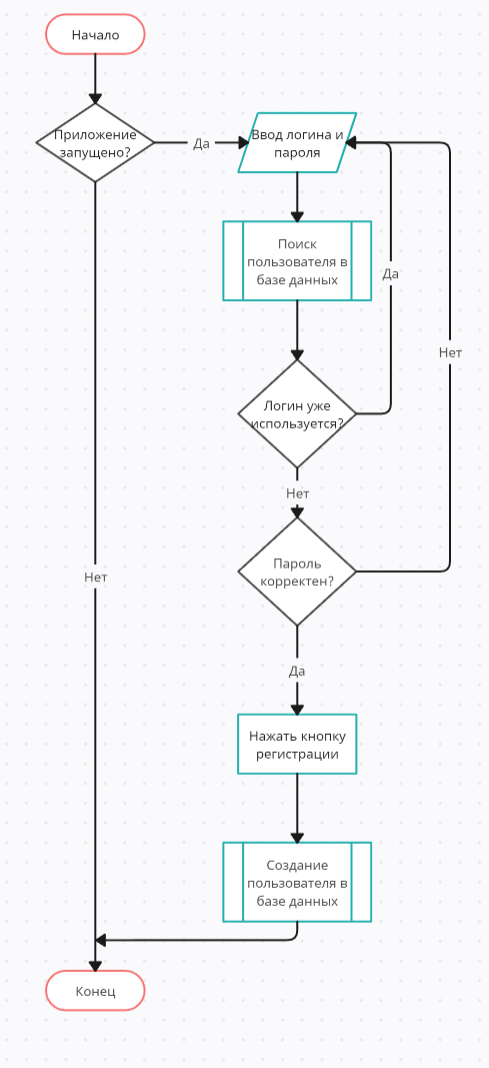


Рисунок 2.3 – Блок-схема алгоритма регистрации пользователя

Данный алгоритм гарантирует, что в приложении не будет двух пользователей с одинаковыми именами и что при повторном вводе пароля, пользователь не будет сомневаться, что он правильно ввел данные, которые будут сохранены в базе данных.

Теперь мы рассмотрим алгоритм создания теста. Для начала мы авторизуемся как преподователь, и после успешной авторизации будет на панели меню кнопка тесты, на которую мы нажмем и она переведет нас на страницу с тестами. Далее заполняем данными форму для создания тестов и сохраняем это.

Такое же происходит при создании вопросов и карточек и из-за этого алгоритм является одним из основных , он представлен на рисунке 2.4.



Рисунок 2.4 – Блок-схема алгоритма создания теста

## **2.5 Вывод**

Таким образом в конце данной главы была спроектирована база данных, описаны UML-диаграммы вариантов использования для разных типов пользователей, как для администратора, так и для преподователя и студента, был описан алгоритм регистрации пользователей с хорошей проверкой на корректность введенных данных, а также определен стек технологий, используемый в данном проекте.

Таким образом, заранее известен список всех используемых технологий, сферы деятельности, которые они покрывают и некоторые основные алгоритмы, вокруг и на основе которых потом будут появляться новые механизмы взаимодействия между частями системы.

1. **Разработка web-приложения**

Приложение будет разрабатываться посредством языка TypeScrtpt используя для реализации кроссплатформенную технологию node.js и модель MRCS.

MRCS — шаблон проектирования, с помощью которого его компоненты (модель данных приложения, маршруты приложения, службы приложения и взаимодействие с пользователем) разделены на четыре отдельных так, что модификация одного из них оказывает минимальное воздействие на остальные.

## **3.1 Разработка серврной части**

Архитектура программного обеспечения — совокупность важнейших решений об организации программной системы. Архитектура включает: выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов; соединение выбранных элементов структуры и поведения, во всё более крупные системы; архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Структура проекта состоит из папок «*dist*», «*node\_modules*», «*prisma*», «*src*», а также файлов «*.env*», «*package.json*», «*.gitignore*», «*package-lock.json*» и «*tsconfig.json*»,.

Структура проекта изображена на рисунке 3.1.

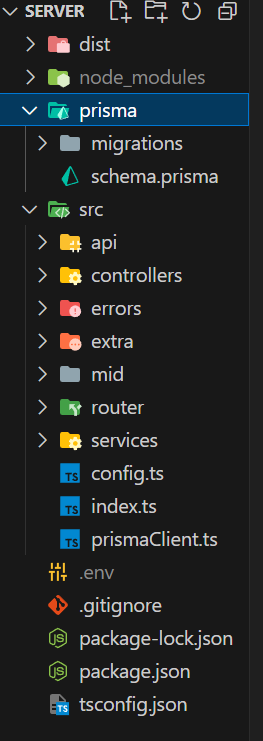


Рисунок 3.1 – Структура проекта

Папка «*prisma*» предназначена для моделирования данных. Каждая из этих моделей описывает таблицу в соответствующей базе данных и служит основой для сгенерированного доступа к данным с API, предоставляемого Prisma Client. В Schema.prisma описывается модель, а в папку migrations мигрирует эта модель в sql.

Папка «*src*» содержит в себе сам код сервера, где описана его логика и структура. В папке содержатся другие папки, как: «*api*», «*controllers», «errors*», «*extra*», «*mid*», *«router»* и *«service»,* а также файлов «*config.ts*», «*index.ts*» и «*.prismaClient.ts*».

В папке *«api»* описывается интерфейсы для сервера. А именно запросы и ответы такие как: создание, изменение, удаление, получение данных.

На рисунке 3.2 показано распределение интерфейcов, а таперь рассмотрим в пример интерфейс для сождания пользователя:

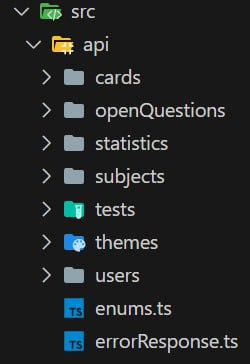


Рисунок 3.2 – Структура папки api

import { Roles } from "../../enums";

export interface ICreateUserRequest {

username: string;

password: string;

refreshToken?: string;

role?: keyof typeof Roles;

}

export interface ICreateUserResponse {

id\_user: number;

username: string;

password: string;

role: keyof typeof Roles;

refreshToken: string;

}

Листинг 3.1 – Структура интерфейса для создания пользователя

Папка *«сontrollers»* отвечает за обработку запросов от клиента и работу с базой данных. В ней находятся контроллеры для работы с каждой сущностью. Одним из условий проекта было построения приложения, который работает в стиле REST.

REST из себя представляет архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

REST-архитектура предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

- get;

- post;

- patch;

- delete.

На рисунке 3.3 показана структкра папки *«controllers»*, а пример одного из контроллеров проекта, представлен в листинге 3.2, где показаны основные методы данного контроллера. Полный код данного контроллера можно изучить в приложении А.

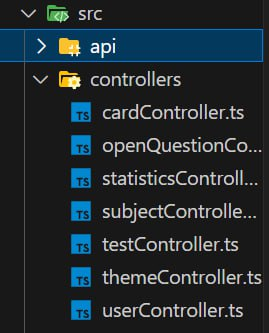


Рисунок 3.3 – Структура папки controller

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateCardRequest,

ICreateCardResponse,

} from "../api/cards/reg/createCard";

import {

IDeleteCardRequest,

IDeleteCardResponse,

} from "../api/cards/reg/deleteCard";

import {

IUpdateCardRequest,

IUpdateCardResponse,

} from "../api/cards/reg/updateCard";

import {

IGetAllCardsRequest,

IGetAllCardsResponse,

} from "../api/cards/reg/getAllCards";

import {

IGetCardByIdRequest,

IGetCardByIdResponse,

} from "../api/cards/reg/getCardById";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import CardService from "../services/cardService";

import UserRequestError from "../errors/userRequestError";

export default class CardController {

//get

static getAllCards: RequestHandler<

undefined,

IGetAllCardsResponse | IErrorResponse,

undefined,

IGetAllCardsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.getAllCards(req.query);

res.json({

cardsData: result,

cursor: result[result.length - 1]?.id\_card || null,

});

} catch (e) {

return next(e);

}

};

static getCardById: RequestHandler<

undefined,

IGetCardByIdResponse | IErrorResponse,

undefined,

IGetCardByIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.getCardById(req.query);

if (!result)

return next(

UserRequestError.NotFound(

CARD WITH ID ${req.query.id\_card} NOT FOUND

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

//create

static createCard: RequestHandler<

undefined,

ICreateCardResponse | IErrorResponse,

ICreateCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.createCard(req.body);

res.status(201).json(result);

} catch (e) {

return next(e);

}

};

//update

static updateCardData: RequestHandler<

undefined,

IUpdateCardResponse | IErrorResponse,

IUpdateCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.updateCardData(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

//delete

static deleteCard: RequestHandler<

undefined,

IDeleteCardResponse | IErrorResponse,

IDeleteCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await CardService.deleteCard(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}

Листинг 3.2 – Структура cardController.ts для карточки

В папке *«errors»* описывается класс и конструкторы для обработки ошибок на стороне клиента.

Папка *«extra»* испольуется для дополнительного написания кода, в нем описываюся фалы с функциями.

В папке *«mid»* написаныпромежуточные и связующее фрагменты кода в конвейере приложения, используемый для обработки запросов и ответов.

Папка *«router»* в себе содержит компоненты для с отслеживаниия состояния, который заставляет работать все остальные компоненты навигации и хуки, решение для переключения и маршрутизации страниц.

В router мы определим маршруты для модульной структуры нашего кода обработки маршрута и выполнения в контроллере методов и событий.

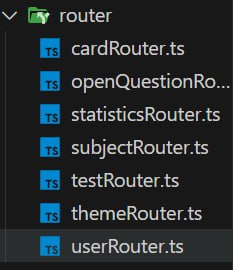


Рисунок 3.4 – Структура папки router

import { Router } from 'express'

import ThemeEndpoints from '../api/themes/endpoints'

import ThemeController from '../controllers/themeController'

const themeRouter = Router()

themeRouter.get(

ThemeEndpoints.GET\_BY\_ID,

ThemeController.getThemeById

)

themeRouter.get(

ThemeEndpoints.GET\_ALL\_THEMES,

ThemeController.getAllThemes

)

themeRouter.post(

ThemeEndpoints.CREATE,

ThemeController.createTheme

)

themeRouter.patch(

ThemeEndpoints.UPDATE,

ThemeController.updateTheme

)

themeRouter.delete(

ThemeEndpoints.DELETE,

ThemeController.deleteTheme

)

export default themeRouter

Листинг 3.3 – Структура themeRouter.ts для темы

И последняя папочка *«service»*, где инкапсулируется определенная бизнес-логика или функциональность приложения. Он предоставляет интерфейс для взаимодействия с другими компонентами приложения и абстрагирует детали реализации функциональности. Классы *service* обычно используются в объектно-ориентированном программировании для достижения разделения задач и для того, чтобы сделать код более модульным и удобным в обслуживании.

А здесь уже отдельная бизнес-логика для функциональности с предметом, (создание, удаление, обнавление, получение всех предметов и т.д.)

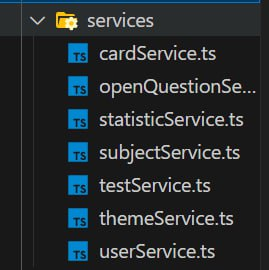


Рисунок 3.5 – Структура папки service

import { ICreateSubjectRequest } from "../api/subjects/reg/createSubject";

import { IDeleteSubjectRequest } from "../api/subjects/reg/deleteSubject";

import { IUpdateSubjectRequest } from "../api/subjects/reg/updateSubject";

import { IGetAllSubjectsRequest } from "../api/subjects/reg/getAllSubjects";

import { IGetSubjectBySubIdRequest } from "../api/subjects/reg/getSubjectBySubId";

import UserRequestError from "../errors/userRequestError";

import prismaClient from "../prismaClient";

export default class SubjectService {

//get

static getSubjectBySubId = async ({

id\_subject,

}: IGetSubjectBySubIdRequest) =>

prismaClient.subjects.findUnique({

where: { id\_subject: +id\_subject },

});

static getAllSubjects = async ({

cursor,

subjectName,

skip,

take,

}: IGetAllSubjectsRequest) =>

prismaClient.subjects.findMany({

skip,

take,

cursor: cursor ? { id\_subject: cursor } : undefined,

where: { subjectName: { contains: subjectName, mode: "insensitive" } },

});

//create

static createSubject = async ({ subjectName }: ICreateSubjectRequest) => {

const subject = await prismaClient.subjects.findUnique({

where: { subjectName },

select: { id\_subject: true },

});

if (subject)

throw UserRequestError.NotFound(

SUBJECT WITH NAME ${subjectName} NOT CREATED

);

return prismaClient.subjects.create({

data: {

subjectName,

},

});

};

//update

static updateSubject = async ({

id\_subject,

subjectName,

}: IUpdateSubjectRequest) => {

const subject = await prismaClient.subjects.findUnique({

where: { id\_subject },

select: { id\_subject: true },

});

if (!subject)

throw UserRequestError.NotFound(

SUBJECT WITH ID ${id\_subject} NOT FOUND

);

return prismaClient.subjects.update({

where: { id\_subject },

data: {

subjectName,

},

});

};

//delete

static deleteSubject = async ({ id\_subject }: IDeleteSubjectRequest) => {

const subject = await prismaClient.subjects.findUnique({

where: { id\_subject: id\_subject },

select: { id\_subject: true },

});

if (!subject)

throw UserRequestError.NotFound(

SUBJECT WITH ID ${id\_subject} NOT FOUND

);

return prismaClient.subjects.delete({

where: { id\_subject: id\_subject },

});

};

}

Листинг 3.4 – Структура subjectService.ts для предмета обучения

В конечном итоге получаем функциональный сервер, который позволяет обработать запрос от клиента.

### **3.1.1 Хэширование паролей**

Хеширование паролей — очень нужная и важная практика для обеспечения безопасности пользовательских учетных записей. Для данной цели был использован хэш SHA512 в десятеричном виде.

Данная операция происходит при создании пользователя и очень маловероятна она будет взломана.

Вот кусок кода с хэшированием (листинг 3.5).

return prismaClient.user.create({

data: {

username,

password: createHash("sha512").update(password).digest("hex"),

role,

},

include: { userToken: true },

});

Листинг 3.5 – Хэширование пароля

А также мы далее рассмотрим валидацию в данном web-приложении и как он описан.

### **3.1.2 Валидация данных**

Для валидации данных на стороне сервера был создан класс MainValidator и вместе с ним в папке «Validators» лежат валидаторы для заполнения данными со стороны сервера. В пример дается листинг 3.6, валидатор для статистики.

import MainValidator from "./mainValidator";

import { TLocation } from "./types";

export default class ProductDataValidator extends MainValidator {

static user\_id(location: TLocation) {

return MainValidator.id(location, "theme\_id");

}

static rightAnswered(location: TLocation, isOptional = false) {

return isOptional

? location("rightAnswered")

.optional({ values: "undefined" })

.isInt({ min: 0, max: 32767 })

.withMessage("SHOULD BE AN INT >= 0 AND <= 32767")

.toInt()

: location("rightAnswered")

.isInt({ min: 0, max: 32767 })

.withMessage("SHOULD BE AN INT >= 0 AND <= 32767")

.toInt();

}

static mark(location: TLocation, isOptional = false) {

return isOptional

? location("mark")

.optional({ values: "undefined" })

.isInt({ min: 0, max: 32767 })

.withMessage("SHOULD BE AN INT >= 0 AND <= 32767")

.toInt()

: location("mark")

.isInt({ min: 0, max: 32767 })

.withMessage("SHOULD BE AN INT >= 0 AND <= 32767")

.toInt();

}

static title(

location: TLocation,

isOptional: boolean = true,

length: { min?: number; max: number } | undefined = undefined

) {

return MainValidator.title(location, isOptional, length, "title");

}

}}

Листинг 3.6 – Валидация для статистики

Ну а теперь, мы плавно перейдем к клиентской части данного web-приложения и рассмотрим ее поподробнее.

* 1. **Разработка клиентской части**

Клиентская часть приложения связана непосредственно с технологий TypeScript, например, с использованием React.

Самым основным инструментом в нашей клиентской части являются различные хуки, такие как: useState и useEffect.

Первый позволяет добавлять функциональные компоненты и он возвращает такие знвчения как: текущее состояние и для егор добавления функцию. А второй выполняет эффекты в функциональных компонентах, который представляет собой подписку на события и загрузку данных или изменение DOM.

Также еще есть разные другие хуки, но они в большинстве своем в данном проекте не рассматривались.

Стиль контента на странице использовался с помощью SCSS, расширение языка CSS, которое дает дополнительные функции и возможности для более эффективного написания и организации стилей.

Для обращение к серверу используется Axios, библиотека JavaScript, которая предоставляет простой в использовании интерфейс для выполнения HTTP-запросов из браузера или из Node.js. Она широко используется в различных фреймворках и библиотеках, включая React.

В контексте React, Axios может быть использован для выполнения HTTP-запросов на сервер для получения данных, отправки данных на сервер или выполнения других операций, связанных с HTTP.

Структура клиента представлена на рисунке 3.6.

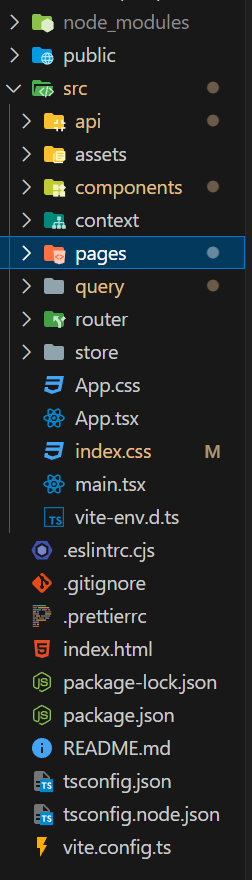


Рисунок 3.6 – Структура клиента

По итогу получается простой и динамический дизайн, в котором используются выше описанные инструменты.

* 1. **Развертывание приложения на базе Docker-Desktop**

В наше время огромное количество оперативных систем, которые имеют свою инфраструктуру. И в следствии этого, созданное на одном ОС приложение, не заработает на другом. Чтобы оно запустилось на другой ОС, нужно что-то дополнительно докачивать. Поэтому нам в помощь идет Docker.

Docker — программное обеспечение для автоматизации, развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений, позволяет запускать приложения в изолированном окружении [6].

Его работа основана на принципах клиент-серверной архитектуры, которая основана на взаимодействии клиента с веб-сервером. Клиент отправляет запросы на получение данных, а сервер их предоставляет.

Web-приложение было разработано на ОС Windows 10, на этом ОС будет процесс настройки администрирования web-приложения.

Для того, чтобы реализовать «контейнеризацию», нужно создать docker-файл, в которой описана конфигурация.

|  |
| --- |
| FROM node:21-alpine3.18  WORKDIR /app  COPY . .  WORKDIR /app/server  RUN npm i  RUN npx prisma generate  RUN npm run compile  WORKDIR /app/client  RUN npm i  WORKDIR /app  RUN chmod +x startup.sh  CMD ["/bin/sh", "startup.sh"] |

Листинг 3.4 – Структкра Dockerfile

## Сначала мы скачиваем образ из DockerHub платформы node.js. Далее, мы копируем содержимое в папку app для публикации, работа приложения будет осуществляться на раннее описанных портах и последнее, это запуск файла startup.sh.

Для дальнейшой работы, нам понадобиться обновить систему Windows Subsystem for Linux (WSL). Windows Subsystem for Linux (WSL) — слой совместимости для запуска Linux-приложений (двоичных исполняемых файлов в формате ELF) в ОС Windows 10.

После установки приложения, мы его запускаем. Для корректной работы надо иметь достаточно места и создать два контейнера: с приложением и с БД.

Сперва необходимо загрузить нужный образ БД с репозитория образов, после чего создать конфигурационный файл для Docker Compose для сборки всего проекта в один контейнер.

Docker Compose — это средство для определения и запуска приложений Docker с несколькими контейнерами. При работе в Compose вы используете файл YAML для настройки служб приложения. Затем вы создаете и запускаете все службы из конфигурации путем выполнения одной команды.

Находясь в папке с созданным конфигурационным файлом, остается ввести команду docker-compose up в консоль и контейнер начнет работать. Подключаемся по ip и порту к клиенту и можно приступать к работе.

## **3.4 Вывод**

По результату проведенной работы в данном разделе, было разработано приложение программно удовлетворяющее функционалу, определенному ранее.

После разработки стоит приступить к следующей не менее важной части разработки любого приложения – к тестированию. Так как, прежде чем оно увидит свет, оно должно быть протестировано в основных местах, чтобы при взаимодействии с пользователем были учтены и исправлены все ошибки.

1. **Тестирование**

Тестирование - направление, которое позволяет выявить подобные ошибки, проблемы и баги еще на пути массового тиражирования программы. Эксперт в мире тестирования ищет пробелы в коде, которые вызывают проблемы и заставляют программу работать не так, как задумал инженер-разработчик.

Основные задачи тестирования:

* выявить ошибки и оперативно их устранить;
* проверить соответствие продукта заявленным требованиям;
* оценить качество работы разработчиков;
* получить информацию, необходимую для принятия дальнейших решений.

Во время разработки было принято решено, провести ручное тестирование.

## **4.1 Ручное тестирование**

Тестирование включает в себе проверку правильности осуществления перехода между страницами приложения, которые совершают пользователи системы, также проверку правильности отправления запросов, их обработки, а также правильности отображение представлений.

Рассмотрим вариант авторизации пользователя, когда он вводит не верно, а также неправильно вводит пароль. При заполнении формы неправильными данными должно быть получено сообщение об ошибке. Заполнение формы неправильными данными изображено на рисунке 4.1.

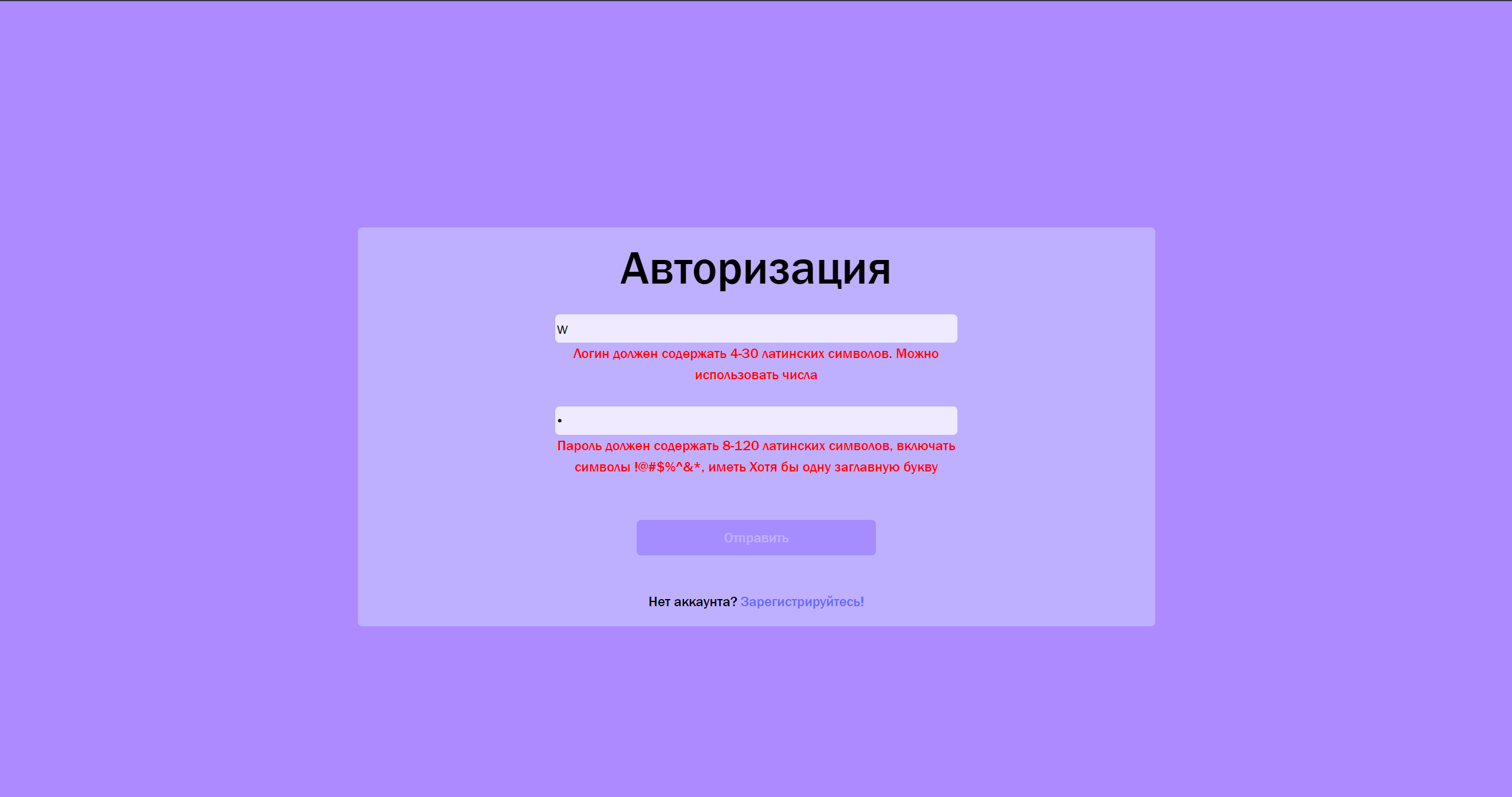


Рисунок 4.1 – Форма авторизации с неверными данными

Теперь заполним поля авторизации правильно и войдем в систему под всеми пользователями. А начнем мы с роли администратора, у которого при успешном входе отображается страница управления пользователями, как показано на рисунке 4.2.

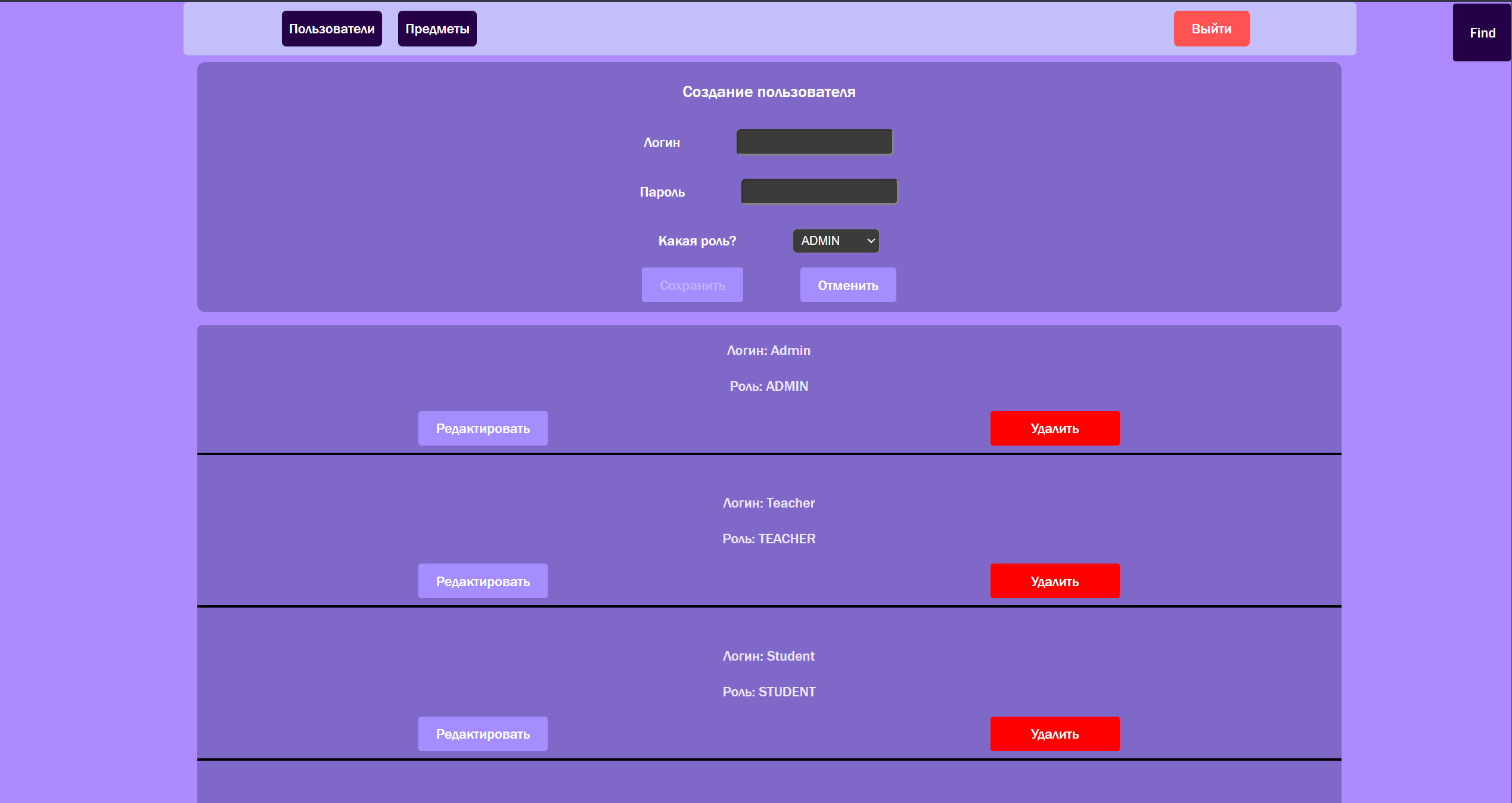


Рисунок 4.2 – Форма авторизации с верными данными

Если в качестве логина ввести специальные символы или кириллицу, то кнопка создания не должна быть доступна, а границы текстового поля красятся в красный цвет. Тоже самое, если ввести маленький пароль, кириллицу и не соблюсти правила написания пароля, то кнопка создания будет недоступна, а границы текстового поля подсветится красным, тоже самое при редактировании пользователя (рисунок 4.3).

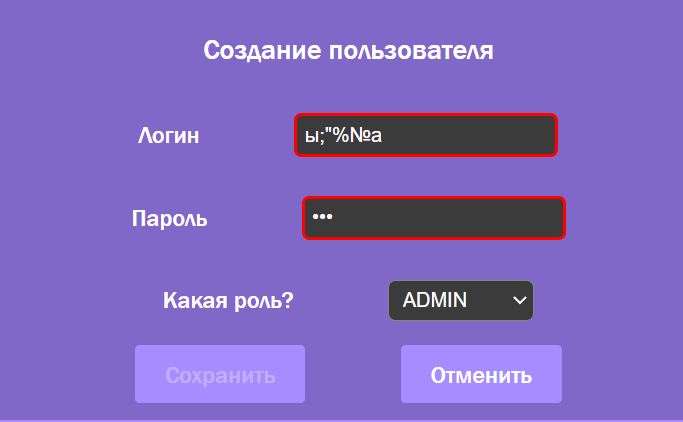


Рисунок 4.3 – Форма создания пользователя

Также, если не вводить ничего в строку, то кнопка будет не доступна, как показано на рисунке 4.4



Рисунок 4.4 – Форма создания учебного предмета

А сейчас мы проверим соотношение тем к тестам. Мы создадим тему с режимом «тест» рисунок 4.5, проверим ее создание среди всех тем 4.6, а потом конкретно среди режима «тест» на рисунке 4.7.

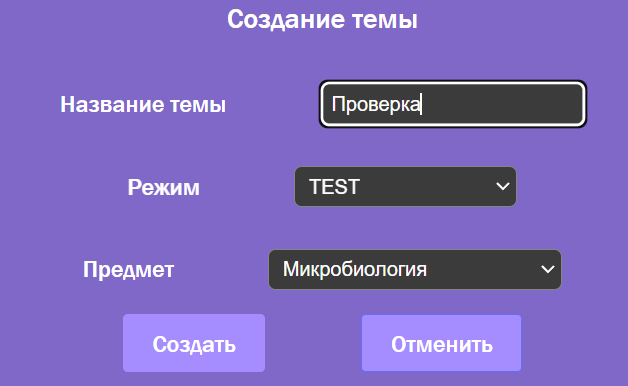


Рисунок 4.5 – Форма создания темы



Рисунок 4.6 – Список всех тем

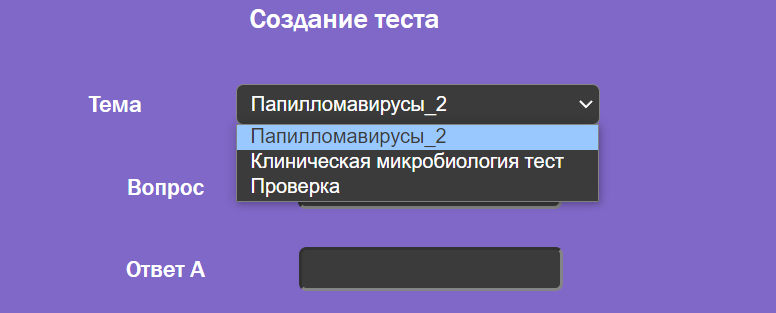


Рисунок 4.7 – Список тем с режимом «тест»

И заключительное, что мы проверим, это получение статистики, пройденого теста учеником преподавателю. Зайдем за пользователя и выберем учебный предмет, как на рисунке 4.8, далее выбираем тему учебного предмета (рисунок 4.9) и отвечаем на вопрос правильно как показано на рисунке 4.10 и смотрим ее у преподователя на странице (рисунок 4.11).

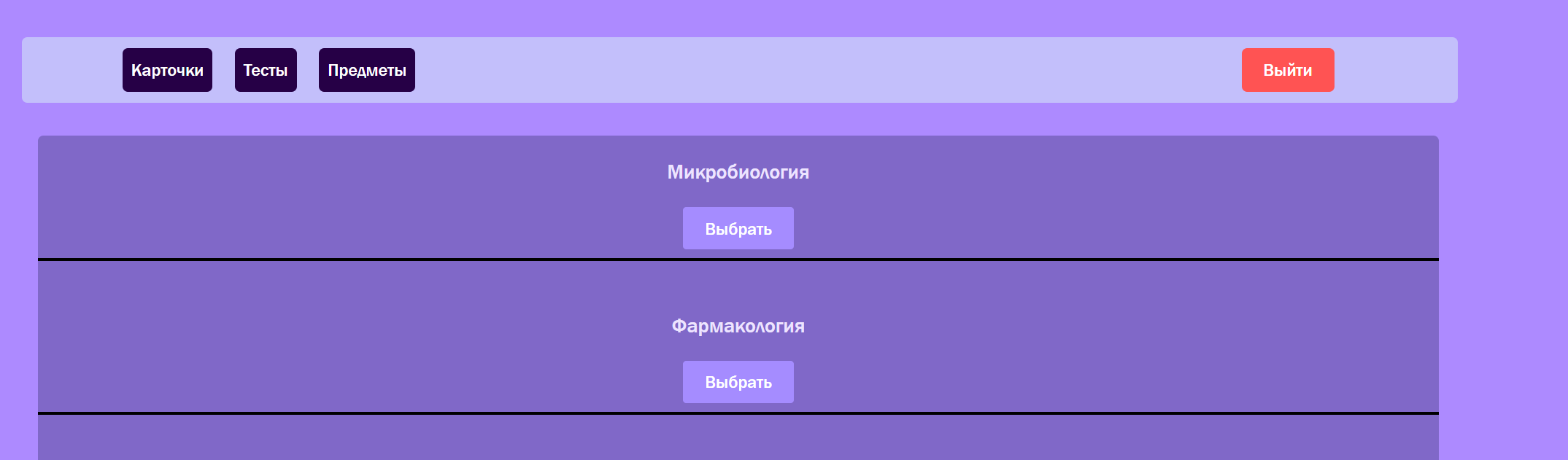


Рисунок 4.8 – Список всех учебных предметов

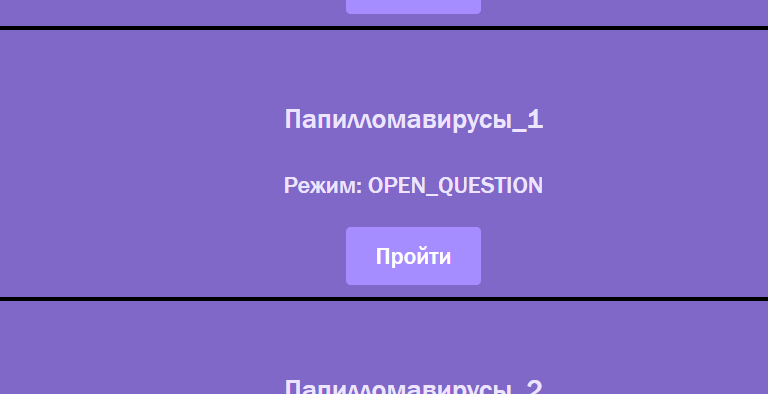


Рисунок 4.9 – Список всех тем данного учебного предмета

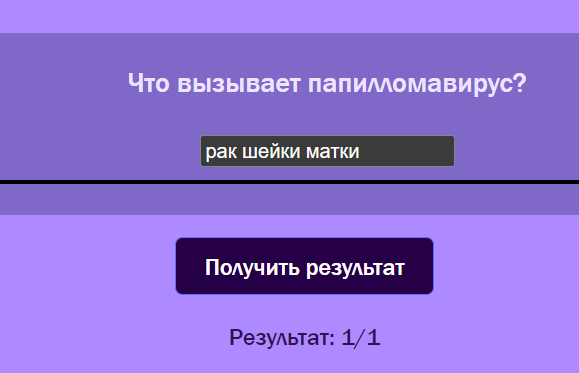


Рисунок 4.10 – Правильный ответ на вопроси иотображение результата

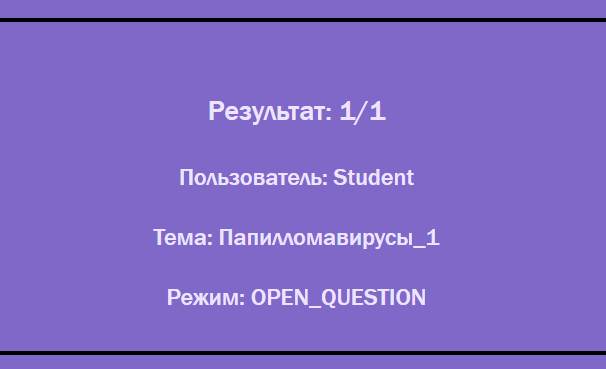


Рисунок 4.11 – Отображение статистики преподователю

По итогу этой проверки, преподователь может проверять учеников, зная кто, как и на какую тему отвечал.

## **4.2 Вывод**

После проведения ручного тестирования, можно сделать вывод, что покрываемый функционал работает правильно и никаких ошибок в ходе исполнения замечено не было. Все функции приложения отрабатывают корректно, а в случае, если возникает ошибка пользователь получает соответствующее уведомление. В свою очередь это говорит о грамотном проектировании приложения.

# **5 Руководство пользователя**

Главной задачей курсового проекта является создание веб-приложения, которое бы позволяло создать обучающие курсы. Для этого программно было реализовано с помощью Visual Studio взаимодействие с элементами страниц приложения.

## **5.1 Запуск проекта**

В данном руководстве описано как пользователю работать с веб-приложением, которое имеет первоначальный вид авторизации, представленный на рисунке 5.1.

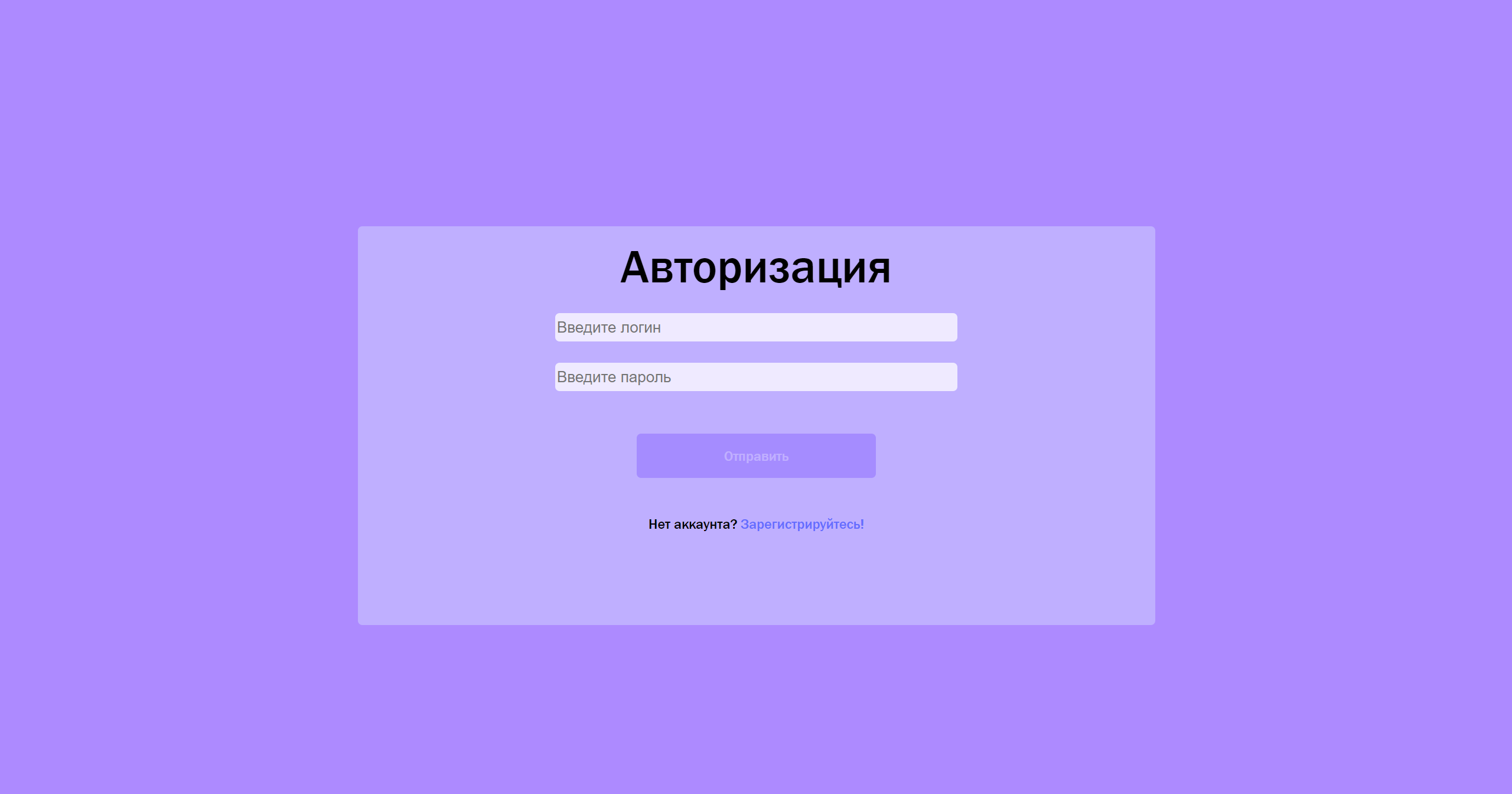


Рисунок 5.1 – Страница авторизации

Также новый пользователь может войти в систему зарегестривавшись в ней как видно по рисунку 5.2.

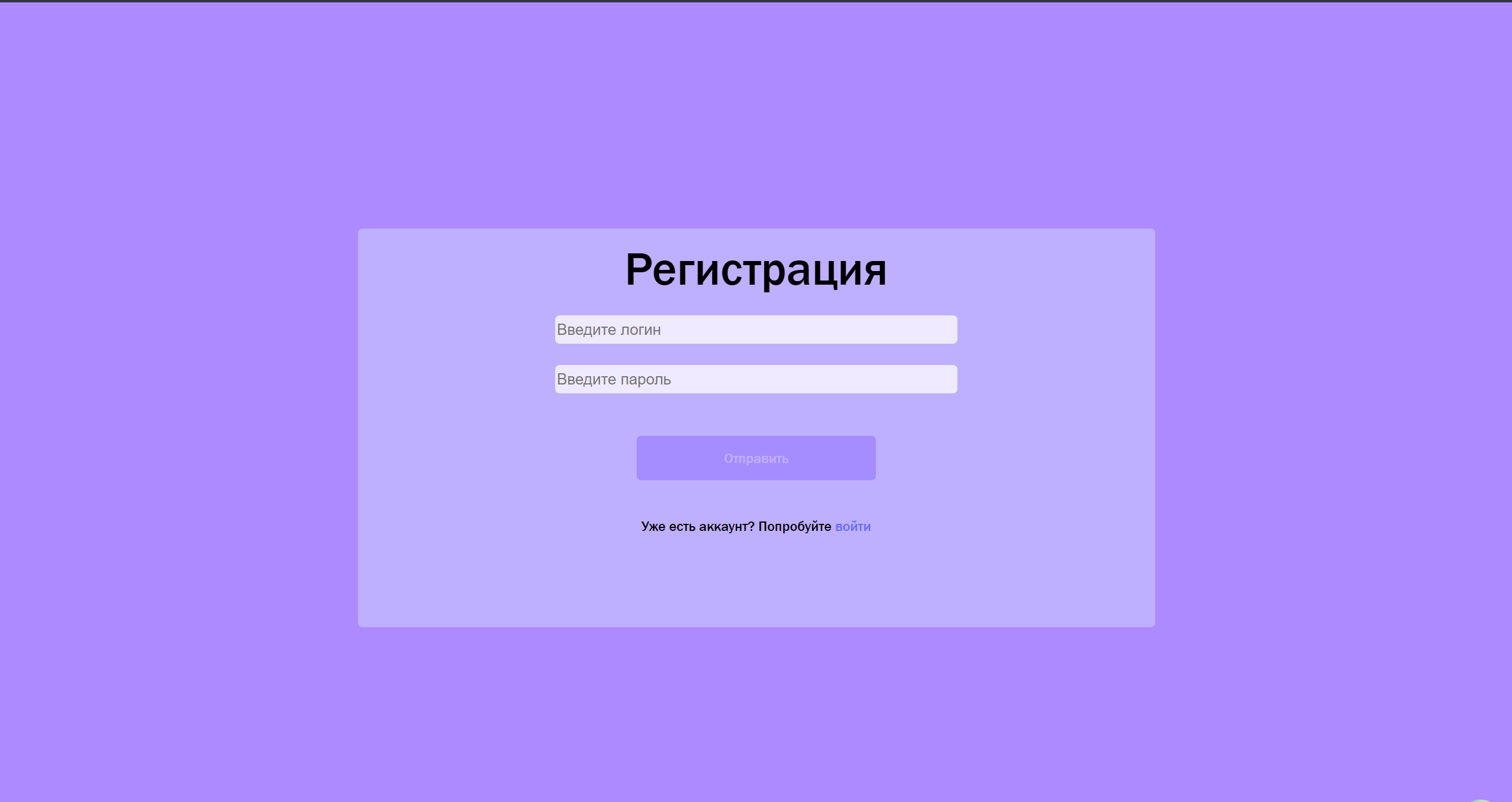


Рисунок 5.2 – Страница регистрации

После прохождения регистрации, пользователь может войти в систему как студент, либо можно авторизоваться под администратора, преподователя и студента.

## **5.2 Руководство администратора**

В случае успешной авторизации в аккаунт пользователя с ролью администратора, пользователь увидит экран с пользователями (рисунок 5.3)

На этой странице пользователь может управлять пользователями (создавать, обнавлять, удалять) (рисунок 5.3) или же перейти к другой странице приложения «Предметы», используя навигацию там тоже можно управлять предметами, как показано на рисунке 5.4.

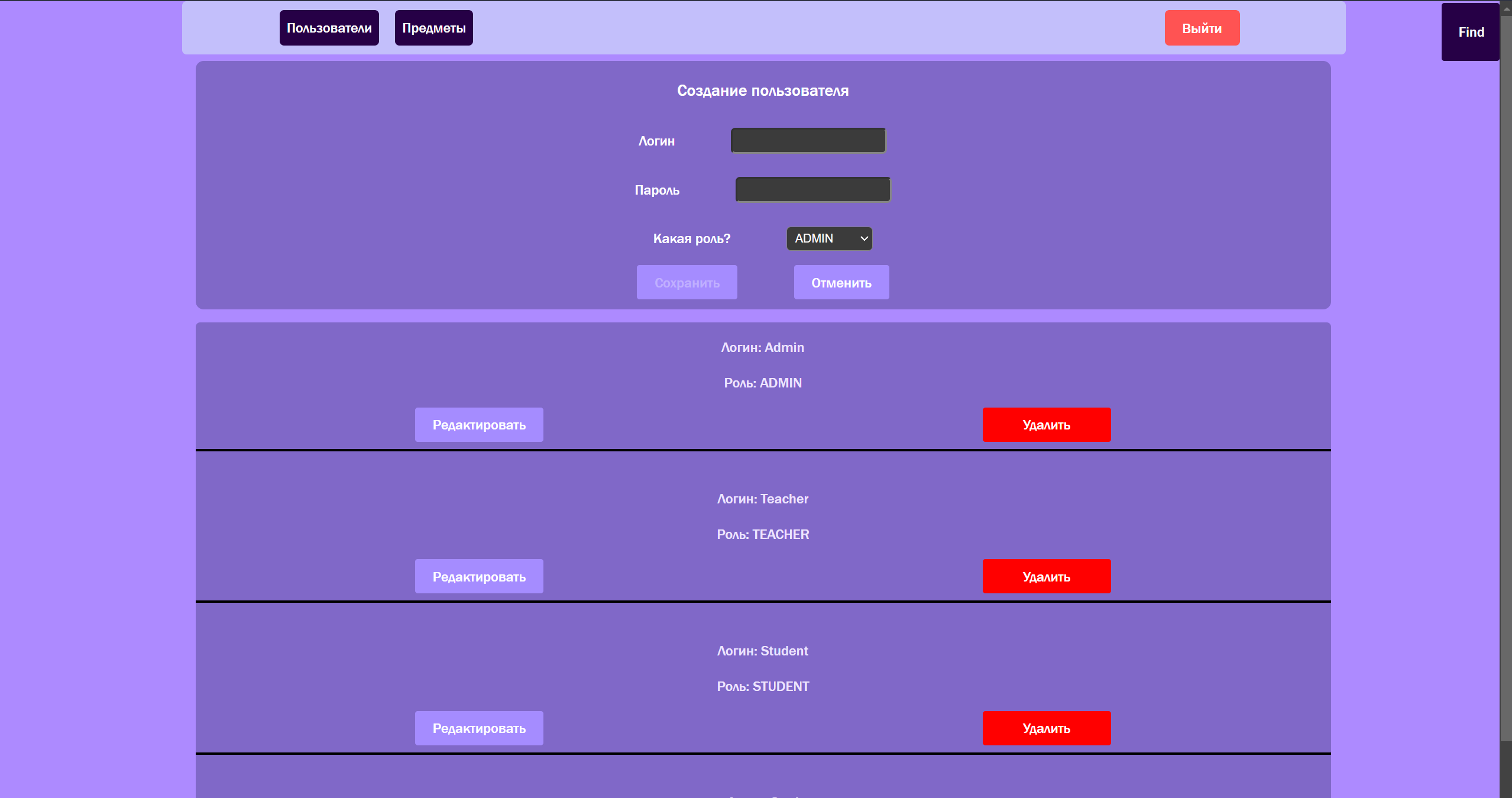


Рисунок 5.3 – Страница админа с пользователями

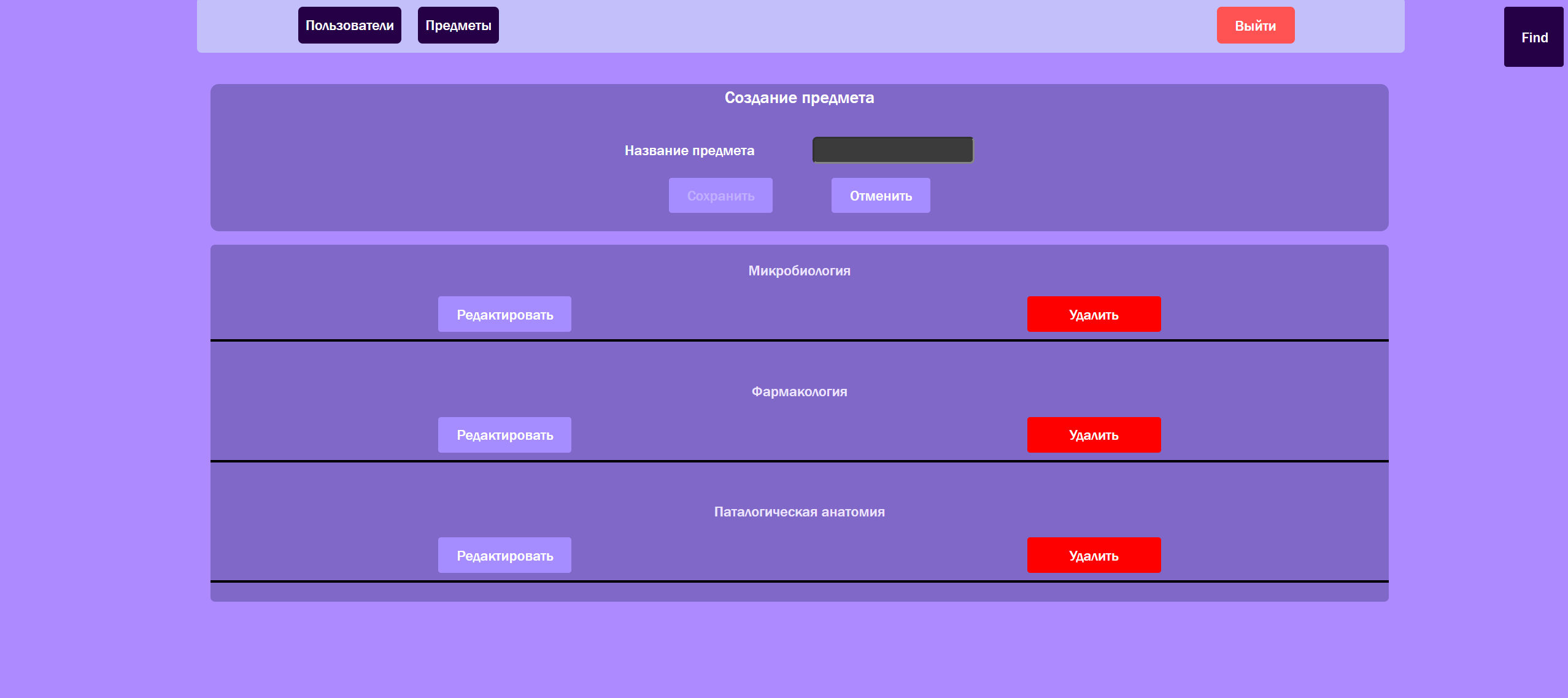


Рисунок 5.4 – Страница админа с учебными предметами

После создания предмета от лица администратора, у роли преподователя появляется возможность создавать темы по этому учебному предмету.

## **5.3 Руководство преподавателя**

Как уже говорилось, после создания администратором учебного предмета, у преподователя есть возможность управлять темами этого предмета, а потом и видами закрепления знаний по этим темам. Для начала рассмотрим страницу управления темами предметов, (рисунок 5.5), на которой можно назвать тему, выбрать режим из трех (тесты, открытые вопросы и карточки), а также выбор предмета, из всех созданых администратором, по-которому делается эта тема.

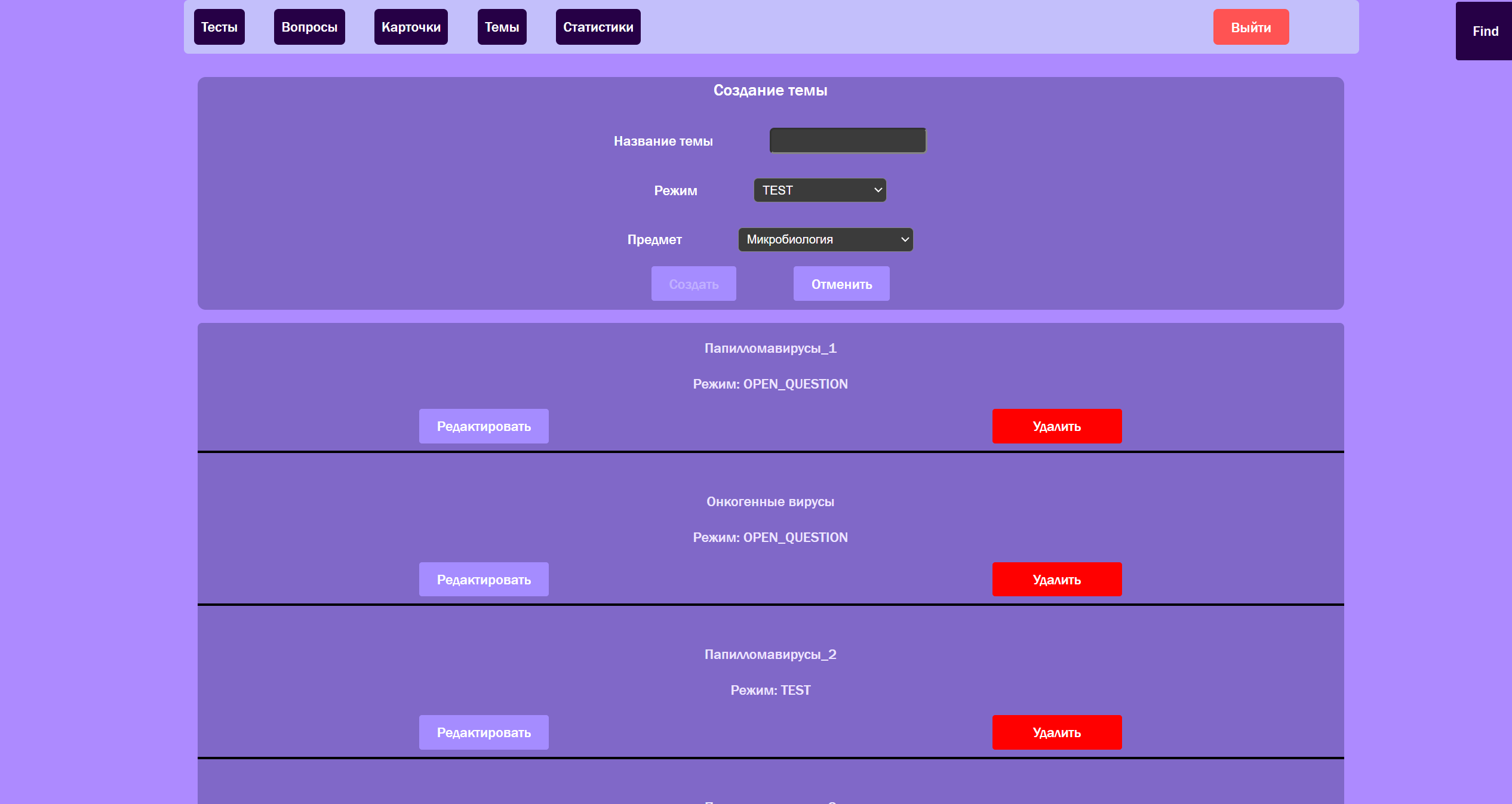


Рисунок 5.5 – Страница преподователя с темами предметов

После создания темы предмета, таким же способом, на трех вкладках по этой теме мы можем создовать выды закрепления знаний, такие как тесты, открытые вопросы и карточки. Рассмотрим на примере страницу открытых вопросов, как показано на рисунке 5.6

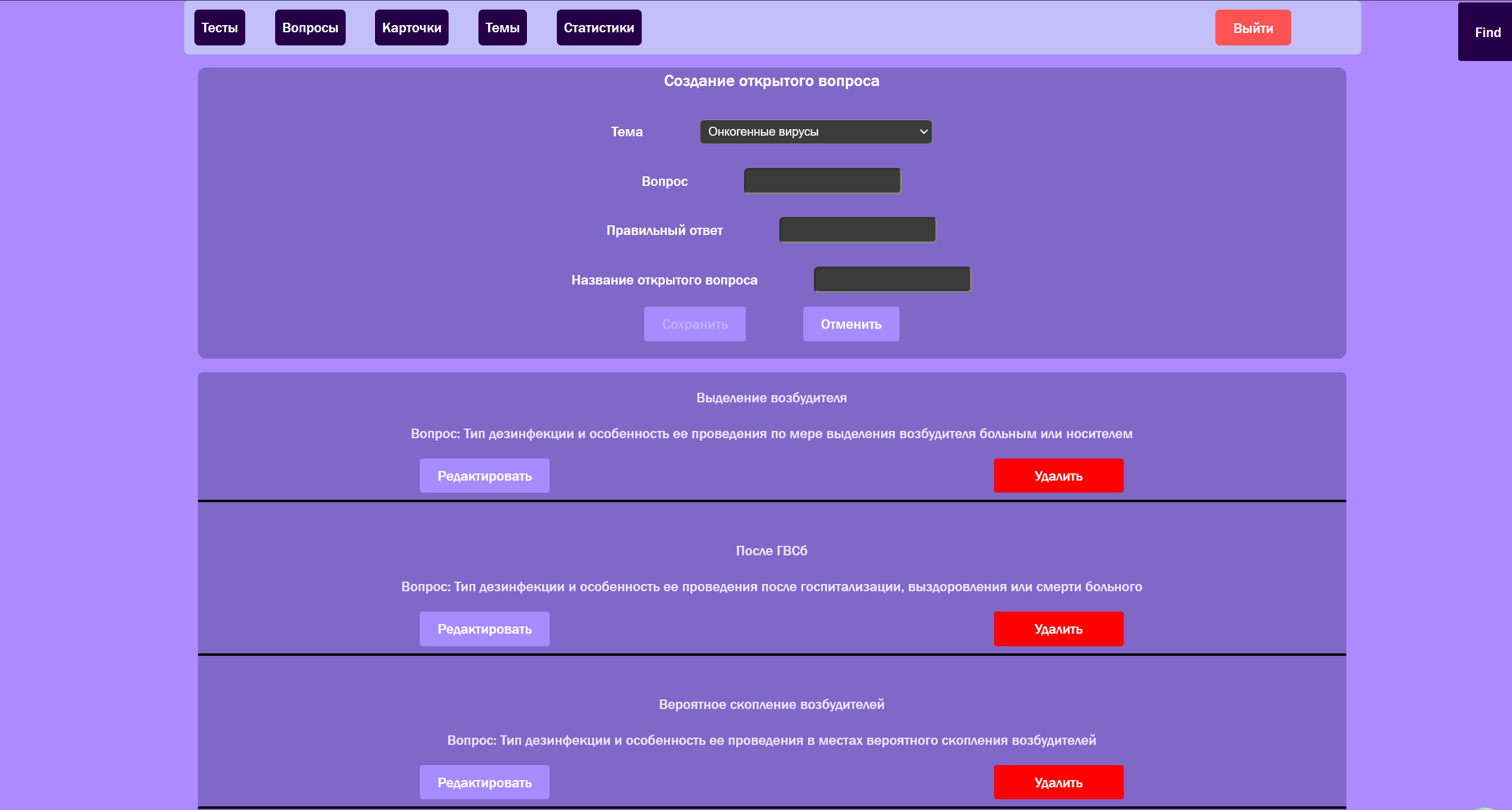


Рисунок 5.6 – Страница преподователя с вопросами по теме

Таким образом преподователь управляет темами и по ним видами закрепления знаний, но также, преподователь может отслеживать успехи студентов на странице статистики, где показв=ываются результаты прохождения материалов (рисунок 5.7).

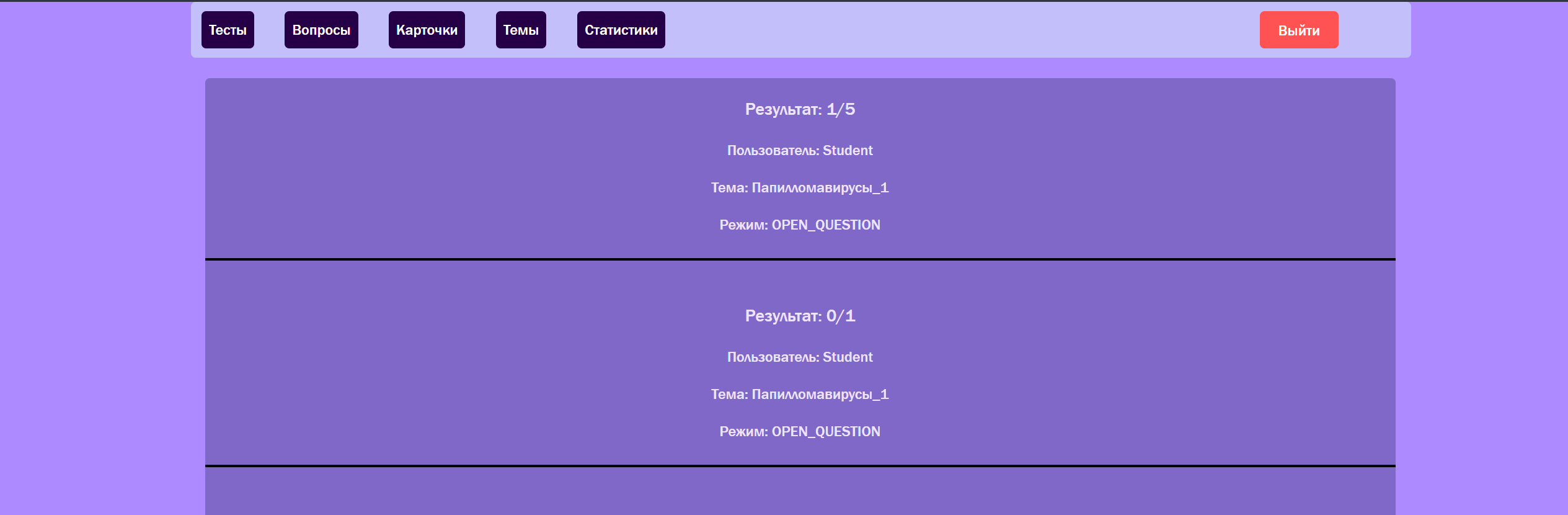


Рисунок 5.7 – Страница преподователя со статистикой студентов

Таким вот образом происходит работа за ролью преподователя в данном web-приложении.

## **5.4 Руководство студента**

И на последок осталась заключительная роль студента, за которой мы можем уже проходить созданные преподователем материалы для закрепления знаний. Вначале встречает страница с учебными предметами на выбор для дальнейшего проходжления, как на рисунке 5.8.

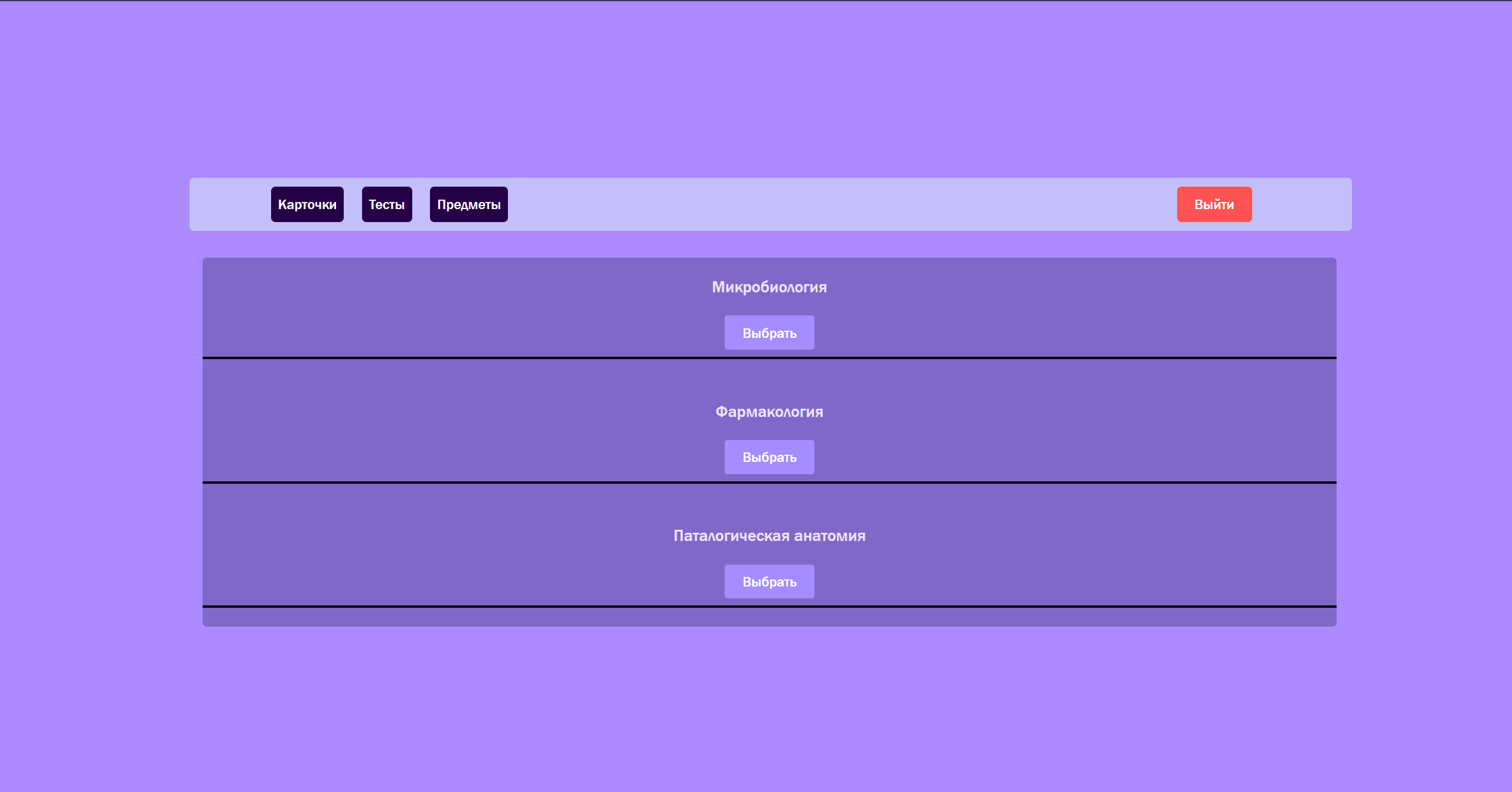


Рисунок 5.8 – Страница студента со списком предметов на выбор

После выбора учебного предмета, выбирается тема для этого предмета (рисунок 5.9).

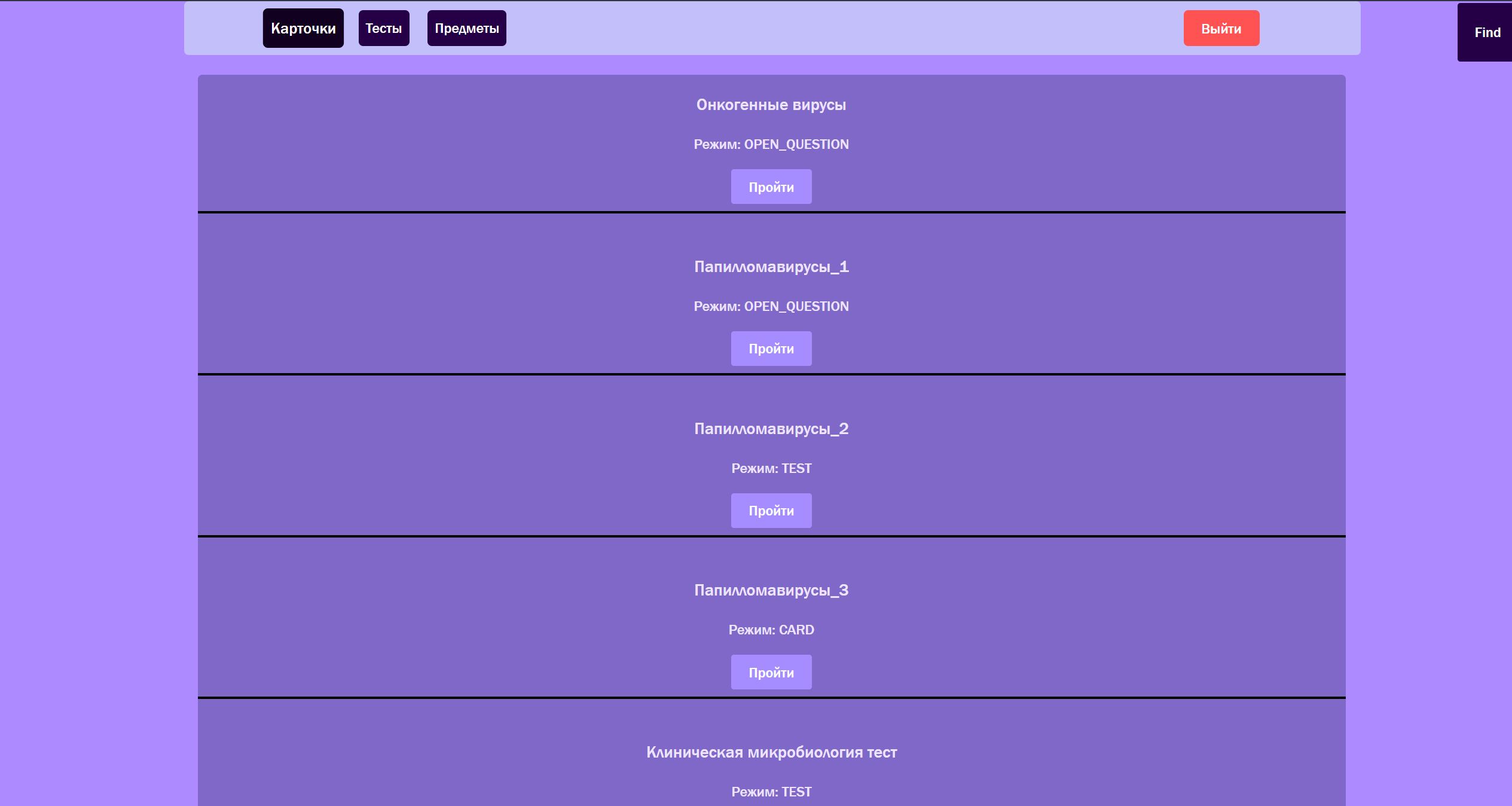


Рисунок 5.9 – Страница студента со списком тем предмета на выбор

А теперь самое интересное, это прохлждение материалов, для нашего закрепления знаний. Нажимаем на пройти и отвечаем на вопросы (рисунок 5.10).

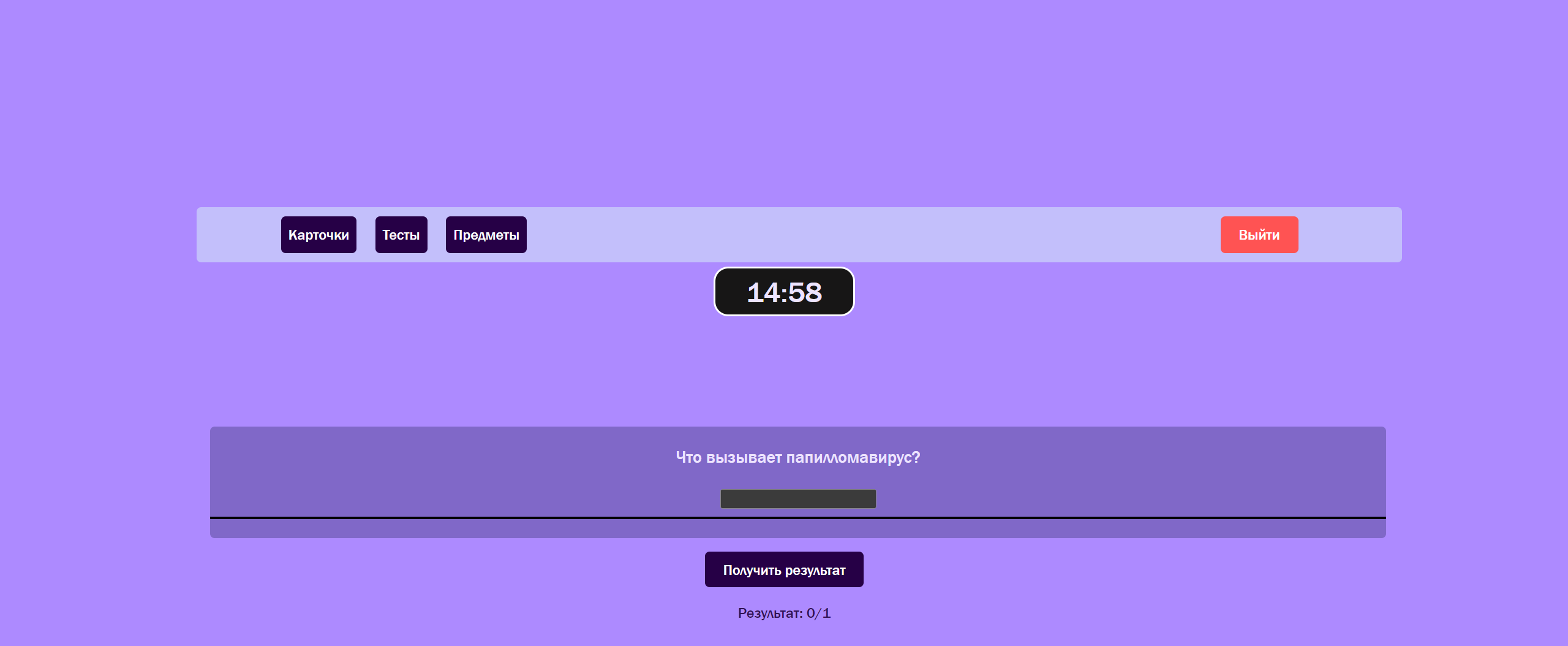


Рисунок 5.10 – Страница студента с прохождением вопросов по теме на выбор

После прохождения материалов, надо нажать на кнопку получить результат и мы можем его увидеть.

У студента остались вкладки управления тестами и карточками, для своего саморазвития, с ними он может только создавать и удалять. И это бло все руководство по приложению.

## **5.4 Вывод**

Подводя итог руководства пользователя, можно отметить, что весь необходимый функционал для администрирования был предоставлен в полном объеме, как и функционал приложения. А также пользователь-учитель обладает широким доступом к ресурсам для работы с веб-приложением.

Заключение

В ходе выполнения курсового проекта мной были рассмотрены основные аналоги разрабатываемого программного средства. Были выявлены основные недостатки рассматриваемых систем. Затем был сформирован набор функциональных требований, покрывающих необходимый функционал.

Для реализации приложения использовалась платформа node.js. Node.js может работать поверх кроссплатформенной среды, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux.

Была разработана реляционная база данных для веб-приложения, использующая экземпляр PostgreSQL, имеющая в своем наборе 9 таблиц.

Были созданы дополнительные конфигурационные файлы, позволяющие быстро и легко администрировать, расширять функционал приложения.

В результате проведенной работы, был создан проект, который удовлетворяет потребностям разных пользователей и корректным образом выполняет все необходимые функции.

Также я получила как опыт проектирования, так и опыт разработки приложения. Приобрела опыт работы с большим стеком технологий и изучила новые.

Опыт развертывания приложений, полученный во время выполнения курсового проекта, считаю немаловажным дополнением ко всем приобретённым навыкам разработки программного средства.

# **Список используемых источников**

1. Quizizz [Электронный ресурс]. – Режим доступа: https://quizizz.com/?lng=en. – Дата доступа: 05.10.2023.

2. Kahoot! [Электронный ресурс]. – Режим доступа: https://kahoot.com/. – Дата доступа: 05.10.2023.

3. Prisma: Полноценный ORM для Node.js и TypeScript [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/588562/>. – Дата доступа: 15.12.2023.

4. Node JS — Router and Routes [Электронный ресурс]. – Режим доступа: https://medium.com/swlh/node-js-router-and-routes-a4a3cfced5c4. – Дата доступа: 14.12.2023.

5. How To Use NodeJS Service Class [Электронный ресурс]. – Режим доступа: <https://medium.com/@soulaimaneyh/how-to-use-nodejs-service-class-a1e919a5cbd8>. – Дата доступа: 16.12.2023.

6. Руководство по Docker [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/ruvds/blog/450312//. – Дата доступа: 17.12.2023.

**Приложение А**

Листинг контроллера «cardController»

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateCardRequest,

ICreateCardResponse,

} from "../api/cards/reg/createCard";

import {

IDeleteCardRequest,

IDeleteCardResponse,

} from "../api/cards/reg/deleteCard";

import {

IUpdateCardRequest,

IUpdateCardResponse,

} from "../api/cards/reg/updateCard";

import {

IGetAllCardsRequest,

IGetAllCardsResponse,

} from "../api/cards/reg/getAllCards";

import {

IGetCardByIdRequest,

IGetCardByIdResponse,

} from "../api/cards/reg/getCardById";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import CardService from "../services/cardService";

import UserRequestError from "../errors/userRequestError";

export default class CardController {

//get

static getAllCards: RequestHandler<

undefined,

IGetAllCardsResponse | IErrorResponse,

undefined,

IGetAllCardsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.getAllCards(req.query);

res.json({

cardsData: result,

cursor: result[result.length - 1]?.id\_card || null,

});

} catch (e) {

return next(e);

}

};

static getCardById: RequestHandler<

undefined,

IGetCardByIdResponse | IErrorResponse,

undefined,

IGetCardByIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.getCardById(req.query);

if (!result)

return next(

UserRequestError.NotFound(

`CARD WITH ID ${req.query.id\_card} NOT FOUND`

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

//create

static createCard: RequestHandler<

undefined,

ICreateCardResponse | IErrorResponse,

ICreateCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.createCard(req.body);

res.status(201).json(result);

} catch (e) {

return next(e);

}

};

//update

static updateCardData: RequestHandler<

undefined,

IUpdateCardResponse | IErrorResponse,

IUpdateCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await CardService.updateCardData(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

//delete

static deleteCard: RequestHandler<

undefined,

IDeleteCardResponse | IErrorResponse,

IDeleteCardRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await CardService.deleteCard(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}

**Приложение Б**

Листинг контроллера «openQuestionController»

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateOpenQuestionRequest,

ICreateOpenQuestionResponse,

} from "../api/openQuestions/reg/createOpenQuestion";

import {

IDeleteOpenQuestionRequest,

IDeleteOpenQuestionResponse,

} from "../api/openQuestions/reg/deleteOpenQuestion";

import {

IUpdateOpenQuestionRequest,

IUpdateOpenQuestionResponse,

} from "../api/openQuestions/reg/updateOpenQuestion";

import {

IGetAllOpenQuestionsRequest,

IGetAllOpenQuestionsResponse,

} from "../api/openQuestions/reg/getAllOpenQuestions";

import {

IGetOpenQuestionByIdRequest,

IGetOpenQuestionByIdResponse,

} from "../api/openQuestions/reg/getOpenQuestionById";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import OpenQuestionsService from "../services/openQuestionService";

import UserRequestError from "../errors/userRequestError";

export default class OpenQuestionController {

//get

static getAllOpenQuestions: RequestHandler<

undefined,

IGetAllOpenQuestionsResponse | IErrorResponse,

undefined,

IGetAllOpenQuestionsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await OpenQuestionsService.getAllOpenQuestions(req.query);

res.json({

openQuestionsData: result,

cursor: result[result.length - 1]?.id\_openQuestion || null,

});

} catch (e) {

return next(e);

}

};

static getOpenQuestionById: RequestHandler<

undefined,

IGetOpenQuestionByIdResponse | IErrorResponse,

undefined,

IGetOpenQuestionByIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await OpenQuestionsService.getOpenQuestionById(req.query);

if (!result)

return next(

UserRequestError.NotFound(

`OPEN QUESTION WITH ID ${req.query.id\_openQuestion} NOT FOUND`

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

//create

static createOpenQuestion: RequestHandler<

undefined,

ICreateOpenQuestionResponse | IErrorResponse,

ICreateOpenQuestionRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await OpenQuestionsService.createOpenQuestion(req.body);

res.status(201).json(result);

} catch (e) {

return next(e);

}

};

//update

static updateOpenQuestionData: RequestHandler<

undefined,

IUpdateOpenQuestionResponse | IErrorResponse,

IUpdateOpenQuestionRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await OpenQuestionsService.updateOpenQuestionData(

req.body

);

res.json(result);

} catch (e) {

return next(e);

}

};

//delete

static deleteOpenQuestions: RequestHandler<

undefined,

IDeleteOpenQuestionResponse | IErrorResponse,

IDeleteOpenQuestionRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await OpenQuestionsService.deleteOpenQuestions(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}

# **Приложение** В

Листинг контроллера «statisticsController»

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

IUpdateStatisticRequest,

IUpdateStatisticResponse,

} from "../api/statistics/reg/updateStatistics";

import {

IGetAllStatisticsRequest,

IGetAllStatisticsResponse,

} from "../api/statistics/reg/getAllStatistics";

import {

IGetStatisticByUserIdRequest,

IGetStatisticByUserIdResponse,

} from "../api/statistics/reg/getStatisticByUserId";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import StatisticService from "../services/statisticService";

import UserRequestError from "../errors/userRequestError";

export default class StatisticsController {

//get

static getAllStatistics: RequestHandler<

undefined,

IGetAllStatisticsResponse | IErrorResponse,

undefined,

IGetAllStatisticsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await StatisticService.getAllStatistics(req.query);

res.json({

statisticsData: result,

cursor: result[result.length - 1]?.id\_statistics || null,

});

} catch (e) {

return next(e);

}

};

static getStatisticByUserId: RequestHandler<

undefined,

IGetStatisticByUserIdResponse[] | IErrorResponse,

IGetStatisticByUserIdRequest

// undefined,

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await StatisticService.getStatisticByUserId(req.body);

if (!result) return next(UserRequestError.NotFound(""));

res.json(result);

} catch (e) {

return next(e);

}

};

//update

static updateStatisticsData: RequestHandler<

undefined,

IUpdateStatisticResponse | IErrorResponse,

IUpdateStatisticRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await StatisticService.updateStatisticsData(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

}

# **Приложение Г**

Листинг контроллера «subjectController»

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateSubjectRequest,

ICreateSubjectResponse,

} from "../api/subjects/reg/createSubject";

import {

IDeleteSubjectRequest,

IDeleteSubjectResponse,

} from "../api/subjects/reg/deleteSubject";

import {

IGetSubjectBySubIdRequest,

IGetSubjectBySubIdResponse,

} from "../api/subjects/reg/getSubjectBySubId";

import {

IUpdateSubjectRequest,

IUpdateSubjectResponse,

} from "../api/subjects/reg/updateSubject";

import {

IGetAllSubjectsRequest,

IGetAllSubjectsResponse,

} from "../api/subjects/reg/getAllSubjects";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import SubjectService from "../services/subjectService";

import UserRequestError from "../errors/userRequestError";

export default class SubjectController {

static getSubjectBySubId: RequestHandler<

undefined,

IGetSubjectBySubIdResponse | IErrorResponse,

undefined,

IGetSubjectBySubIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await SubjectService.getSubjectBySubId(req.query);

if (!result)

return next(

UserRequestError.NotFound(

`SUBJECT WITH ID ${req.query.id\_subject} NOT FOUND`

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

static getAllSubjects: RequestHandler<

undefined,

IGetAllSubjectsResponse | IErrorResponse,

undefined,

IGetAllSubjectsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await SubjectService.getAllSubjects(req.query);

res.json({

subjectsData: result,

cursor: result[result.length - 1]?.id\_subject || null,

});

} catch (e) {

return next(e);

}

};

//create

static createSubject: RequestHandler<

undefined,

ICreateSubjectResponse | IErrorResponse,

ICreateSubjectRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await SubjectService.createSubject(req.body);

res.status(201).json({

...result,

});

} catch (e) {

return next(e);

}

};

//update

static updateSubjectData: RequestHandler<

undefined,

IUpdateSubjectResponse | IErrorResponse,

IUpdateSubjectRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await SubjectService.updateSubject(req.body);

res.json({

...result,

});

} catch (e) {

return next(e);

}

};

//delete

static deleteSubject: RequestHandler<

undefined,

IDeleteSubjectResponse | IErrorResponse,

IDeleteSubjectRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await SubjectService.deleteSubject(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}

# **Приложение** **Д**

Листинг контроллера «testController»

import { RequestHandler } from "express";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateTestRequest,

ICreateTestResponse,

} from "../api/tests/reg/createTest";

import {

IDeleteTestRequest,

IDeleteTestResponse,

} from "../api/tests/reg/deleteTest";

import {

IUpdateTestRequest,

IUpdateTestResponse,

} from "../api/tests/reg/updateTest";

import {

IGetAllTestsRequest,

IGetAllTestsResponse,

} from "../api/tests/reg/getAllTests";

import {

IGetTestByIdRequest,

IGetTestByIdResponse,

} from "../api/tests/reg/getTestById";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import TestsService from "../services/testService";

import UserRequestError from "../errors/userRequestError";

export default class TestController {

//get

static getAllTests: RequestHandler<

undefined,

IGetAllTestsResponse | IErrorResponse,

undefined,

IGetAllTestsRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await TestsService.getAllTests(req.query);

res.json({

testsData: result,

cursor: result[result.length - 1]?.id\_test || null,

});

} catch (e) {

return next(e);

}

};

static getTestById: RequestHandler<

undefined,

IGetTestByIdResponse | IErrorResponse,

undefined,

IGetTestByIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await TestsService.getTestById(req.query);

if (!result)

return next(

UserRequestError.NotFound(

`TEST WITH ID ${req.query.id\_test} NOT FOUND`

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

//create

static createTest: RequestHandler<

undefined,

ICreateTestResponse | IErrorResponse,

ICreateTestRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await TestsService.createTest(req.body);

res.status(201).json(result);

} catch (e) {

return next(e);

}

};

//update

static updateTestData: RequestHandler<

undefined,

IUpdateTestResponse | IErrorResponse,

IUpdateTestRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await TestsService.updateTestData(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

//delete

static deleteTest: RequestHandler<

undefined,

IDeleteTestResponse | IErrorResponse,

IDeleteTestRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await TestsService.deleteTest(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}

# **Приложение Е**

Листинг контроллера «themeController»

import { RequestHandler } from 'express'

import { IErrorResponse } from '../api/errorResponse'

import {

ICreateThemeRequest,

ICreateThemeResponse,

} from '../api/themes/reg/createTheme'

import {

IDeleteThemeRequest,

IDeleteThemeResponse,

} from '../api/themes/reg/deleteTheme'

import {

IGetThemeByIdRequest,

IGetThemeByIdResponse,

} from '../api/themes/reg/getThemeById'

import {

IUpdateThemeRequest,

IUpdateThemeResponse,

} from '../api/themes/reg/updateTheme'

import {

IGetAllThemesRequest,

IGetAllThemesResponse,

} from '../api/themes/reg/getAllThemes'

import callUnprocessableEntity from '../extra/callUnprocessableEntity'

import getValidationResult from '../extra/getValidationResult'

import ThemeService from '../services/themeService'

export default class ThemeController {

//get

static getThemeById: RequestHandler<

undefined,

IGetThemeByIdResponse | IErrorResponse,

undefined,

IGetThemeByIdRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req)

if (errorData) return callUnprocessableEntity(next, errorData)

try {

const result = await ThemeService.getThemeById(req.query)

res.json(result)

} catch (e) {

return next(e)

}

}

static getAllThemes: RequestHandler<

undefined,

IGetAllThemesResponse | IErrorResponse,

undefined,

IGetAllThemesRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req)

if (errorData) return callUnprocessableEntity(next, errorData)

try {

const result = await ThemeService.getAllThemes(req.query)

res.json({

themesData: result,

cursor: result[result.length - 1]?.id\_theme || null,

})

} catch (e) {

return next(e)

}

}

//create

static createTheme: RequestHandler<

undefined,

ICreateThemeResponse | IErrorResponse,

ICreateThemeRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req)

if (errorData) return callUnprocessableEntity(next, errorData)

try {

const result = await ThemeService.createTheme(req.body)

res.status(201).json(result)

} catch (e) {

return next(e)

}

}

//update

static updateTheme: RequestHandler<

undefined,

IUpdateThemeResponse | IErrorResponse,

IUpdateThemeRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req)

if (errorData) return callUnprocessableEntity(next, errorData)

try {

const result = await ThemeService.updateTheme(req.body)

res.json(result)

} catch (e) {

return next(e)

}

}

//delete

static deleteTheme: RequestHandler<

undefined,

IDeleteThemeResponse | IErrorResponse,

IDeleteThemeRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req)

if (errorData) return callUnprocessableEntity(next, errorData)

try {

await ThemeService.deleteTheme(req.body)

res.json({ count: 1 })

} catch (e) {

return next(e)

}

}

}

**Приложение Ж**

Листинг контроллера «userController»

import { RequestHandler } from "express";

import { createHash } from "node:crypto";

import { IErrorResponse } from "../api/errorResponse";

import {

ICreateUserRequest,

ICreateUserResponse,

} from "../api/users/reg/createUser";

import {

IDeleteUsersRequest,

IDeleteUsersResponse,

} from "../api/users/reg/deleteUsers";

import {

IDeleteUserRequest,

IDeleteUserResponse,

} from "../api/users/reg/deleteUser";

import {

IGetAllUsersRequest,

IGetAllUsersResponse,

} from "../api/users/reg/getAllUsers";

import {

IGetUserByUsernameRequest,

IGetUserByUsernameResponse,

} from "../api/users/reg/getUserByUsername";

import {

IGetUserTokensRequest,

IGetUserTokensResponse,

} from "../api/users/reg/getUserTokens";

import {

ILoginUserRequest,

ILoginUserResponse,

} from "../api/users/reg/loginUser";

import {

IUpdateUserDataRequest,

IUpdateUserDataResponse,

} from "../api/users/reg/updateUserData";

import UserRequestError from "../errors/userRequestError";

import callUnprocessableEntity from "../extra/callUnprocessableEntity";

import getValidationResult from "../extra/getValidationResult";

import Tokenizator from "../extra/tokenizator";

import UserService from "../services/userService";

export default class UserController {

//login

static loginUser: RequestHandler<

undefined,

ILoginUserResponse | IErrorResponse,

ILoginUserRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const user = await UserService.getUserByUsername(req.body);

if (!user)

return next(

UserRequestError.NotFound(`USER ${req.body.username} NOT FOUND`)

);

if (

createHash("sha512").update(req.body.password).digest("hex") !==

user.password

)

return next(UserRequestError.BadRequest("WRONG PASSWORD"));

const { refreshToken } = Tokenizator.generateTokens({

username: user.username,

role: user.role,

});

res

.cookie("refreshToken", refreshToken, {

maxAge: 30 \* 24 \* 60 \* 60 \* 1000,

httpOnly: true,

})

.json({

refreshToken,

username: user.username,

role: user.role,

id\_user: user.id\_user,

});

} catch (e) {

return next(e);

}

};

//get

static getUserByUsername: RequestHandler<

IGetUserByUsernameRequest,

IGetUserByUsernameResponse | IErrorResponse

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await UserService.getUserByUsername(req.params);

if (!result)

return next(

UserRequestError.NotFound(

`USER WITH LOGIN ${req.params.username} NOT FOUND`

)

);

res.json(result);

} catch (e) {

return next(e);

}

};

static getAllUsers: RequestHandler<

undefined,

IGetAllUsersResponse | IErrorResponse,

undefined,

IGetAllUsersRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await UserService.getAllUsers(req.query);

res.json({

usersData: result,

cursor: result[result.length - 1]?.id\_user || null,

});

} catch (e) {

return next(e);

}

};

static getUserTokens: RequestHandler<

undefined,

IGetUserTokensResponse[] | IErrorResponse,

undefined,

IGetUserTokensRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await UserService.getUserTokens(req.query);

res.json(result);

} catch (e) {

return next(e);

}

};

//create

static createUser: RequestHandler<

undefined,

ICreateUserResponse | IErrorResponse,

ICreateUserRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const { refreshToken } = Tokenizator.generateTokens(req.body);

const result = await UserService.createUser({

...req.body,

refreshToken,

});

res.cookie("refreshToken", refreshToken, {

maxAge: 30 \* 24 \* 60 \* 60 \* 1000,

httpOnly: true,

});

res.status(201).json(result);

} catch (e) {

return next(e);

}

};

//update

static updateUserData: RequestHandler<

undefined,

IUpdateUserDataResponse | IErrorResponse,

IUpdateUserDataRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await UserService.updateUserData(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

//delete

static deleteUsers: RequestHandler<

undefined,

IDeleteUsersResponse | IErrorResponse,

IDeleteUsersRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

const result = await UserService.deleteUsers(req.body);

res.json(result);

} catch (e) {

return next(e);

}

};

static deleteUser: RequestHandler<

undefined,

IDeleteUserResponse | IErrorResponse,

IDeleteUserRequest

> = async (req, res, next) => {

const errorData = getValidationResult(req);

if (errorData) return callUnprocessableEntity(next, errorData);

try {

await UserService.deleteUser(req.body);

res.json({ count: 1 });

} catch (e) {

return next(e);

}

};

}