Частное образовательное учреждение высшего образования

«Казанский инновационный университет имени В. Г. Тимирясова (ИЭУП)»

Факультет менеджмента и инженерного бизнеса

Кафедра информационных технологий и техносферной безопасности

**КурсовОЙ ПРОЕКТ**

**по дисциплине «Интернет-программирование»**

**на тему: «**Создание системы «PRObatio» с базой данных проверочных заданий для полуавтоматической подготовки тестовых наборов.**»**

Направление подготовки: 09.03.03 «Прикладная информатика»

**Выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

обучающийся гр. 1091

очной формы обучения

Кавкаева Татьяна Александровна

**Руководитель:\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

доцент кафедры КТиБ

Фасхутдинов Л. М.

**Результат защиты:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Казань, 2022**

Оглавление

[Введение 4](#_Toc102793659)

[1 Проектирование системы 6](#_Toc102793660)

[1.1 Описание предметной области 6](#_Toc102793661)

[1.2 Структура проекта. Карта сайта 7](#_Toc102793662)

[1.3 Проектирование базы данных 11](#_Toc102793663)

[2 Разработка системы 15](#_Toc102793664)

[2.1 Разработка интерфейса системы 15](#_Toc102793665)

[Интерфейс страницы «Вход» 15](#_Toc102793666)

[Интерфейс страницы «Восстановление логина/пароля» 15](#_Toc102793667)

[Интерфейс страницы «Личный кабинет (Администратор)» 15](#_Toc102793668)

[Интерфейс страницы «Создание нового пользователя 15](#_Toc102793669)

[Интерфейс страницы «Личный кабинет (Преподаватель)» 15](#_Toc102793670)

[Интерфейс страницы «Создание теста» 15](#_Toc102793671)

[Интерфейс страницы «Тестирование» 15](#_Toc102793672)

[Интерфейс страницы «Успеваемость по тесту» 15](#_Toc102793673)

[Интерфейс страницы «Успеваемость студента» 15](#_Toc102793674)

[Интерфейс страницы «Личный кабинет (Студент)» 15](#_Toc102793675)

[2.2 Разработка базы данных 15](#_Toc102793676)

[2.3 Разработка серверной части системы 15](#_Toc102793677)

[3 Развертывание информационной системы 16](#_Toc102793678)

[3.1 Настройка и развертывание серверного программного обеспечения 16](#_Toc102793679)

[3.2 Тестирование системы 16](#_Toc102793680)

[Заключение 17](#_Toc102793681)

[Список использованных источников 18](#_Toc102793682)

[Приложение А. Детализированная карта сайта 19](#_Toc102793683)

[Приложение Б 20](#_Toc102793684)

# Введение

Тестирование является одной из самых популярных форм проверки знаний. В нем предлагаются несколько фиксированных вариантов ответов, некоторые из которых являются заведомо правильными. При проверке ответов на тестирование необходимо сравнивать данные тестируемым ответы с верными – то есть, действовать по строго фиксированному алгоритму. А всё, что действует по алгоритму, можно автоматизировать, что облегчит работу людям, занимающимися проведением тестирований (в данном случае преподавателей).

Автоматизация данного процесса поможет не только сохранить время людей, проводящих тестирование, но и помочь тестируемым людям – снижается риск появления ошибок в результатах тестирования, которые могут возникнуть из-за человеческого фактора.

Целью данного проекта является создание системы с базой данных проверочных заданий для полуавтоматической подготовки тестовых наборов

Для достижения данной цели были поставлены следующие задачи:

* исследование предметной области;
* разработка структуры проекта;
* проектирование базы данных;
* разработка интерфейса системы;
* разработка базы данных;
* разработка серверной части;
* настройка и развертывание информационной системы;
* тестирование системы.

Так как иметь доступ к системе необходимо большому количеству пользователей не только из учебного заведения, она будет представлена в виде web-сайта (страницы информации, связанные друг с другом гипертекстовыми ссылками, среди них есть главная страница, обычно включающая меню и несущая адрес домена. Эти страницы находятся на веб-сервере и доступны, если нет специальных ограничений, с любого браузера сети Интернет [1]).

В ходе выполнения данного проекта были использованы:

* программные средства: Visual Studio Code, GitHub;
* язык разметки: HTML5;
* язык стилей: CSS;
* язык сценариев: JavaScript;
* серверная платформа: node js;
* СУБД: PostgreSQL.

# 1 Проектирование системы

## 1.1 Описание предметной области

Разрабатываемая система в целом предназначена для тестирования. Но для разных видов пользователей она будет выполнять различные функции. Для их разграничения необходимо создать роли с различными возможностями.

Пользователями системы в первую очередь являются тестирующий человек и тестируемый, то есть, Преподаватель и Студент соответственно.

Преподаватель должен иметь возможность составлять и проходить тесты, а также просматривать успеваемость как конкретного студента по всем тестам, так и всех студентов по конкретному тесту.

Студент должен иметь возможность проходить тесты и смотреть свои результаты.

Каждому пользователю необходимо иметь Личный аккаунт для работы в системе. Предполагается, что он должен иметь к нему доступ сразу же, как только он поступил в вуз/устроился на работу. Чтобы не было возможности регистрироваться в системе вообще любому человеку, аккаунты новым пользователям будет заводить Администратор.

Администратор – роль пользователя, которая не принимает непосредственного участия в процессе тестирования, она необходимо для централизованной регистрации новых пользователей.

Администратор должен иметь возможность создавать новые аккаунты – для этого он должен вводить информацию о новом пользователе, которую ему передают вне системы. После ввода информации о пользователе автоматически генерируются логин и пароль от Личного кабинета и приходят на почту пользователя (она должна быть указана в информации о пользователе).

Так же все пользователи должны иметь возможности, связанные не с тестированием, а с взаимодействием с самим сервисом. Это стандартные функции, используемые в многопользовательских системах, а именно – вход и выход из аккаунта, восстановление логина или пароля и редактирование личных данных.

Таким образом, в разрабатываемой системе предусмотрены 3 роли: Преподаватель, Студент и Администратор. Все функции, необходимые для них представлены на рисунке 1.

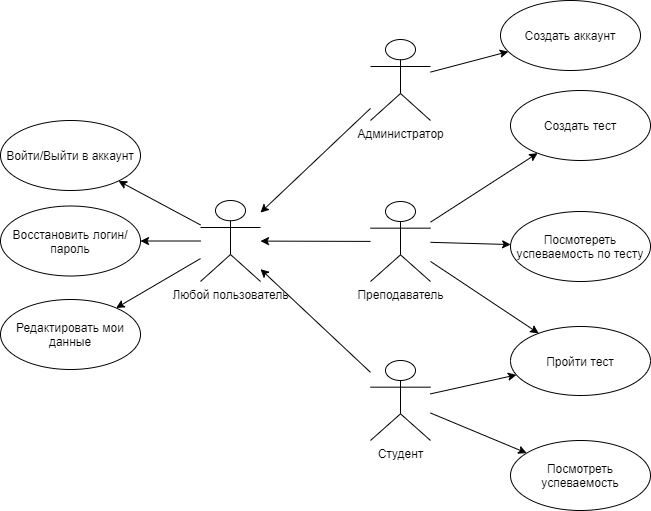


Рисунок 1 – Диаграмма вариантов использования

## 1.2 Структура проекта. Карта сайта

Проанализируем, какие страницы (Веб-страница – самостоятельная часть веб-сайта; документ, снабженный уникальным адресом (URL) [2]) понадобятся для обеспечения потребностей пользователей. Удобнее всего сделать это в таблице.

В таблице 1 представлены страницы, на которых реализуются необходимые для пользователей функции, а также их взаимосвязи (колонка Страницы, ведущие на текущую страницу).

Таблица 1 – Перечень web-страниц

|  |  |  |
| --- | --- | --- |
| Функция | Страница | Страницы, ведущие на текущую страницу |
| Вход в аккаунт | Вход | * Личный кабинет * Восстановление логина/пароля |
| Восстановление логина/пароля | Восстановление логина/пароля | Вход |
| Редактирование личных данных | Личный кабинет (Администратор) | Вход |
| Редактирование личных данных | Личный кабинет (Преподаватель) | Вход |
| Редактирование личных данных | Личный кабинет (Студент) | Вход |
| Создание нового аккаунта | Создать пользователя | Личный кабинет (Администратор) |
| Создание теста | Создать тест | Личный кабинет (Преподаватель) |
| Тестирование | Пройти тестирование | * Личный кабинет (Студент) * Личный кабинет (Преподаватель) |
| Просмотр успеваемости студента | Успеваемость | * Личный кабинет (Студент) * Личный кабинет (Преподаватель) |
| Просмотр успеваемости по тесту | Успеваемость по тесту | * Личный кабинет (Преподаватель) * Открыть тест |

Для наглядности данные из таблицы 1 представлены в виде карты сайта (рисунок 2).

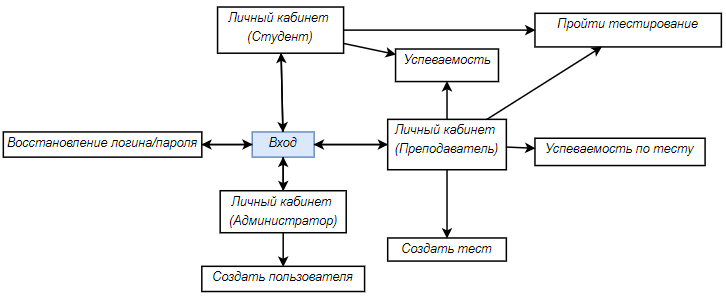


Рисунок 2 – Карта сайта

Детализированная карта сайта представлена в Приложении А. Она дополнена информацией о содержании каждой страницы.

Страницы представлены в виде таблиц, в верхней части которых размещены их названия, а в нижней – содержание (рисунок 3).

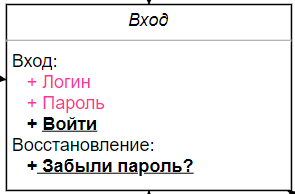


Рисунок 3 – Содержание страницы «Вход»

Содержание в свою очередь представлено в виде иерархического списка элементов страницы, которые сгруппированы по разделам. Разделы не всегда могут быть видны пользователю, они предназначены для обеспечения логической взаимосвязи элементов и удобства разработчика. Например, на рисунке 3 представлены 2 раздела – «Вход» и «Восстановление». Заголовок «Восстановление» не будет виден пользователю.

Элементы, зависящие от другого или являющиеся его частью, помещены на уровень ниже, чем этот элемент. Например, на рисунке 4 видимость «Количества попыток» зависит от того, включено ли пользователем «Ограничение по количеству попыток», а элемент «Пометить как правильный» является действием, которое пользователь может совершить с элементом «Вариант ответа».

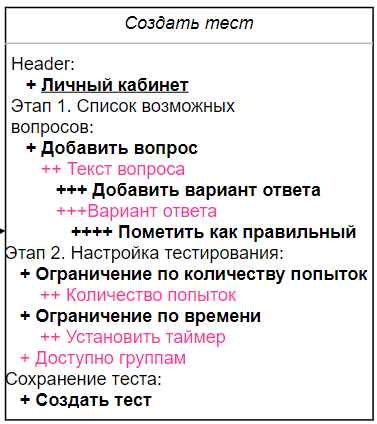


Рисунок 4 – Содержание страницы «Создать тест»

По своему назначению элементы могут быть разделены на 3 группы:

1. предназначенные для заполнения пользователем. Переданная в них информация не заносится в базу данных с их помощью (это делается с помощью элемента 2 группы).

2. предназначенные для выполнения какого-то функционала (запись полученной информации из элемента первой группы в бд, переход на новую страницу, отображение скрытых элементов страницы), действуют после взаимодействия с ними пользователем.

3. информативные – не несут в себе никакой функциональной значимости, предназначены только для чтения пользователем.

Для наглядности элементы данных групп выделяются на схеме:

* у элементов 1 группы шрифт розового цвета (на рисунке 3 пользователю необходимо ввести свои «Логин» и «Пароль»);
* у 2 – жирный (на рисунке 3 действие «Создать тест» позволяет записать в базу данных введённую Преподавателем информацию о тесте; выбор «Ограничения по времени» открывает возможность «Установить таймер»), те из элементов 2 группы, которые являются ссылками на другую страницу подчеркнуты (на рисунке 3 после ввода «Логина» и «Пароля» при помощи «Входа» осуществляется проверка наличия в системе пользователя с такими параметрами и, в случае успешного её прохождения, переход в «Личный кабинет» пользователя);
* так как элементы 3 группы не играют особой роли в функциональности системы, то и на схеме они не выделяются (на странице «Успеваемость», представленной на рисунке 5 пользователь может просмотреть личные данные (ФИО и номер группы) необходимого пользователя и успеваемость, представленную в виде таблицы с графами «Название теста», «Дата», «Баллы» и «Номер попытки», но он не может удалять илb редактировать эти данные).

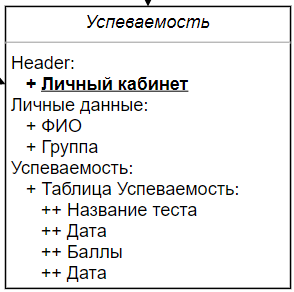


Рисунок 5 – Содержание страницы «Успеваемость»

Карта сайта является его предварительным представлением, конечный результат может отличаться от запланированного, так как в процессе реализации структура сайта может быть изменена.

## 1.3 Проектирование базы данных

При работе в системе необходимо где-то хранить данные. В данном проекте для этого используется база данных.

На основе анализа предметной области выделим данные, которые будут необходимы для работы системы.

Данные о пользователях должны содержать:

* Фамилия – текстовое значение;
* Имя – текстовое значение;
* Отчество – текстовое значение;
* Роль – выбирается из списка: «Администратор», «Преподаватель» и «Студент»;
* Телефон – текстовое значение;
* Email – текстовое значение;
* Логин – генерируется автоматически на основании имеющихся данных (Фамилия + порядковый номер, если уже есть пользователь с похожим логином), будет являться первичным ключом;
* Пароль – генерируется автоматически (может содержать в себе цифры, буквы английского алфавита (как строчные, так и заглавные));
* Группа, если Роль – Студент;
* Список прикрепленных студентов, если Роль – Преподаватель – содержит ссылки на прикрепленных пользователей с ролью Студент.

Для хранения информации о тестах необходимо создать сущность Тест, которая будет содержать в себе следующую информацию:

* Название – текстовое значение;
* Автор – ссылка на пользователя, создавшего этот тест;
* Время (если отсутствует ограничение по времени, то принимает значение 0);
* Количестве попыток (если отсутствует ограничение по количеству попыток, то принимает значение 0);
* Каким группам доступен тест – список;
* Каким студентам доступен тест – список;
* Вопросы и варианты ответов – хранится в формате json, шаблон: {‘текст\_вопроса1’:{{‘вариант\_ответа\_1’:‘правильный’}, {‘вариант\_ответа\_2’:‘правильный’}, {‘вариант\_ответа\_3’:‘правильный’}}, ‘текст\_вопроса2’:{{‘вариант\_ответа\_1’:‘правильный’}, {‘вариант\_ответа\_2’:‘правильный’}, {‘вариант\_ответа\_3’:‘правильный’}}, };

поле «правильный» имеет логический тип данных, где 1 – вариант является правильным, 0 – не является.

Так же необходимо хранить информацию о результатах тестирования, чтобы их можно было просматривать как для отдельного студента, так и для отдельного теста. Поэтому необходимо создать таблицу Тест-Студент, которая будет иметь атрибуты:

* Название теста – ссылка на Тест;
* Логин студента – ссылка на Студента, прошедшего Тест;
* Баллы – количество набранных баллов, численное значение;
* Дата – должна фиксироваться автоматически.

На рисунке 6 представлена ER-модель базы данных.

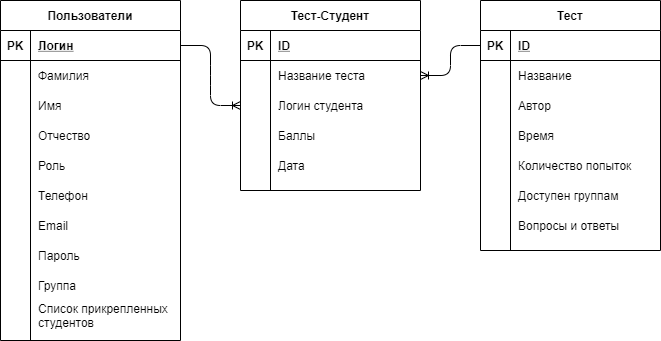


Рисунок 6 - ER-модель базы данных

# 2 Разработка системы

## 2.1 Разработка интерфейса системы

Создание внешнего вида страниц происходит при помощи html-документов. Они позволяют задать структуру и стиль страницы и имеют следующую структуру:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

</body>

</html>

Листинг 1 – Структура HTML-страницы

Открывающиеся теги (<тег>) ознаменовывают начало какого-то элемента документа, закрывающиеся (</тег>) – его конец. Они существуют для определения элемента. Теги имеют атрибуты (дополнительная информация; не являются обязательными).

В теге <head> размещены характеристики документа. В их число входят название страницы (которое отображается пользователю на вкладке), а также ссылки на файл css, стили из которого применяются в документе.

В теге <body> размещено само содержание страницы.

Каждый тип элемента имеет свой тег. В данном подразделе рассмотрены теги, которые были использованы для создания системы.

Для всех тегов применим атрибут style. Он позволяет задать внешний вид элемента. Для разграничения стиля и структуры страницы их обычно выносят в отдельный файл формата .css.

Стиль задаётся элементу путём присвоения ему класса (атрибут class) или уникального идентификатора (атрибут id). Отличие класса от уникального идентификатора в том, что id является уникальным, на странице не должно быть элементов с повторяющимися id, в то время как в класс может входить большое количество различных элементов.

Чтобы задать стиль для типа элемента в css необходимо написать название этого элемента, а рядом, в фигурных скобках писать необходимые характеристики. Например, так задаётся стиль для всего «тела» страницы:

body {background-color: red;}

Стили для классов задаются путем добавления «.» перед названием класса:

.название\_класса {height: 20px;}

для id – путем добавления «#» перед его названием:

#id {color: blue}

Как отмечалось ранее, страницы сайта обычно имеют одинаковый стиль, поэтому все стили, используемые в разрабатываемой системе, будут храниться в одном файле: PRObatio\general\css\style.css.

Поведение страницы при взаимодействии пользователя с ней описывается в виде алгоритма на языке сценариев. Он так же может храниться в html-файле страницы, но для разделения его со структурой документа можно вынести его в отдельный файл формата .js. В таком случае необходимо будет указать ссылку на файл.js с помощью тега <script src=””>, вставив его в самом конце тега <body>.

Страницы могут быть статическими – всегда одинаковы при любых условиях. и динамическими – такие страницы способны адаптироваться под конкретного пользователя. Создание статических страниц происходит с помощью обычных HTML-документов. Для динамических же используются шаблонизаторы. Как понятно из названия, они представляют собой шаблоны, в которые подставляются необходимые динамические данные, поэтому они занимают по 2 файла.

Статическими страницами являются страницы «Вход» и «Восстановление доступа», а динамическими – «Личные кабинеты», «Создание нового пользователя», «Создание теста», «Тестирование», «Успеваемость по тесту», «Успеваемость студента», «Результат».

### Интерфейс страницы «Вход»

Работа пользователя в системе начинается с главной страницы [3] – в PRObatio это страница «Вход».

Интерфейс данной страницы представлен на рисунке 7. HTML-код – в файле PRObatio\backend\html\general\index.html.

На данной странице используются следующие теги:

<div> используется для группировки элементов, а также задания им одинакового стиля;

<form> содержит в себе элементы, предназначенные для работы с пользователем, позволяет взаимодействовать с сервером [4];

<h3> позволяет выделять заголовок раздела;

<input> используется для получения данных от пользователя:

* с помощью свойства placeholder можно выводить на экран какой-то текст, пока пользователь сам не ввёл данные
* параметр required указывает на то, что поле обязательно к заполнению. В случае, если оно не заполнено, пользователь видит сообщение от системы об этом и не может пользоваться некоторыми функциями;
* атрибут minlength указывает на минимальное количество символов, которые должен ввести пользователь [5];

<button> - кнопка, пользователь ожидает, что при нажатии на неё произойдут какие-то действия;

<a> используется для указания ссылки на другую страницу (с его помощью кнопка «Забыли пароль?» перенаправляет пользователя на страницу «Восстановление логина/пароля»)

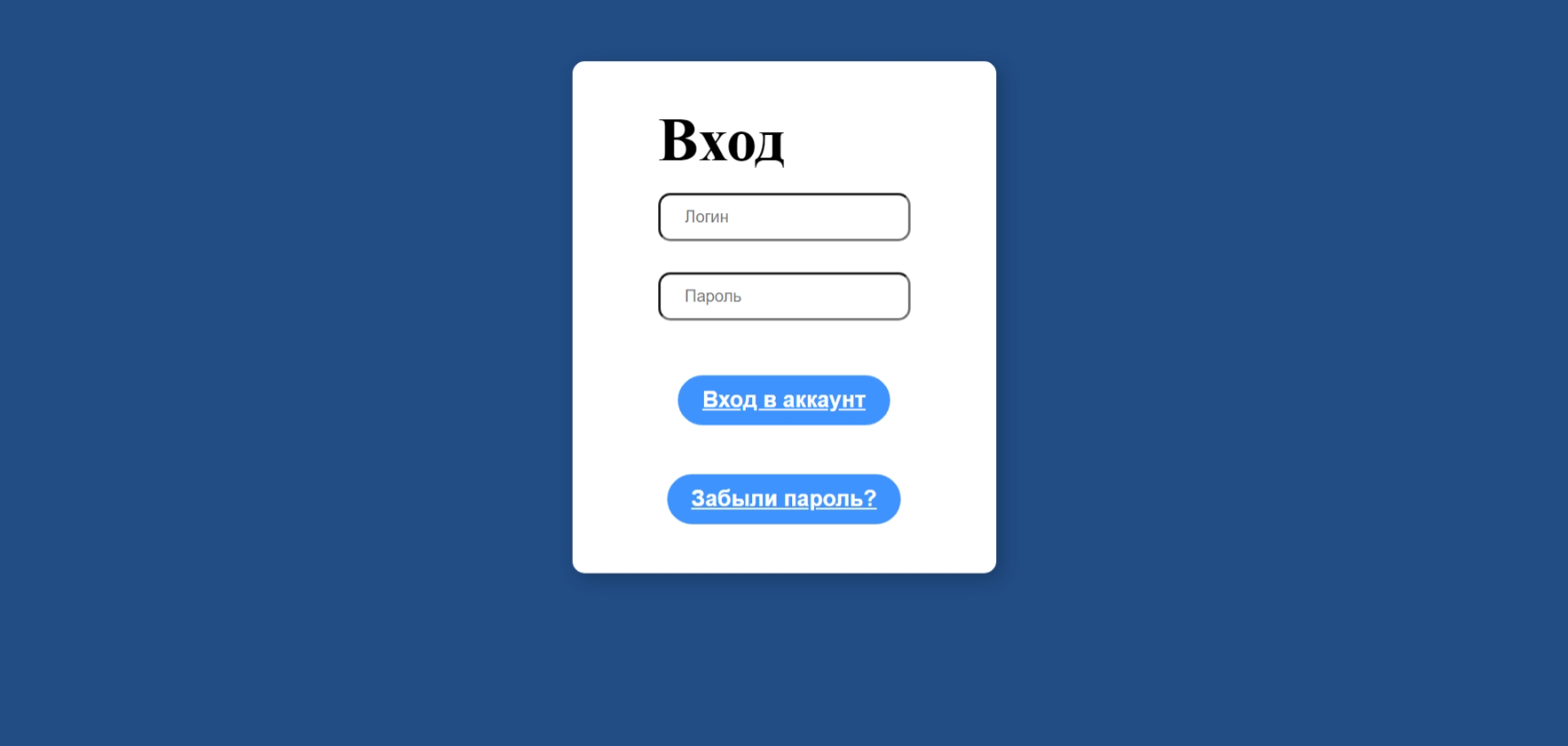


Рисунок 7 – Интерфейс страницы «Вход»

### Интерфейс страницы «Восстановление доступа»

С главной страницы пользователь может перейти на страницу «Восстановление логина/пароля». Интерфейс данной страницы представлен на рисунке 8. HTML-код – в файле PRObatio\backend\general\html\reset\_password.html.

На данной странице используются следующие теги:

<div>, <form>, <h3>

<button> - кнопка «восстановить» должна по нажатию отправлять логин и пароль пользователя на введенную им в <input> почту. После этого с помощью <a> пользователя снова перебрасывает на главную страницу.

Данный алгоритм реализуется на серверной части.

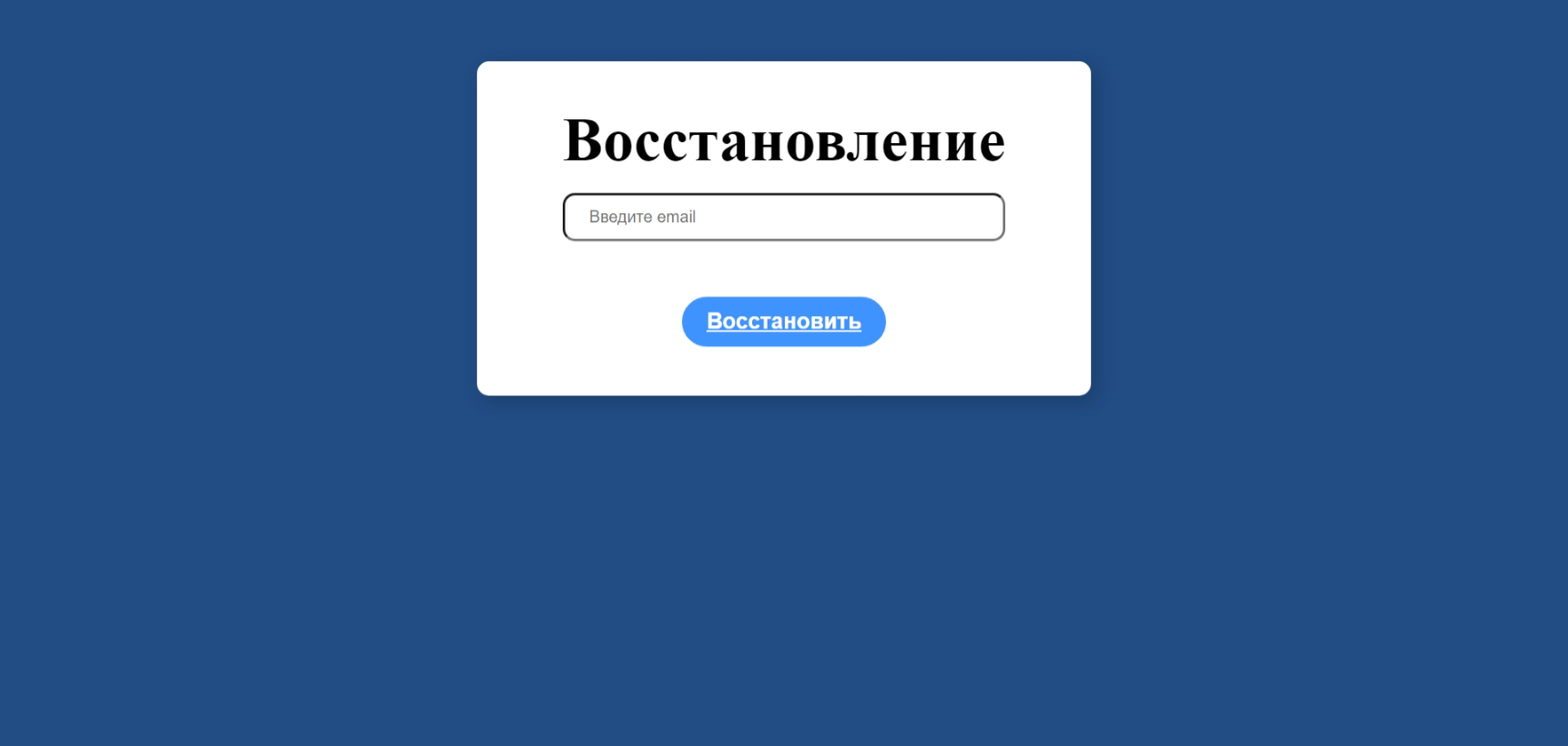


Рисунок 8 - Интерфейс страницы «Восстановление логина/пароля»

### Интерфейс страницы «Личный кабинет (Администратор)»

Так же с главной страницы пользователь может перейти в свой «Личный кабинет». В зависимости от роли пользователя, выбирается страница для личного кабинета. Это решается в серверной части, поэтому ссылок на «личный кабинет» нет на главной странице.

Интерфейс Личного кабинета для пользователя с ролью Администратор представлен на рисунке 9. HTML-код – в файлах PRObatio\backend\views\admin\_account.hbs и PRObatio\backend\views\layouts \admin\_account\_layout.hbs

На данной странице используются следующие теги:

<div>, <form>, <h3>

С помощью <button> <a> «Выйти» пользователь может завершить работу с системой, по ссылке перейдет на начальную страницу PRObatio. Эта функция доступна всем пользователям, независимо от роли.

<p> используется для отображения параграфа текста

Функция «редактировать мои данные» по логике должна быть реализована тегом <button>, однако, так как при её запуске пользователю становятся доступными поля <input> на этой же странице, для того, чтобы страница не перезагружалась каждый раз при использовании данной функции, она реализована с помощью тега <p>, которому добавили реакцию на нажатие – пользователю открываются поля для редактирования, сама кнопка пропадает, а на её месте появляется <button> «Сохранить изменения». Интерфейс страницы после этого представлен на рисунке 10. При нажатии на «Сохранить изменения» в базу данных перезаписываются значения, введенные пользователем, поля для редактирования исчезают, снова появляется функция «Редактировать мои данные» вместе кнопки «Сохранить изменения».

Функция редактирования также доступна всем пользователям, поэтому данный алгоритм представлен в файле PRObatio\general\js\edit\_data.js.

Для указания ссылки на данный файл используется тег <script> [6].

В разделе Действия с помощью <button> <a> «Добавить пользователя» администратор может перейти на страницу «Добавление пользователя».

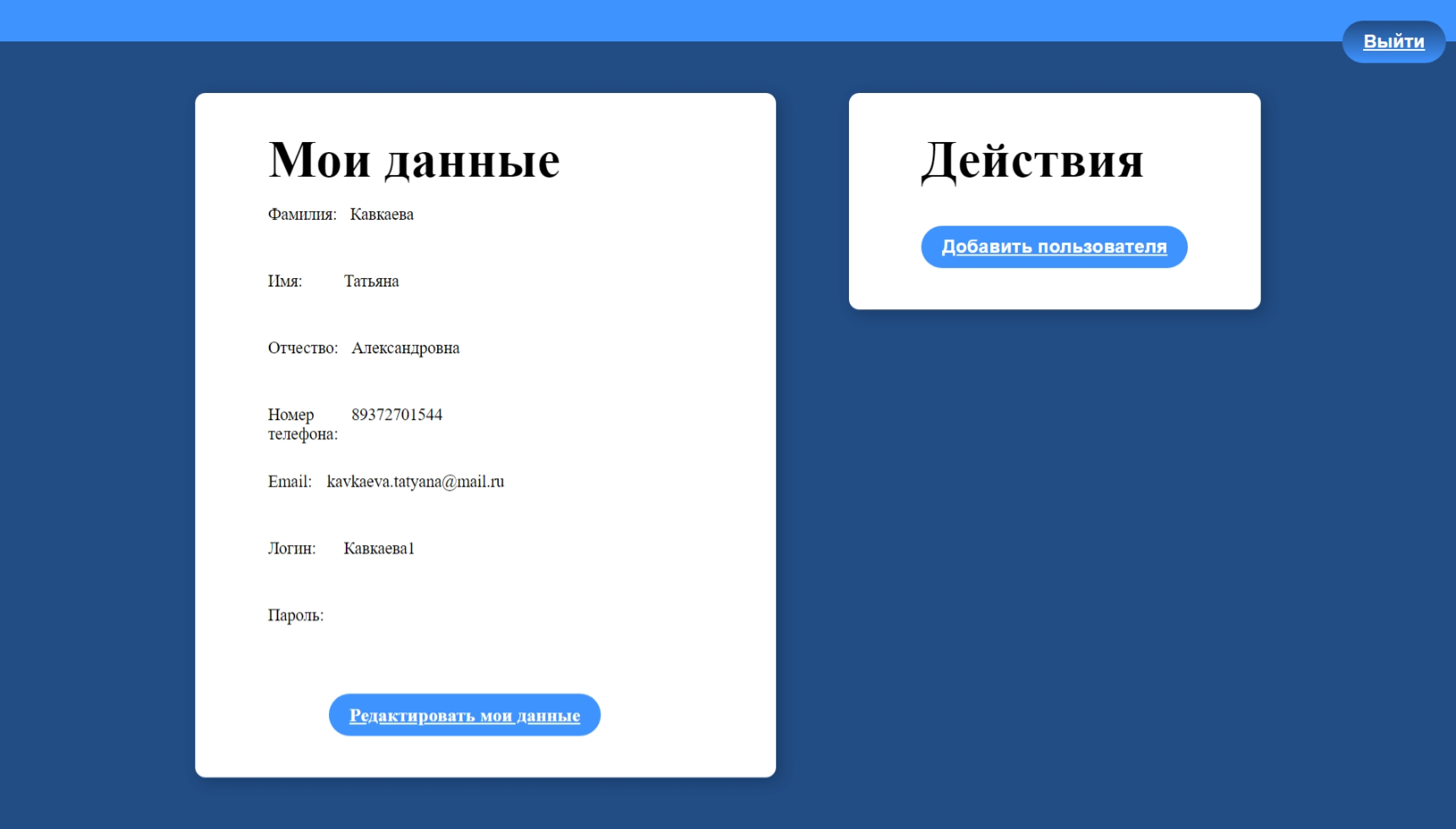


Рисунок 9 - Интерфейс страницы «Личный кабинет (Администратор)»

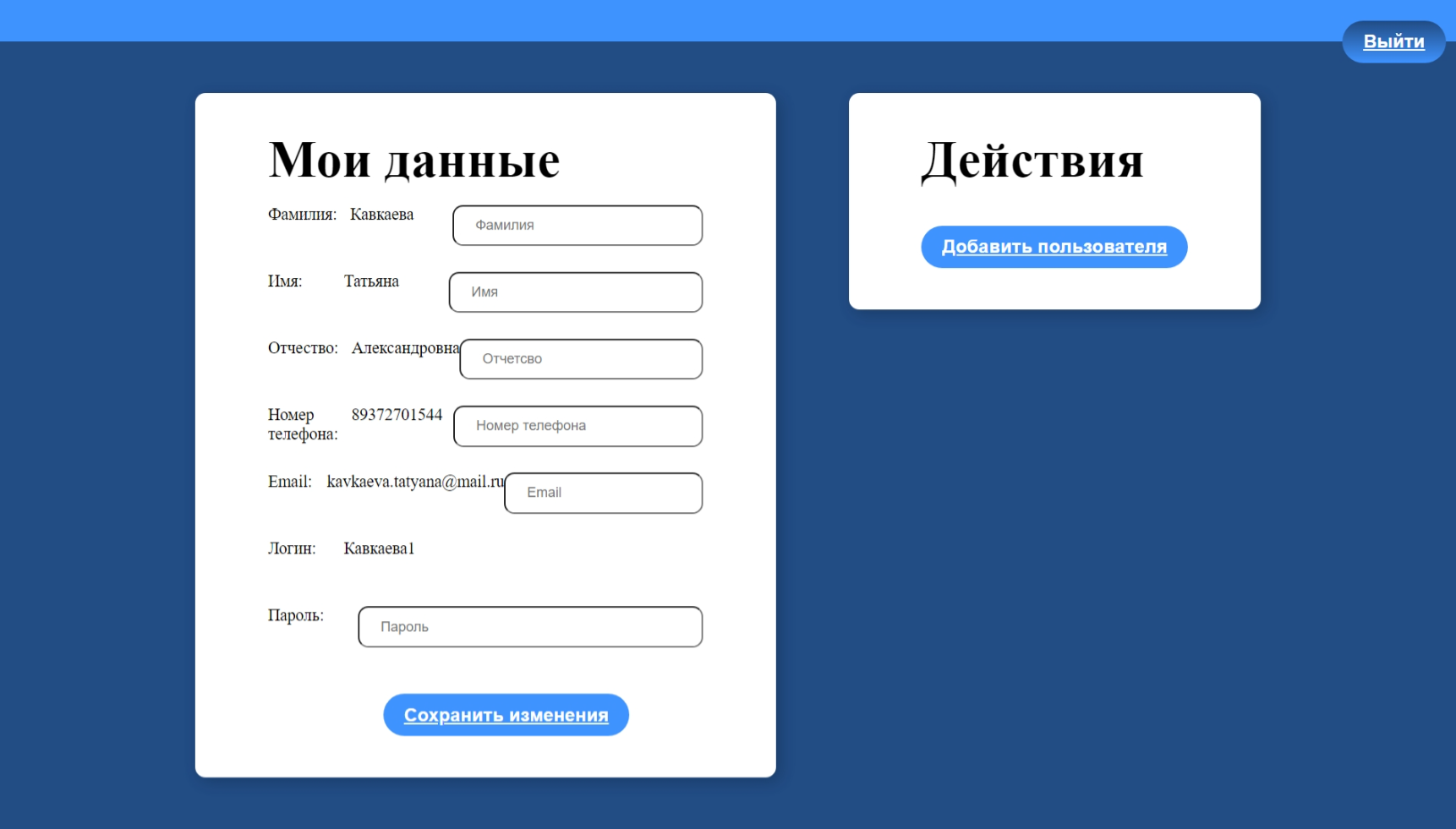


Рисунок 10 - Интерфейс страницы «Личный кабинет (Администратор)» в режиме редактирования

### Интерфейс страницы «Создание нового пользователя

Из своего личного кабинета Администратор может перейти на страницу «Добавление пользователя». Интерфейс данной страницы представлен на рисунках 11-13. Код – в файлах PRObatio\backend\views\layouts\admin\_account\_layout.hbs и PRObatio\backend\ views\admin\_account.hbs

На данной странице используются следующие теги:

<div>, <form>, <h3>, <script>, <p>, <button>

С помощью кнопки «Личный кабинет» пользователь может вернуться на одноименную страницу. Эта функция доступна на всех страницах, к которым есть доступ у авторизованных пользователей, кроме самого «Личного кабинета».

При вводе данных о новом пользователе, Администратор должен выбрать Роль из выпадающего списка, заданного тегом <select>. От выбора роли зависят поля. которые отображаются только для определенной роли. При выборе роли «Преподаватель», становится доступным выбор студентов, прикрепленных к нему – здесь он может выбрать несколько студентов, так как тегу <select> задано свойство multiple (пример показан на рисунке 11). Аналогично для роли Студент – ему открывается список преподавателей, к которым он прикрепляется. Помимо этого, для Студента необходимо ввести номер группы (рисунок 12). Для пользователя с ролью Администратор эти 3 поля недоступны (рисунок 13).

Данный алгоритм приведен в файле PRObatio\backend\js\admin\creating.js.

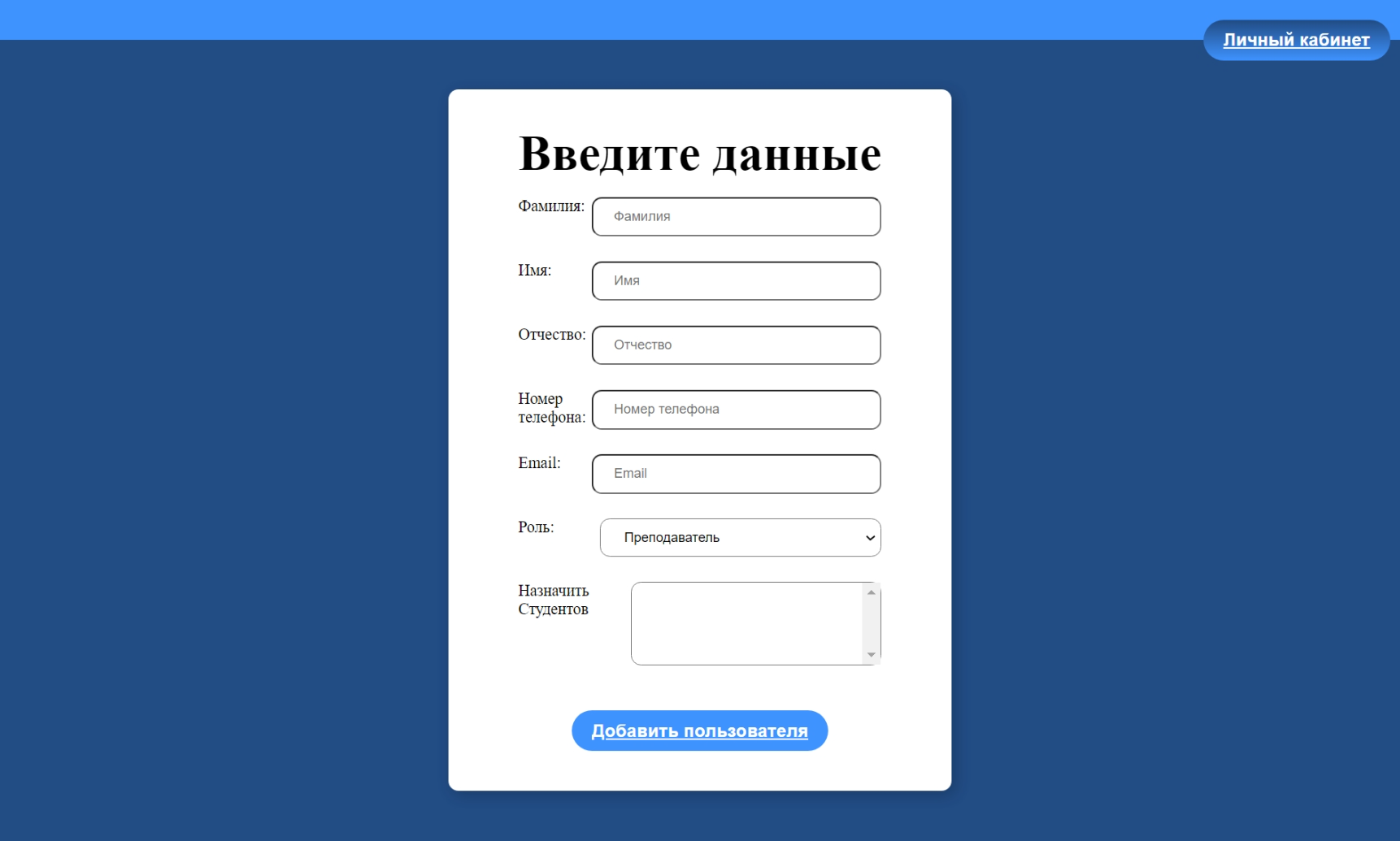


Рисунок 11 - Интерфейс страницы «Создание нового пользователя» для пользователя с ролью «Преподаватель»

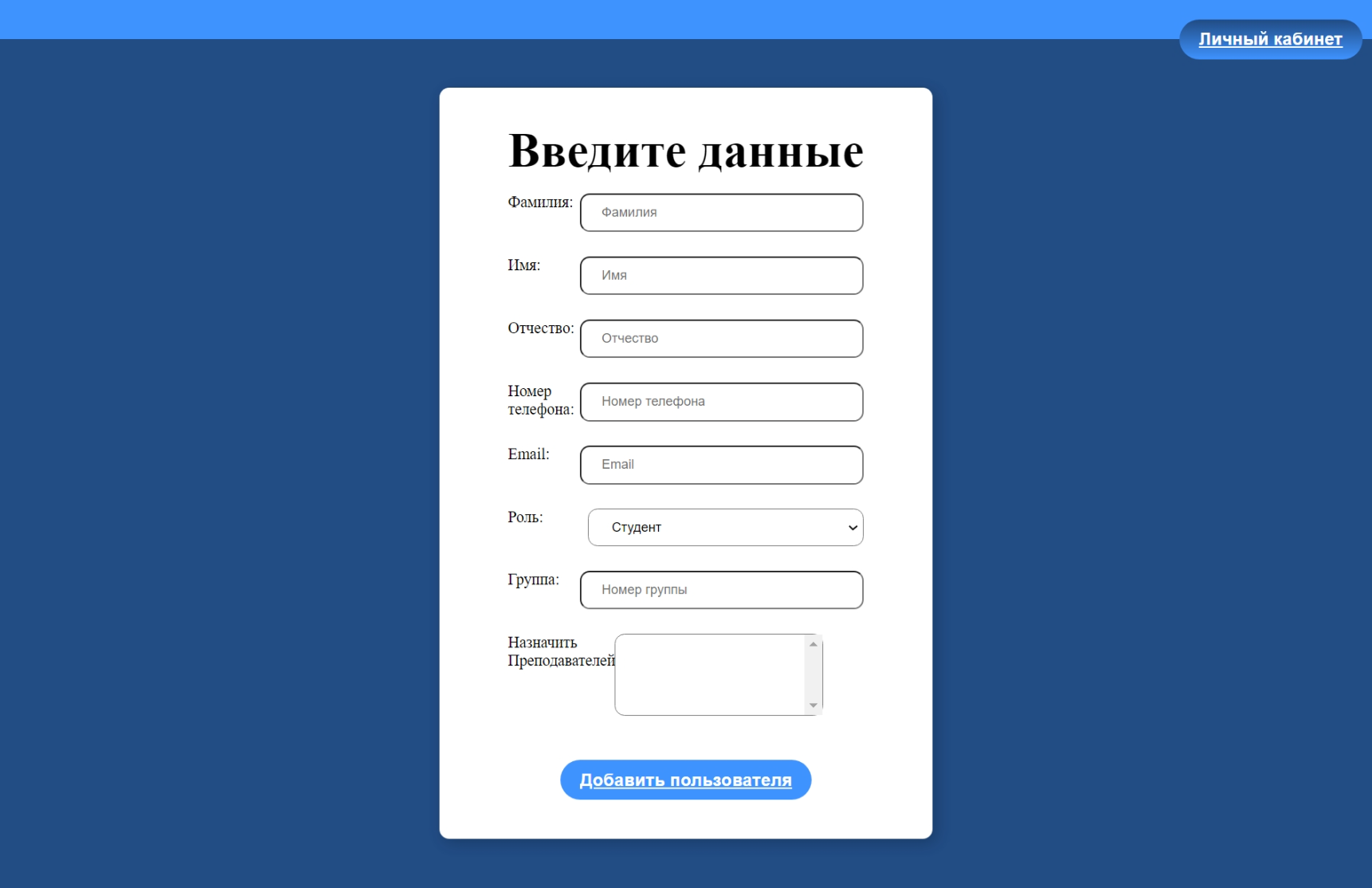


Рисунок 12 - Интерфейс страницы «Создание нового пользователя» для пользователя с ролью «Студент»

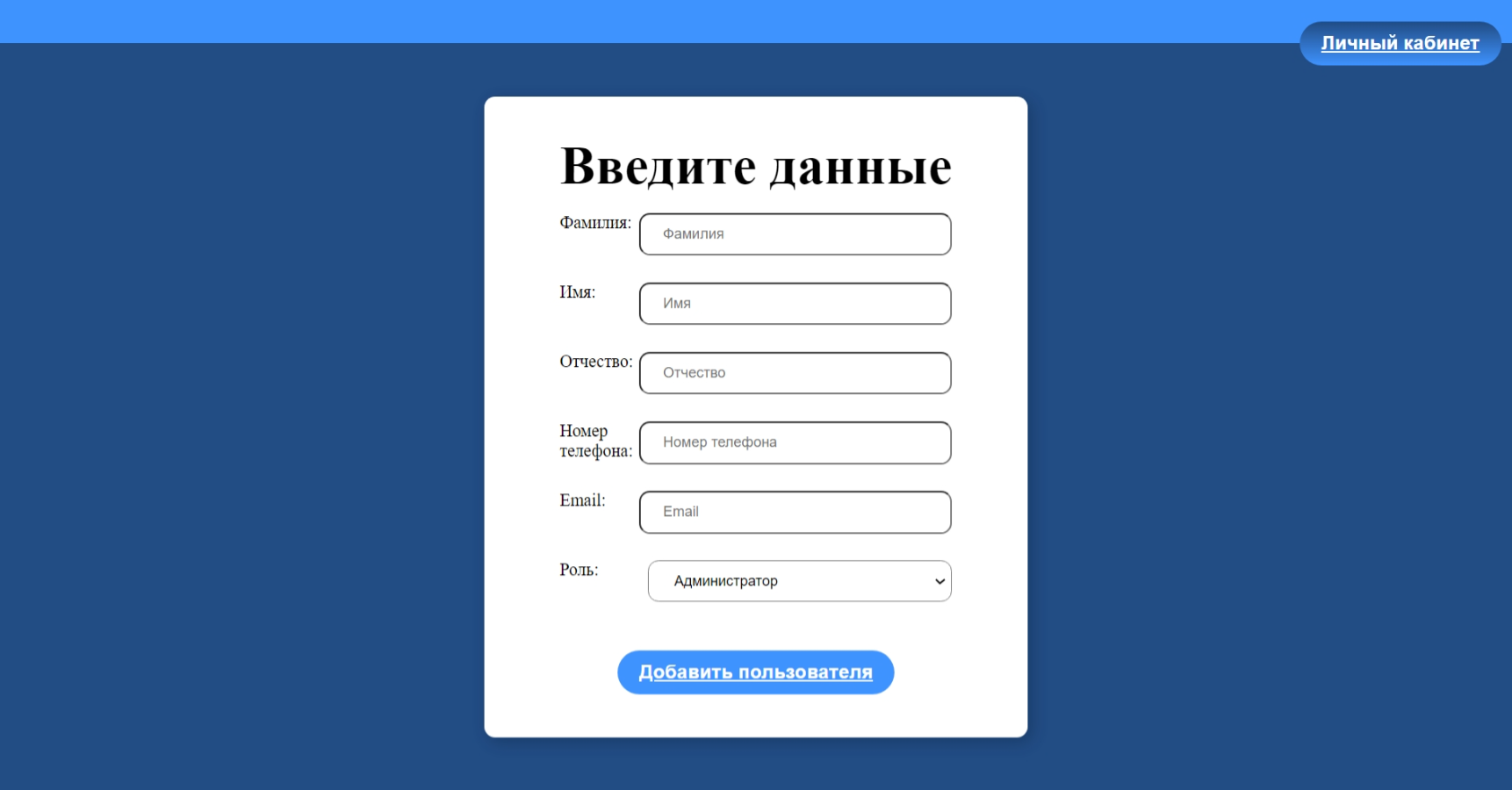


Рисунок 13 - Интерфейс страницы «Создание нового пользователя» для пользователя с ролью «Администратор»

### Интерфейс страницы «Личный кабинет (Преподаватель)»

Интерфейс Личного кабинета для пользователя с ролью Преподаватель представлен на рисунке 14. Код – в файлах PRObatio\backend\views\layouts\teacher\_account\_layout.hbs и PRObatio\backend\views\teacher\_account.hbs.

На данной странице используются следующие теги:

<div>, <form>, <h3>, <script>, <p>

Преподаватель может «Создать тест»: откроется новая страница «Создание теста». В списке созданных тестов, заданном с помощью тега <table>, он может «Пройти тестирование» (ведет на страницу «Тестирование») и просмотреть «Успеваемость» по тесту (ведет на страницу «Успеваемость по тесту»)

У каждого студента из <table> он может просмотреть «Успеваемость» (ведет на страницу «Успеваемость студента»).

<hr> используется для рисования линии, которая визуально разделяет разделы.

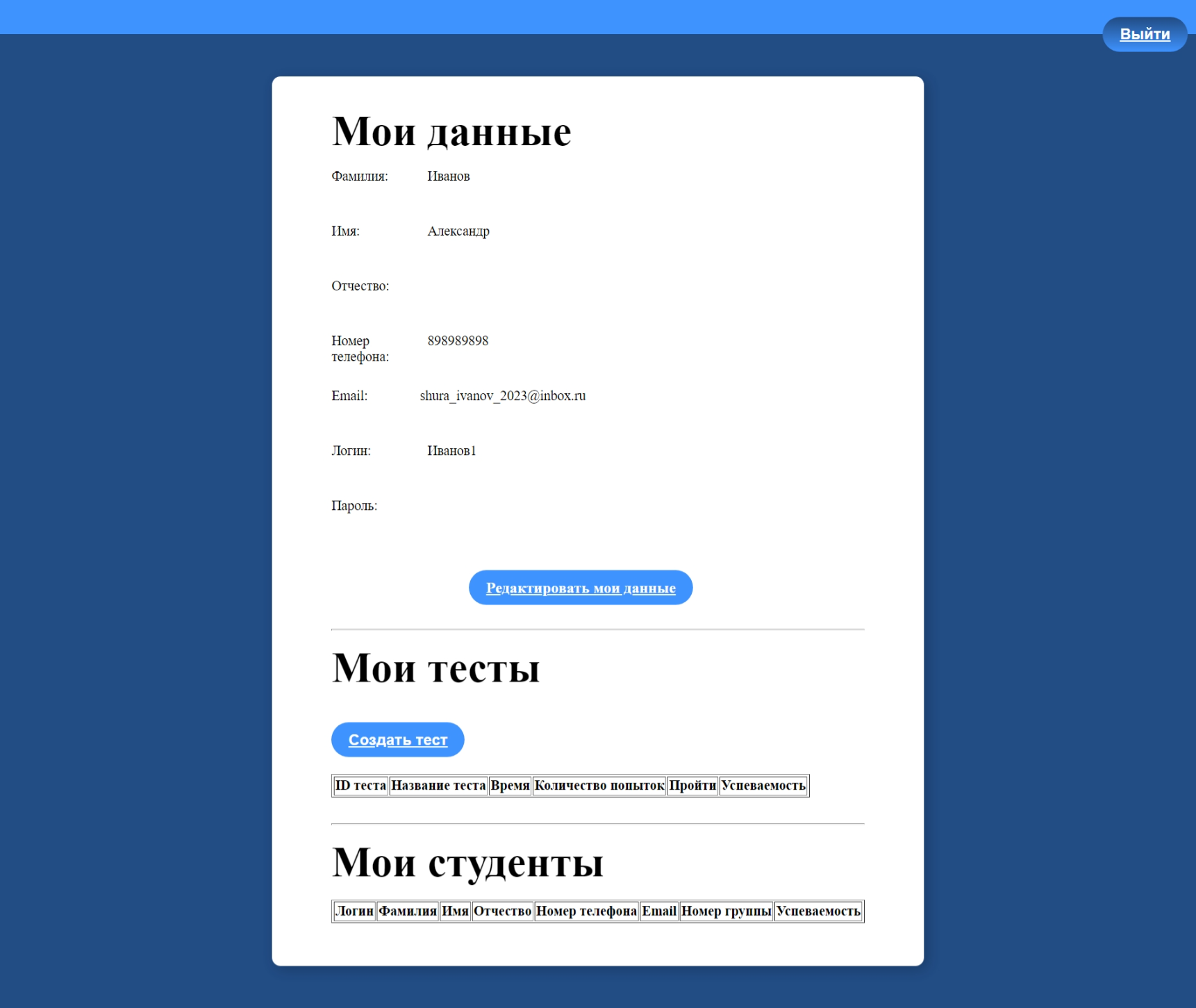


Рисунок 14 - Интерфейс страницы «Личный кабинет (Преподаватель)»

### Интерфейс страницы «Создание теста»

Интерфейс страницы «Создание теста» представлен на рисунке 15. Код – в файлах PRObatio\backend\views\create\_test.hbs и PRObatio\backend\views\layouts\create\_test\_layout.hbs

На данной странице используются следующие теги:

<div>, <form>, <h3>, <script>, <p>, <select>

<ol> - нумерованный список, используется для структурирования перчня вопросов

<li> - элемент списка

В данном случае используются особые input – они имеют тип “checkbox”. В Этапе 1 они необходимы для определения правильности вариантов ответов, а в Этапе 2 – для определения наличия ограничений: если оно существует, появляются поля для их ввода. Данный алгоритм представлен в файле PRObatio\backend\js\teacher\create\_test.js

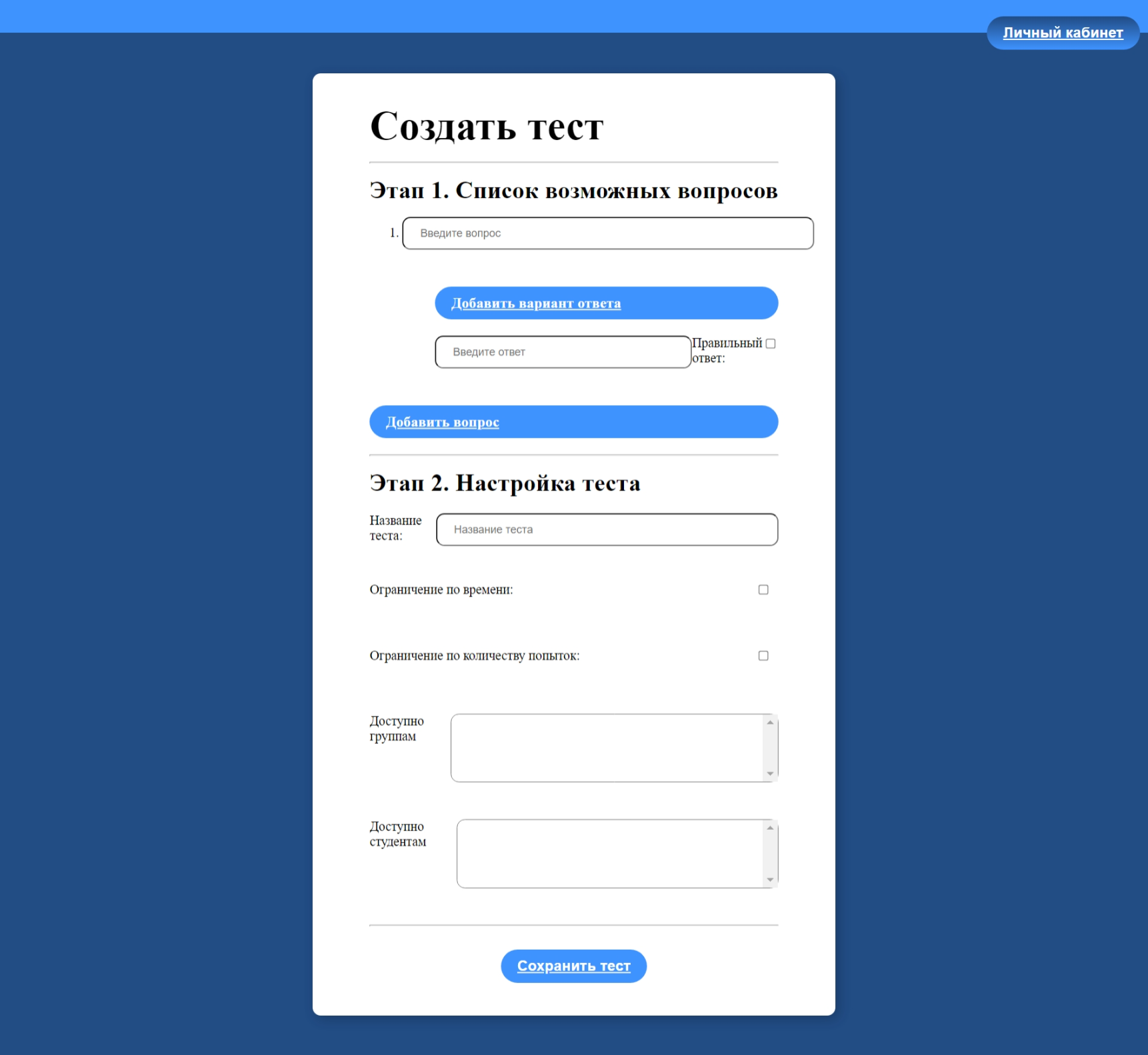


Рисунок 15 - Интерфейс страницы «Создание теста»

### Интерфейс страницы «Тестирование»

Интерфейс страницы «Тестирование» представлен на рисунке 16. Код – в файлах PRObatio\backend\views\layouts\testing\_layout.hbs и PRObatio\backend\views\testing.hbs.

На данной странице используются следующие теги:

<div>, <form>, <h3>, <p>, <ol> ,<li>, <input type=”checkbox”>, <script>

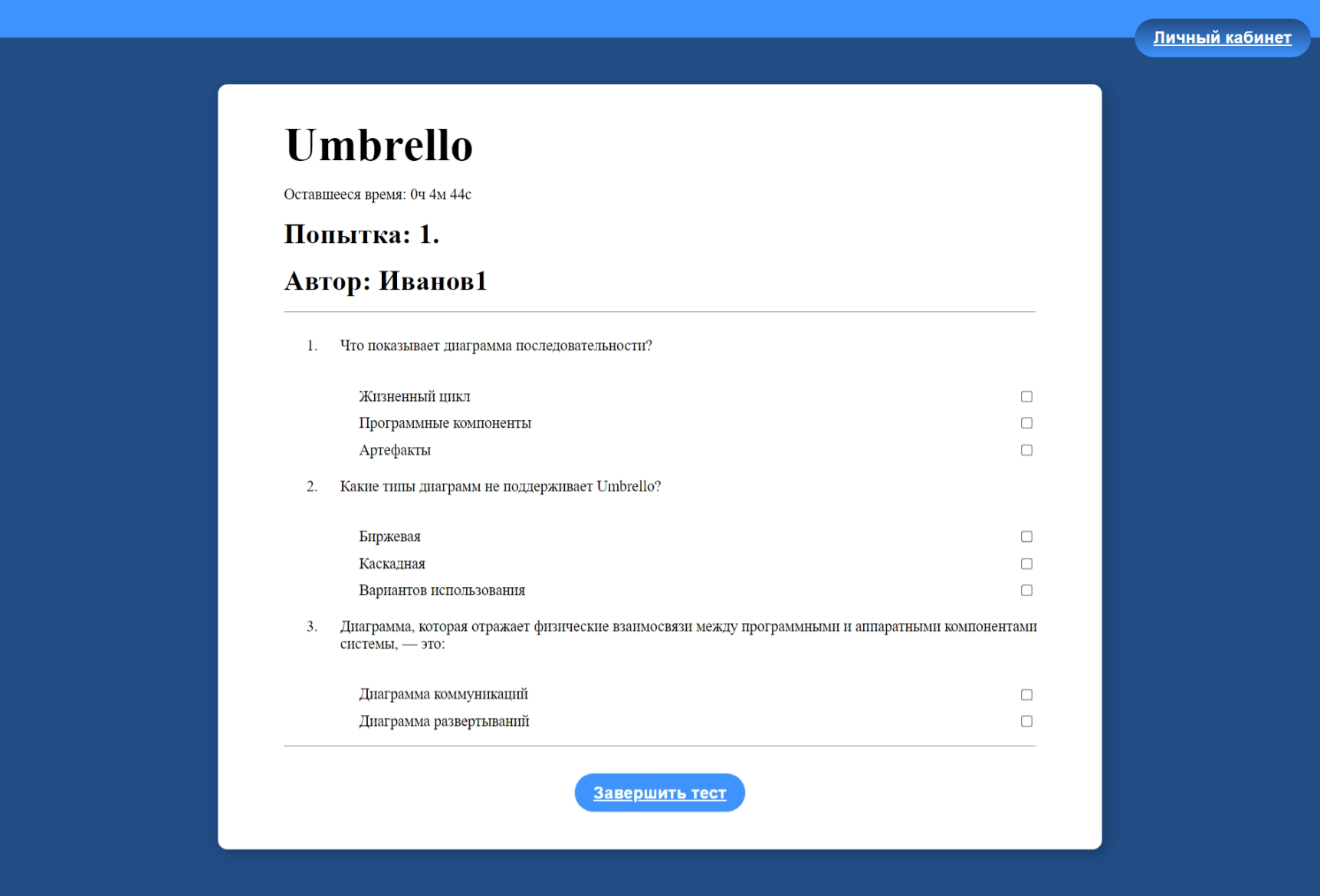


Рисунок 16 - Интерфейс страницы «Тестирование»

### Интерфейс страницы «Результат»

После того, как тест завершен, пользователь перенаправляется на страницу, на которой показан его результат. Интерфейс данной страницы представлен на рисунке 17. Код – в файлах PRObatio\backend\views\result.hbs и PRObatio\backend\views\layouts\result\_layout.hbs.

На данной странице используются следующие теги:

<div>, <h3>,<a>, <p>

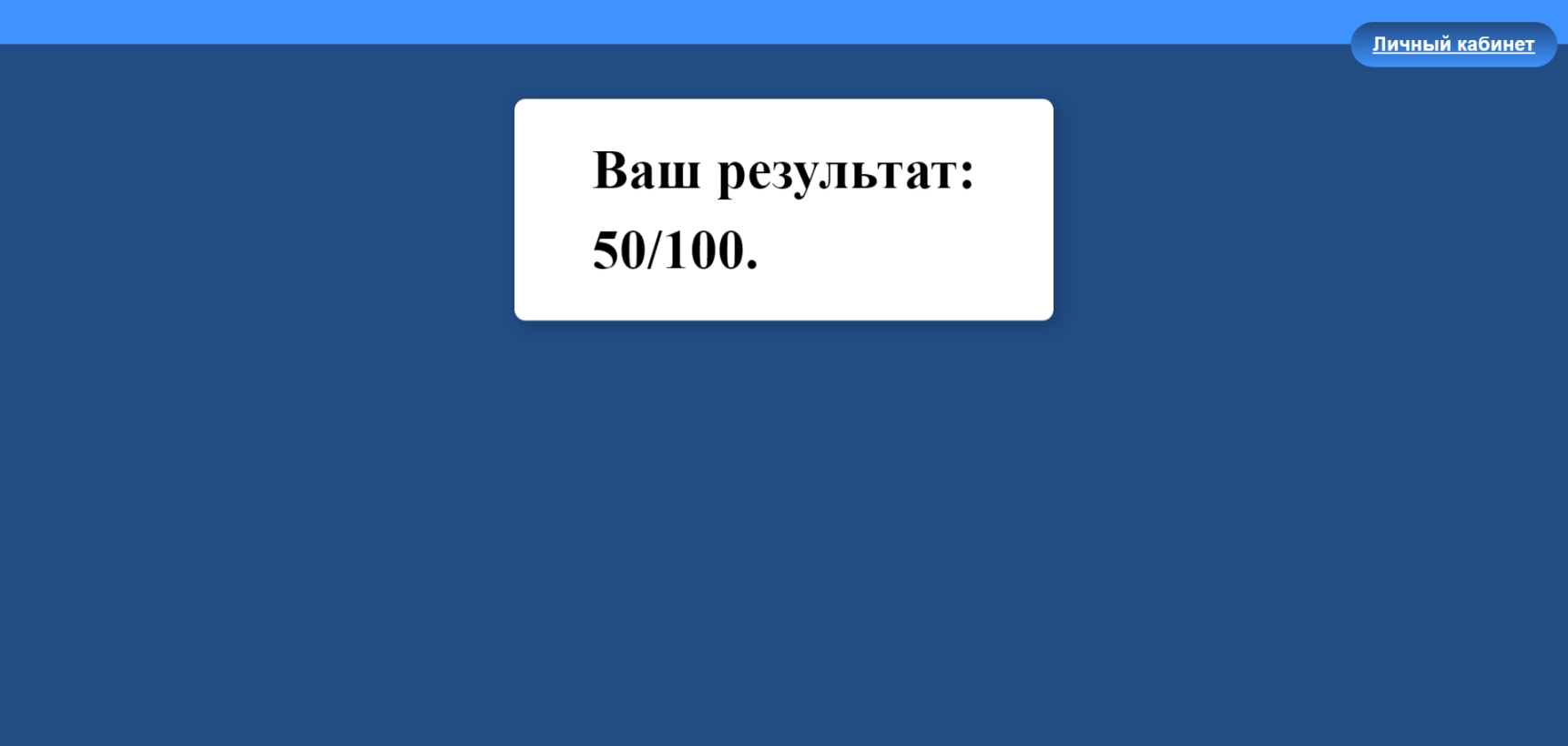


Рисунок 17 - Интерфейс страницы «Результат»

### Интерфейс страницы «Успеваемость по тесту»

Интерфейс страницы «Успеваемость по тесту» представлен на рисунке 18. Код – в файлах PRObatio\backend\views\test\_progress.hbs и PRObatio\backend\views\layouts\test\_progress\_layout.hbs

На данной странице используются следующие теги:

<div>, <h3>,<a>

Пользователь может просмотреть успеваемость по тесту в виде списка студентов, сдававших этот тест, отображенного тегом <table>, а также просмотреть «Успеваемость» конкретного студента (ведет на страницу «Успеваемость студента»).

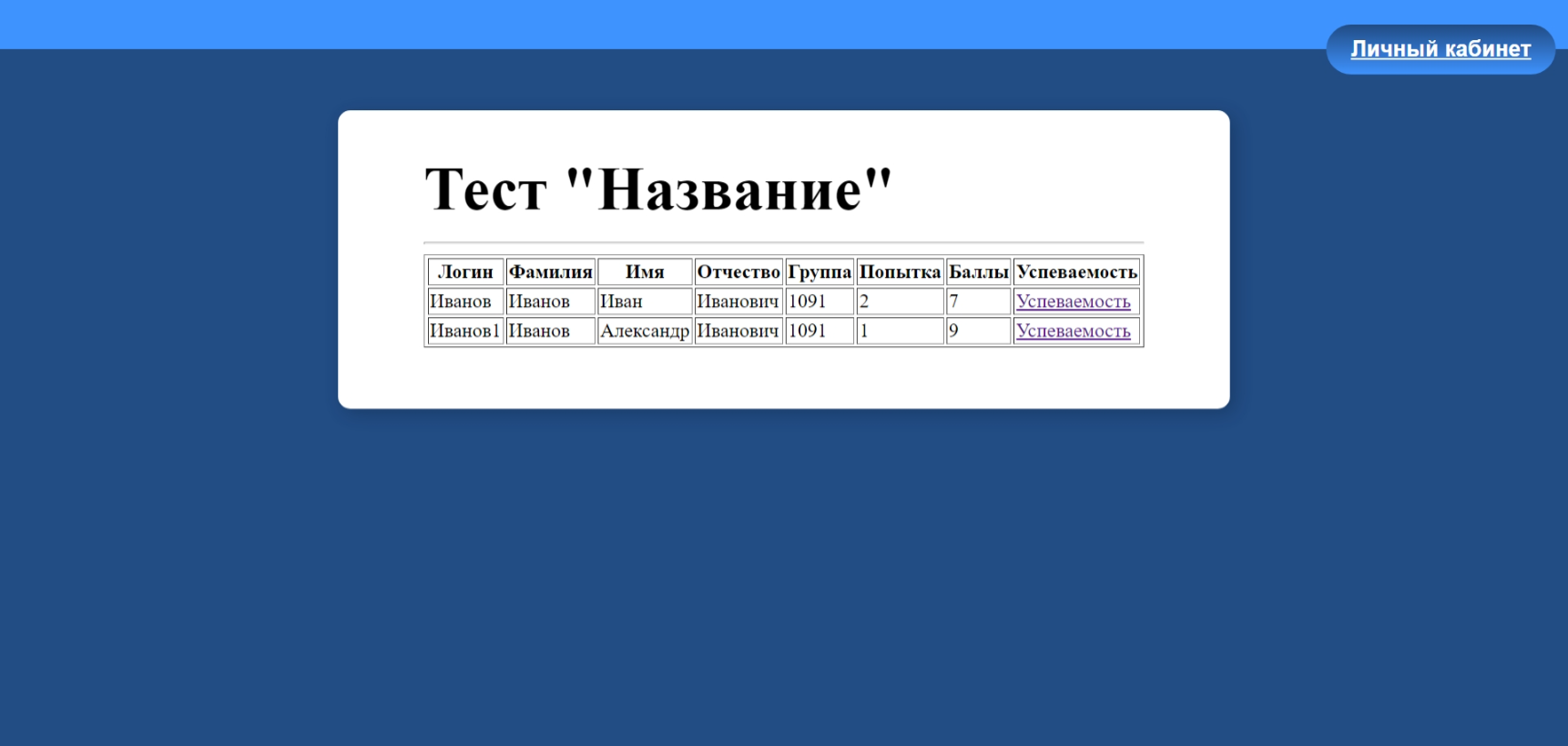


Рисунок 18 - Интерфейс страницы «Успеваемость по тесту»

### Интерфейс страницы «Успеваемость студента»

Интерфейс страницы «Успеваемость по тесту» представлен на рисунке 19. Код – в файлах PRObatio\backend\views\student\_progress.hbs и PRObatio\backend\views\layouts\student\_progress\_layout.hbs

На данной странице используются следующие теги:

<div>, <h3>, <p>, <table>.

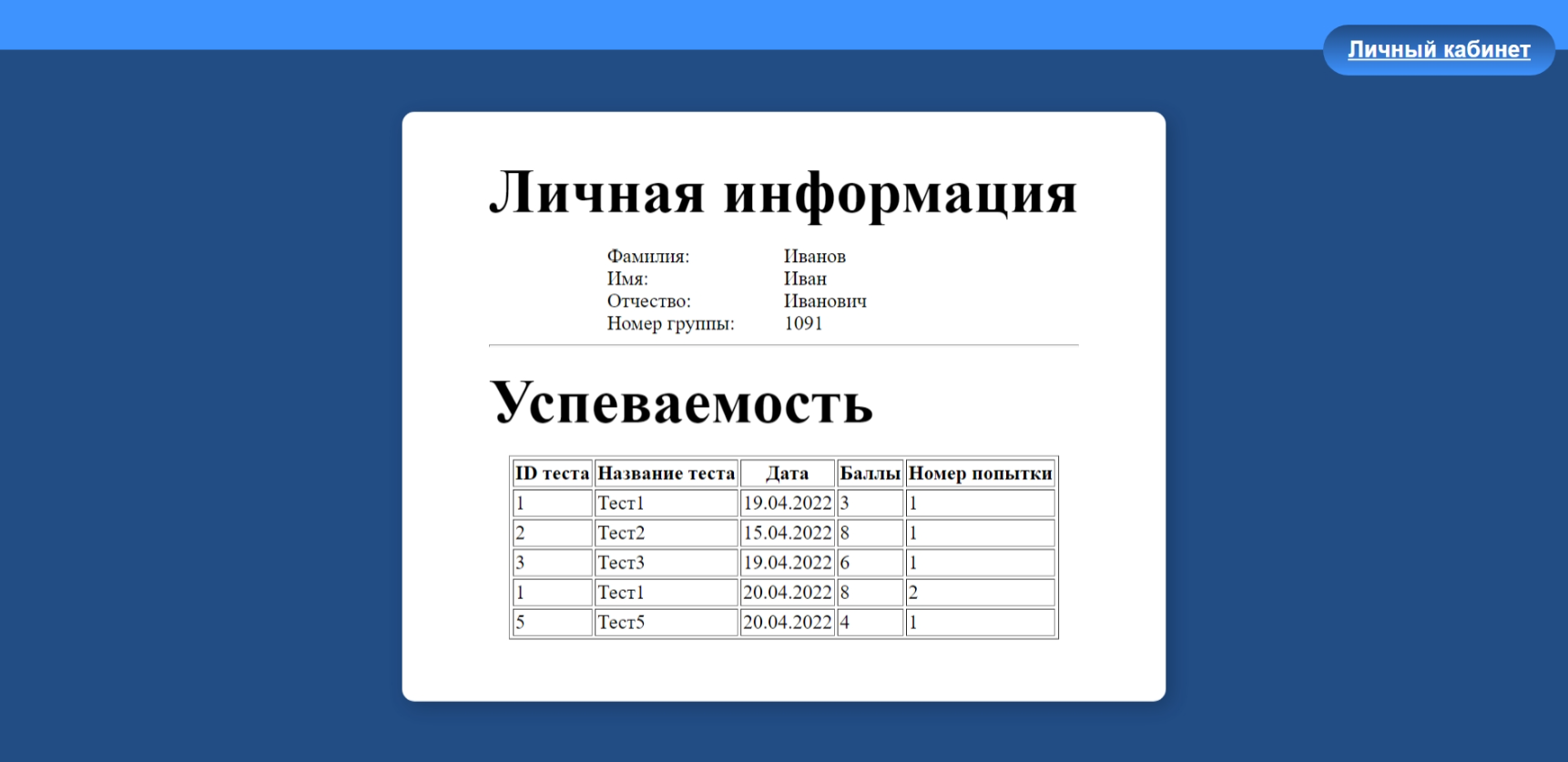


Рисунок 19 - Интерфейс страницы «Успеваемость студента»

### Интерфейс страницы «Личный кабинет (Студент)»

Интерфейс Личного кабинета для пользователя с ролью Студент представлен на рисунке 20. Код – в файлах PRObatio\backend\views\student\_account.hbs и PRObatio\backend\views\layouts\student\_account\_layout.hbs.

На данной странице используются следующие теги:

<div>, <form>, <h3>, <script>, <p>, <hr>

Студент может просматривать доступные ему тесты с помощью «Тестирования» (открывается список тестов, ранее скрытый; по повторному клику список закрывается – это алгоритм реализован в файле PRObatio\backend\js\student\secret\_elements.js) и проходить их с помощью «Пройти тестирование» (ведет на страницу «Тестирование»). Так же у Студента есть возможность просмотреть свою «Успеваемость» (ведет на страницу «Успеваемость студента»).

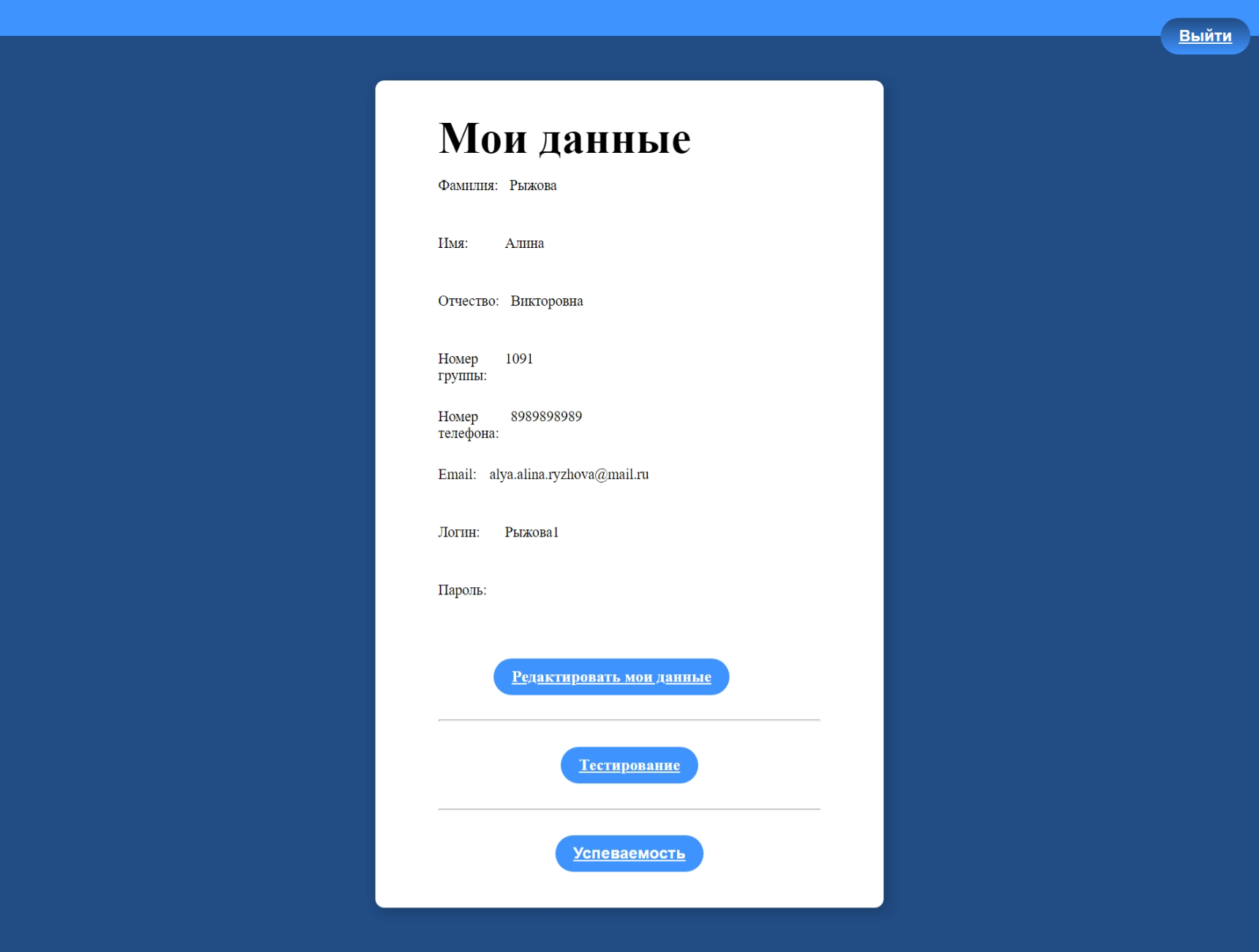


Рисунок 20 - Интерфейс страницы «Личный кабинет (Студент)»

## 2.2 Разработка базы данных

По спроектированной ER-модели была составлена база данных. Полный её код представлен в файле PRObatio\db\database.txt.

Так же в данном файле приведен код создания начального пользователя – Администратора, который и будет создавать аккаунты другим пользователям. Этот же код приведен на рисунке 21.

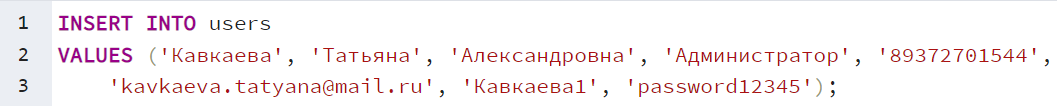


Рисунок 21 – Добавление начального пользователя с ролью Администратор

Так как начальный пользователь создан в sql-коде, а не в самой системе, письмо с логином и паролем не придет ему на почту.

## 2.3 Разработка серверной части системы

Для запуска сервера необходимо сначала создать каталог, в котором будет храниться сервер, затем запустить в нём команду npm init. После этого установить модуль express как зависимость.

Далее в созданном каталоге необходимо создать файл, в котором будет находиться сервер и разместить в нём код, приведенный на листинге 2.

const express = require('express');

const exphbs = require('express-handlebars')

const cookieParser = require('cookie-parser')

const routes = require('./route')

const exp = express();

const host = '127.0.0.1';

const port = process.env.PORT || 7000;

exp.use(express.static(\_\_dirname))

exp.use(express.json())

exp.use(cookieParser())

exp.use(express.urlencoded({ extended: true }))

// шаблонизатор

const hbs = exphbs.create({

  defaultLayout:'create\_new\_user',

  extname:'hbs'

})

exp.engine('hbs',hbs.engine)

exp.set('view engine', 'hbs')

exp.set('views', 'views')

exp.use(routes)

const start = async () => {

  try {

    //запуск сервера

    exp.listen(port, host , () => console.log(`Server listens http://${host}:${port}`))

  }catch(er) {

    console.log(er)

  }

}

start()

Листинг 2 – Создание сервера

Все обращения к северу обрабатываются в файле PRObatio\backend\route.js

Для того, чтобы получить информацию с сервера используются GET запросы, чтобы отправить её на сервер – POST, он обычно отправляется с наступлением какого-то события. Для обращения к серверу используется url(Унифицированный указатель ресурса — система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса.)

Рассмотрим, какие запросы используются в PRObatio.

Чтобы использовать функции, находящиеся в 1 файле, в другом файле

В верх файла вынесены все импортирования (листинг 3) и локальные переменные.

const {Router} = require('express')

const router = Router()

const bcrypt = require('bcryptjs')

const jwt = require('jsonwebtoken');

const Math=require('math')

const { Pool } = require('pg');

const { toArray} = require('list');

const {reset, send\_info} = require('./nodemailer');

Листинг 3 – Импортирование

Для того, чтобы не устанавливать соединение с базой данных каждый раз, когда это необходимо, можно вызвать подключение 1 раз, и дальше работать с пулами (листинг 4).

// подключение к бд

const pool = new Pool({

  host: '127.0.0.1',

  port: 5432,

  database: 'PRObatio',

  user: 'postgres',

  password: 'postgres',

});

Листинг 4 – Подключение к базе данных

Автоматическая генерация пароля (листинг 5) возвращает пароль, состоящий из строчных и заглавных английских букв и цифр. Она используется при создании нового пользователя, а также при восстановлении доступа к уже существующему аккаунту, так как в базе данных не хранятся пароли в явном виде.

// автоматическая генерация пароля

function gen\_password(){

            var password = "";

            var length=Math.floor(Math.random()\*(20-8+1))

            var symbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

            for (var i = 0; i < length; i++){

                password += symbols.charAt(Math.floor(Math.random() \* symbols.length));

            }

            return password;

        }

Листинг 5 – Автоматическая генерация пароля

Токен (листинг 6) используется для идентификации пользователя в системе. Роль пользователя входит в состав токена для разграничения возможностей в соответствии с ролью.

//генерация токена

const generateAccessToken = (login, user\_role) => {

    const payload = {

        login,

        user\_role

    }

    return jwt.sign(payload,'SECRET\_KEY', {})

}

//проверка токена

 cookieJwtAuth = (req,res,next) => {

        const token = req.cookies.token

        try {

            const user = jwt.verify(token,'SECRET\_KEY')

            req.user = user

            next()

        } catch(er) {

            res.clearCookie('token')

            return res.redirect('/')

        }

    }

Листинг 6 -Токен

Некоторые страницы могут как получать информацию с сервера, так и отправлять. Для удобства используется запись router.route(url), а ниже располагают .get() и .post(), как на листинге 7. На нём представлены запросы, которые приходят от клиента и уходят к нему, когда он находится на странице входа. GET – формируется страница «Вход»; POST – идет идентификация и перенаправление в нужное место. При особых условиях перенаправление происходит по url=’/personal\_account', запросы к нему представлены на листинге10.

//страница входа(Главная)

router.route('/')

    .get(function(req, res) {

        res.sendFile(\_\_dirname+'/html/general/index.html')

    })

    .post(async function(req, res){

        const {login,password} = req.body

        pool.query(`SELECT user\_role, user\_password FROM users where login='${login}'`,(q\_err,q\_res) => {

            if(q\_err) console.log(q\_err)

            else {

                const validPassword = bcrypt.compareSync(password,q\_res.rows[0]['user\_password']) //сравниваем пароли

            if (validPassword) {

                // if (password==q\_res.rows[0]['user\_password']){

                const token = generateAccessToken(login, q\_res.rows[0]['user\_role']) //создаем токен

            res.cookie('token',token,{  //сохраняем его в cookie

                httpOnly: true

            })

                return res.redirect('/personal\_account')

            }

            else{ return res.redirect('/')}

        }

    })}

    )

Листинг 7 – Обработка Входа

На листинге 8 представлена обработка Восстановления доступа. GET – формируется страница «Восстановление доступа»; POST – ведётся поиск введённого адреса электронной почты в базе данных, при совпадении логин и новый пароль отправляются на этот адрес. Данная функция реализована в файле PRObatio\backend\nodemailer.js (в нём хранятся все функции, связанные с отправлением почты).

//восстановление доступа

router.route('/reset')

    .get(function(req, res) {

        res.sendFile(\_\_dirname+'/html/general/reset\_password.html')

    })

    .post(function(req, res) {

        const email=req.body

        password=gen\_password()

        hash\_password=bcrypt.hashSync(password, 6)

        pool.connect(function(error, client, done) {

            let sql=`SELECT login from users WHERE email=${email}`

            let sql2 = `UPDATE users SET password=${hash\_password} WHERE email=${email}`

            client.query(sql).then((response)=>{client.query(sql2, [], function(err, resp) {

                done();

                if (response.rows.length!=0){

                    reset(email, response.rows[0]['login'], password)}

                res.redirect('/')

                })})

    })

    })

Листинг 8 – Обработка Восстановления доступа

На листинге 9 представлена обработка Выхода. GET – очищается токен, формируется главная страница.

//выход

router.route('/exit')

    .get(function(req, res){

        res.clearCookie('token')

        return res.redirect('/')

    })

Листинг 9 – Обработка Выхода

Обработка маршрутизации по личным кабинетам, представленная на листинге 10, читает роль из токена и перенаправляет на нужную страницу.

//маршрутизация по личным кабинетам в зависимости от роли

router.route('/personal\_account')

    .get(function (req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

            if (req.cookies.token){

                let role= payload.user\_role

            if (role=='Администратор'){

                return res.redirect('/admin\_account')

            }

            if (role=='Преподаватель'){

                return res.redirect('/teacher\_account')

            }

            if (role=='Студент'){

                return res.redirect('/student\_account')

            }

    }})

Листинг 10 – Обработка маршрутизации по личным кабинетам

На листинге 11 представлена обработка запросов из Личного кабинета администратора. GET – формируется страница «Личный кабинет»; POST – осуществляется перезапись в бд измененных данных. Обработка всех Личных кабинетов действует по аналогичной схеме (листинги 13 и 18).

//личный кабинет администратора

router.route('/admin\_account')

    .get(cookieJwtAuth, function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                // личная информация

                let sql = `SELECT \* FROM users WHERE login='${payload.login}'`

                client.query(sql).then((response)=>{

                        done();

                    res.render('admin\_account',{

                    layout: 'admin\_account\_layout',

                    user:response.rows[0]

                });

                    })})

    }

    })

    //сохранение изменений

    .post(function(req, res){

        var {new\_lastname, new\_firstname,new\_middlename, new\_phone, new\_email, new\_password}=req.body

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        let fields=[]

        if (new\_lastname){

            fields.push(`lastname = '${new\_lastname}'`)

        }

        if (new\_firstname){

            fields.push(`firstname = '${new\_firstname}'`)

        }

        if (new\_middlename){

            fields.push(`middlename = '${new\_middlename}'`)

        }

        if (new\_phone){

            fields.push(`phone = '${new\_phone}'`)

        }

        if (new\_email){

            fields.push(`email = '${new\_email}'`)

        }

        if (new\_password){

            fields.push(`password = '${bcrypt.hashSync(new\_password, 6)}'`)

        }

        if (fields.length!=0) {

            let sql = `update users set ${fields.join(', ')} where login='${payload.login}'`

            pool.query(sql, (request, response) =>{

                res.redirect('/admin\_account')})}

        else {

            res.redirect('/admin\_account')

        }

    })

Листинг 11 – Обработка запросов из Личного кабинета администратора

На листинге 12 представлена обработка Создания нового пользователя. Во время создания пароля используется функция, представленная на листинге 5. Созданный пароль и логин отправляются на указанную почту – данная функция реализована в файле PRObatio\backend\nodemailer.js

// добавление нового пользователя

router.route('/creating')

    .get(function (req, res){

        pool.connect(function(error, client, done) {

            // список существующих студентов и преподавателей

            let sqlStr = `SELECT \* FROM users where user\_role='Преподаватель'`

            let sqlStr2 = `SELECT \* FROM users where user\_role='Студент'`

            client.query(sqlStr).then((response)=>{client.query(sqlStr2, [], function(err, resp) {

                    done();

                res.render('creating',{

                layout: 'create\_new\_user',

                list\_of\_students:resp.rows,

                list\_of\_teachers:response.rows

            });

                })})

    })})

    .post(async function(req, res){

        // запись данных

        var {lastname,username,middlename,phone,email,role,group, list\_of\_students, list\_of\_teachers}=req.body

        // создание пароля

        password=gen\_password()

        hash\_password=bcrypt.hashSync(password, 6)

        // отправление новому пользователю учетных данных

        pool.connect(function(error, client, done) {

        let sql\_login=`select '${lastname}' || ((SELECT count(\*) from users where login LIKE '${lastname}%')+1) as login`

        client.query(sql\_login).then((response)=>{

            done();

            send\_info(email, response.rows[0]['login'], password);

        }) })

        // запись пользователя в бд(Администратор)

        if (role=='Администратор'){

            pool.query(`INSERT INTO users(lastname,firstname,middlename,user\_role,phone,email,login,user\_password)

            values ('${lastname}','${username}','${middlename}','${role}', '${phone}','${email}',

            (select '${lastname}' || ((SELECT count(\*) from users where login LIKE '${lastname}%')+1)

            as "Login"),'${hash\_password}')`,(q\_err,q\_res) => {

            if(q\_err) return (q\_err)

            res.redirect('/creating')

        })}

        // запись пользователя в бд(Преподаватель)

        if (role=='Преподаватель'){

            // считывание списка прикрепленных студентов

            if (!list\_of\_students)

                {var students=[]

            } else

            {

                if (list\_of\_students.length<2)

                {var students=toArray(list\_of\_students)}

                else {var students=list\_of\_students}

            }

            // запись в бд

            pool.query(`INSERT INTO users(lastname,firstname,middlename,user\_role,phone,email,login,user\_password, list\_of\_students)

             values ('${lastname}','${username}','${middlename}','${role}', '${phone}','${email}',

             (select '${lastname}' || (SELECT count(\*)+1 from users where login LIKE '${lastname}%')

             as "Login"),'${hash\_password}', '{${students}}')`,(q\_err,q\_res) => {

            if(q\_err) return (q\_err)

            // перенаправление на эту же страницу

            res.redirect('/creating')})}

        // запись пользователя в бд(Студент)

        if (role=='Студент'){

            // считывание списка прикрепленных преподавателей

            // добавление к записи каждого преподавателя данного Студента

            if (list\_of\_teachers){

               for (let i=0;i<list\_of\_teachers.length;i++){

                let teacher=list\_of\_teachers[i]

                console.log(teacher)

                pool.query(`update users set list\_of\_students=

                array\_append((select list\_of\_students from users where login='${teacher}'),

                (select '${lastname}' || (SELECT count(\*) from users where login LIKE '${lastname}%')+1)::varchar(255))

                where login='${teacher}'`,(q\_err,q\_res) => {

                    if(q\_err) return (q\_err)

                })}

            }

            // запись в бд

            pool.query(`INSERT INTO users(lastname,firstname,middlename,user\_role,phone,email,login,user\_password,

                user\_group) values ('${lastname}','${username}','${middlename}','${role}', '${phone}','${email}',

                (select '${lastname}' || ((SELECT count(\*) from users where login LIKE '${lastname}%')+1)

                as "Login"),'${hash\_password}','${group}')`,(q\_err,q\_res) => {

                    if(q\_err) return (q\_err)

                    res.redirect('/creating')})}

                })

  Листинг 12 – Обработка Создания нового пользователя

//личный кабинет преподавателя

router.route('/teacher\_account')

    .get(cookieJwtAuth, function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                let sql = `SELECT \* FROM users WHERE login='${payload.login}'`

                let sql2=`select \* from users where (select list\_of\_students from users where login='${payload.login}') && ARRAY[login];`

                let sql3=`select id, name, time, quantity\_attempts from test where author='${payload.login}'`

                client.query(sql).then((response)=>{

                    client.query(sql2).then((resp)=>{

                        client.query(sql3, [], function(err, respon) {

                            done();

                        res.render('teacher\_account',{

                        layout: 'teacher\_account\_layout',

                        user:response.rows[0],

                        student: resp.rows,

                        tests: respon.rows

                    });

                        })

                    })

                })

                    })

    }

    })

    //сохранение изменений

    .post(function(req, res){

        var {new\_lastname, new\_firstname,new\_middlename, new\_phone, new\_email, new\_password}=req.body

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        let fields=[]

        if (new\_lastname){

            fields.push(`lastname = '${new\_lastname}'`)

        }

        if (new\_firstname){

            fields.push(`firstname = '${new\_firstname}'`)

        }

        if (new\_middlename){

            fields.push(`middlename = '${new\_middlename}'`)

        }

        if (new\_phone){

            fields.push(`phone = '${new\_phone}'`)

        }

        if (new\_email){

            fields.push(`email = '${new\_email}'`)

        }

        if (new\_password){

            fields.push(`password = '${bcrypt.hashSync(new\_password, 6)}'`)

        }

        if (fields.length!=0) {

            let sql = `update users set ${fields.join(', ')} where login='${payload.login}'`

            pool.query(sql, (request, response) =>{

                res.redirect('/teacher\_account')})}

        else {

            res.redirect('/teacher\_account')

        }

    })

Листинг 13 – Обработка запросов из Личного кабинета преподавателя

На листинге 14 представлена обработка создания теста. GET - формируется список студентов и групп, прикрепленных именно к этому преподавателю; при добавлении вопроса или варианта ответа у них динамически формируются имена, которые позже используются в POST, так же там считывается информация о настройках – settings.

//создание теста

router.route('/create\_test')

    .get(cookieJwtAuth, function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        pool.connect(function(error, client, done) {

            // список прикрепленных студентов

            let sqlStr = `SELECT UNNEST(list\_of\_students) as login FROM users where login='${payload.login}'`

            //список прикрепленных групп

            let sqlStr2 = `select user\_group from users

            join (SELECT UNNEST(list\_of\_students) as login FROM users where login='${payload.login}') as list\_st

            on list\_st.login=users.login

            group by users.user\_group`

            client.query(sqlStr).then((response)=>{client.query(sqlStr2, [], function(err, resp) {

                    done();

                res.render('create\_test',{

            layout: 'create\_test\_layout',

                list\_of\_students:response.rows,

                list\_of\_groups:resp.rows

            });

                })})

    })

    })

    //запись теста в бд

    .post(function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        if (req.body) {

            const {test\_name, hours, min, q\_a, list\_of\_students, list\_of\_groups} =req.body

            const settings=['test\_name', 'hours', 'min', 'q\_a', 'list\_of\_students', 'list\_of\_groups']

            var time=0

            if (hours){time=parseInt(hours)\*60}

            if (min){time+=parseInt(min)}

            q\_answ=[]

            var list\_keys=Object.keys(req.body)

            var c={}

            list\_keys.forEach(function(key){

                if (!settings.includes(key)){

                 if (key.indexOf('ans')==-1){

                    var list\_ans={}

                    list\_keys.forEach(function(key\_ans){

                        if (!settings.includes(key\_ans)){

                            if (key\_ans.indexOf(key+'ans')!=-1) {

                                var l\_a=req.body[key\_ans]

                                if (typeof(l\_a)=='string')

                                {list\_ans[l\_a]="off"}

                                else {list\_ans[l\_a[0]]=l\_a[1]}

                        }}})

                    c[req.body[key]]=list\_ans

                    }

                }})

            var n\_q\_a=0

            if (q\_a){n\_q\_a= q\_a}

            sql=`INSERT INTO test(name, author, quantity\_attempts, access\_for\_group,

                access\_for\_students, time, questions\_and\_answer)

                Values ('${test\_name}', '${payload.login}', ${n\_q\_a}, '{${list\_of\_groups}}',

                '{${list\_of\_students}}', ${time}, '${JSON. stringify(c)}'::json)`

            pool.query(sql, (err, response) =>{

            if (err) {console.log(err)}

            res.redirect('/create\_test')

        })

        }

    })

Листинг 14 – Обработка создания теста

//тестирование

router.route('/testing/\*')

    .get(function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        const id\_test = decodeURI(req.url).replace('/testing/','')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                let sql = `select questions\_and\_answer as "q" from test where id='${id\_test}'`

                let sql2=`select name, time, quantity\_attempts, author from test where id='${id\_test}'`

                let sql3=`select count(\*) from test\_student where test\_id='${id\_test}' and student\_id='${payload.login}'`

                client.query(sql).then((response)=>{

                    client.query(sql2).then((resp)=>{

                        client.query(sql3).then((respon)=>{

                        done();

                    var cas=response.rows[0]['q']

                    var attempt=`Попытка: ${parseInt(respon.rows[0]['count'])+1}.`

                    if (resp.rows[0]['quantity\_attempts']!=0){

                        if (respon.rows[0]['count']==resp.rows[0]['quantity\_attempts']) {

                            return res.redirect('/personal\_account')

                        }

                        attempt=`Попытка: ${parseInt(respon.rows[0]['count'])+1}/${resp.rows[0]['quantity\_attempts']}.`

                    }

                    let timer='not\_timer'

                    if (resp.rows[0]['time']!=0){

                         timer='timer'}

                    res.render('testing',{

                    layout: 'testing\_layout',

                    k:cas,

                    time:resp.rows[0]['time'],

                    test\_name: resp.rows[0]['name'],

                    attempt: attempt,

                    author:resp.rows[0]['author'],

                    timer\_name:timer,

                    })

                })

                });

                    })})

    }

    })

    .post(function(req, res) {

        const id\_test = decodeURI(req.url).replace('/testing/','')

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        const ans=req.body

        var c\_c=0

        var w\_w=0

        for (var item in ans) {

            if (item=='on') {

                if (typeof(ans[item])!='string')

                {c\_c=ans[item].length}

                else{c\_c=1}

            }

            else{

                if (typeof(ans[item])!='string')

                {w\_w=ans[item].length}

                else{w\_w=1}

            }

        }

        pool.connect(function(error, client, done) {

            let sql = `select questions\_and\_answer as "q" from test where id='${id\_test}'`

            client.query(sql).then((response)=>{

                let w=0

                let c=0

                var cas=response.rows[0]['q']

                for (var item in cas) {

                    for (var i in cas[item]){

                        if (cas[item][i]=="off"){w+=1}

                        else{c+=1}

                    }

                }

                const result=((c\_c-(w\_w\*0.5))/c)\*100

                if (payload.user\_role=='Студент'){

                    let sql2 = `INSERT INTO test\_student(test\_id, student\_id, marks, date\_testing)

                    VALUES (${id\_test}, '${payload.login}', ${result}, current\_date)`

                    client.query(sql2).then((resp, err)=>{

                        if (err){console.log(err)}

                        done();})

                }

                res.redirect('/result/'+result)

            })

        })

    })

Листинг 15-Обработка прохождения теста

На листинге 16 представлена обработка результата прохождения теста. Она просто возвращает сформированную страницу, не получает обратного запроса. Аналогично с обработками, представленными на листингах 17 и 19.

//результат прохождения теста

router.route('/result/\*')

    .get(function(req, res) {

        const current\_result = decodeURI(req.url).replace('/result/','')

        res.render('result',{

        layout: 'result\_layout',

        result:current\_result

    });

    })

Листинг 16- Обработка результата прохождения теста

Для идентификации теста необходимо передавать в url его уникальный идентификатор. Достаем его в константу current\_test.

//успеваемость по тесту

router.route('/test\_progress/\*')

    .get(function(req, res) {

        const current\_test = decodeURI(req.url).replace('/test\_progress/','')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                let sql=`select marks, date\_testing::varchar, test.name,

                        lastname, firstname, middlename, user\_group, login

                        from test\_student

                        join test on test.id=test\_student.test\_id

                        join users on test\_student.student\_id=users.login

                        where test\_id=${current\_test}`

                client.query(sql).then((response)=>{

                        done();

                    res.render('test\_progress',{

                    layout: 'test\_progress\_layout',

                    student:response.rows,

                    name:current\_test

                });

                    })})

    }

    })

Листинг 17 – Обработка Успеваемости по тесту

//личный кабинет студента

router.route('/student\_account')

    .get(cookieJwtAuth, function(req, res) {

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                //личная информация

                let sql = `SELECT \* FROM users WHERE login='${payload.login}'`

                client.query(sql).then((response)=>{

                    //список тестов

                    let sql2=`select test.id, test.name, test.author, test.quantity\_attempts from users

                right join test on users.login=test.author

                where (test.access\_for\_students && ARRAY['${payload.login}'::varchar(255)]) or

                    (test.access\_for\_group && ARRAY['${response.rows[0]['user\_group']}'::varchar(255)] and

                     users.list\_of\_students && ARRAY['${payload.login}'::varchar(255)])`

                    client.query(sql2, [], function(err, resp) {

                        done();

                    res.render('student\_account',{

                    layout: 'student\_account\_layout',

                    user:response.rows[0],

                    tests: resp.rows

                });

                    })})

                    })

    }

    })

    //сохранение изменений

    .post(function(req, res){

        var {new\_lastname, new\_firstname,new\_middlename, new\_phone, new\_email, new\_password}=req.body

        const payload = jwt.verify(req.cookies.token,'SECRET\_KEY')

        let fields=[]

        if (new\_lastname){

            fields.push(`lastname = '${new\_lastname}'`)

        }

        if (new\_firstname){

            fields.push(`firstname = '${new\_firstname}'`)

        }

        if (new\_middlename){

            fields.push(`middlename = '${new\_middlename}'`)

        }

        if (new\_user\_group){

            fields.push(`user\_group = '${new\_user\_group}'`)

        }

        if (new\_phone){

            fields.push(`phone = '${new\_phone}'`)

        }

        if (new\_email){

            fields.push(`email = '${new\_email}'`)

        }

        if (new\_password){

            fields.push(`password = '${bcrypt.hashSync(new\_password, 6)}'`)

        }

        if (fields.length!=0) {

            let sql = `update users set ${fields.join(', ')} where login='${payload.login}'`

            pool.query(sql, (request, response) =>{

                res.redirect('/student\_account')})}

        else {

            res.redirect('/student\_account')

        }

    })

Листинг 18 – Обработка запросов из Личного кабинета студента

Успеваемость студента может просматривать не только сам студент, но и преподаватель, поэтому нельзя доставать логин студента из токена – его необходимо передавать в url. Достаем этот логин в константу current\_login (листинг 19). Необходимо использовать decodeURI, так как все логины пользователей записаны русскими буквами.

//успеваемость студента

router.route('/student\_progress/\*')

    .get(function(req, res) {

        const current\_login = decodeURI(req.url).replace('/student\_progress/','')

        if (req.cookies.token){

            pool.connect(function(error, client, done) {

                let sql = `SELECT \* FROM users WHERE login='${current\_login}'`

                client.query(sql).then((response)=>{

                    let sql2=`select test\_id, marks, date\_testing::varchar, test.name from test\_student

                        join test on test.id=test\_student.test\_id where student\_id='${current\_login}'`

                        client.query(sql2).then((resp)=>{

                            done();

                            res.render('student\_progress',{

                            layout: 'student\_progress\_layout',

                            user:response.rows[0],

                            test:resp.rows

                            })

                });

                    })})

    }

    })

 Листинг 19 – Обработка успеваемости студента

# 3 Тестирование информационной системы

В целях тестирования созданной системы были созданы несколько фиктивных почтовых аккаунтов, каждый из которого был зарегистрирован в PRObatio. Данные для входа в аккаунт PRObatio и почту представлены в Приложении Б.

# Заключение

1-2 страницы

Заключение завершает изложение курсового проекта. В нем резюмируются итоги выполненной работы в виде обобщения самых существенных положений. Выводы должны отражать достижение поставленной цели работы, этапы ее реализации (достижение задач), и обобщать содержание курсового проекта. Выводы должны быть краткими, ясно и четко сформулированными.

В данном разделе необходимо показать, каким образом были решены задачи, поставленные во введении, привести основные результаты работы, сделать свои умозаключения о целесообразности и эффективности использования на практике, а также дать ряд предложений по внедрению проекта с учетом современных информационных технологий.

# Список использованных источников

1. Толковый словарь по информационному обществу и новой экономике. — [Электронный ресурс] — М. 2007. ULR: http://information\_society.academic.ru

2. Финансовый справочник, Борис Абрамович Райзберг и Леонид Шарапович Лозовский, 1999

3. Круг, С. Веб-дизайн: 2-е издание / С. Круг – Изд-во: Символ-Плюс, 2008 – 280 с.

4. Лазаро, И.К. Полный справочник по HTML, CSS и JavaScript/ И.К. Лазаро, И.К. Джозеф – Изд-во: ЭКОМ Паблишерз, 2007 – 350 с.

5. Сырых, Ю.А. Современный веб-дизайн. Эпоха Веб 3.0 / Ю.А. Сырых – Изд-во: Вильямс, 2013 - 368 с.

6. Берд, Джейсон «Веб-дизайн: Руководство разработчика»/Д.Берд – Изд-во: Питер, 2012 - 224 с.

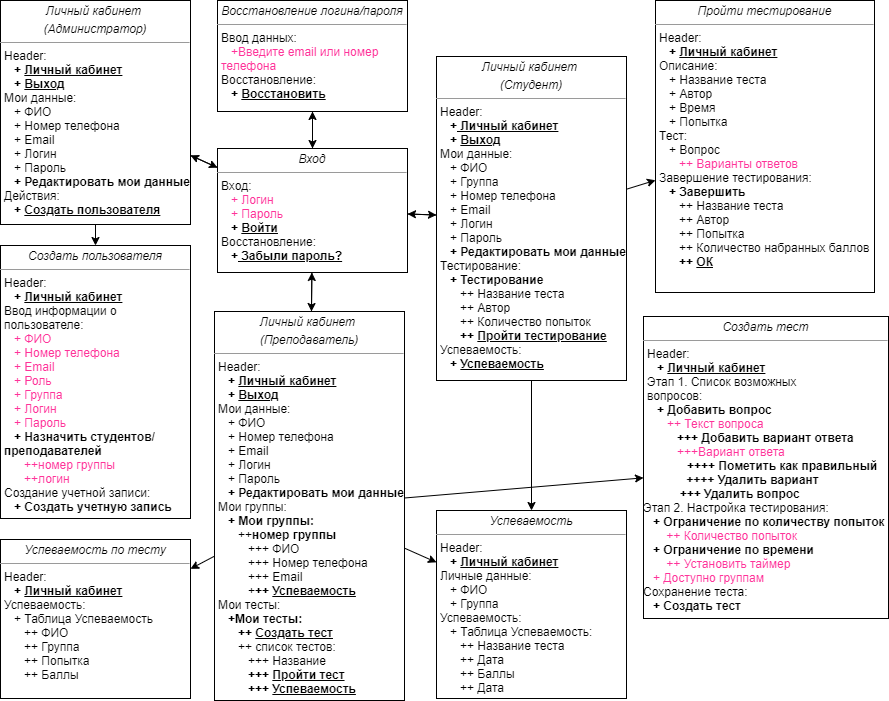
7. Фельке-Моррис Терри «Большая книга веб-дизайна»/ Терри ФелькеМоррис – Изд-во: Эксмо, 2012 - 608 с.

8. Матросов, А.В. HTML 5.0/ А.В. Матросов, М.П. Чаунин – СПб.: БХВПетербург, 2010 – 590 с.

9. Кенин, А.М. Новый уровень создания HTML-документов / А.М. Кенин, Н.С. Печенкина - Екатеринбург: Деловая книга, 2009 – 293 с.

10. Зубкова, С.В. Интерактивные Web-документы / С.В. Зубкова - М.: ДМК Пресс, 2012 – 473 с.

# Приложение А. Детализированная карта сайта



# Приложение Б

В Приложении приводится листинг всех программных элементов системы (исходный код, включая HTML, CSS, JS и т.д.) и графический материал (Презентация) к курсовому проекту. В случае большого объема исходных кодов, допускается передавать его на проверку в электронном виде (например, на электронном носителе, путем направления на электронную почту, или путем размещения в открытом репозитории исходных кодов). При этом в Приложении делается заметка о способе передачи.

* титульник англ
* аннотация
* опросник
* презентация
* список пользователей
* структура файлов