

# Оглавление

Введение .....	5
1 Теоретическая часть .....	7
1.1 Анализ предметной области .....	7
1.2 Анализ аналогов .....	8
1.3 Описание требований к информационной системе .....	12
1.4 Сравнение различных инструментов для реализации проекта.....	13
2 Проектирование .....	17
2.1 Выбор стека технологий для разработки системы.....	17
2.2 Разработка проекта решения .....	18
3 Разработка .....	26
3.1 Разработка окна авторизации .....	26
3.2 Разработка главной страницы .....	32
3.3 Разработка страницы «Мои питомцы» для пользователя .....	34
3.4 Разработка добавления собаки на странице «Мои питомцы» .....	37
3.5 Реализация удаления собаки на странице «Мои питомцы» .....	41
3.6 Разработка страницы «Эксперты» для владельца собаки .....	44
3.7 Разработка страницы «Участники» для администратора (организатора выставки).....	46
3.8 Разработка страницы «Ринги» для администратора (организатора выставки).....	53
3.9 Разработка страницы «Эксперты» для администратора (организатора выставки).....	58
3.10 Разработка страницы «Участники» для эксперта.....	63
3.11 Разработка страницы «Ринги» для эксперта .....	67
3.12 Разработка страницы «Профиль» для администратора .....	68
3.13 Разработка страницы «Профиль» для владельца собак.....	71
3.14 Разработка страницы «Профиль» для эксперта.....	74

Результаты.....	76
Список литературы.....	78

## **Введение**

С каждым годом выставки собак привлекают все большее внимание как профессиональных кинологов, так и любителей. Мероприятие является не только площадкой для демонстрации красоты и породных соответствий животных, но и важным событием для обмена опытом, продвижением пород и развития обращения с собаками в целом.

С учетом растущего интереса к данным мероприятиям разработка информационно-управляющей системы необходима. Она позволит не только автоматизировать процесс регистрации участников, но и структурировать различные данные, упростить организацию судейства и административных процессов.

**Цель курсового проекта** - разработать информационно-управляющую систему для организаторов выставки собак.

### **Задачи:**

- провести анализ предметной области для выявления основных потребностей будущих пользователей информационной системы;
- проанализировать аналоги, выявить их достоинства и недостатки;
- составить список основных функциональных и нефункциональных требований;
- провести сравнительный анализ различных фреймворков и выбрать стек технологий для реализации проекта;
- спроектировать диаграммы (Use Case, диаграмма активности);
- спроектировать базу данных (ER-диаграмма);
- разработать пользовательский интерфейс;
- разработать информационно-управляющую систему, реализовать функционал;
- провести тестирование разработанной системы;
- составить отчет

— предоставить отчет к защите

### **Объект исследования**

Объектом данного исследования является автоматизация процесса организации и проведения выставок собак.

### **Предмет исследования**

Предметом исследования в данной работе является автоматизация этапов подготовки и проведения выставок, включая регистрацию участников, выбор места и времени проведения, формирование жюри.

### **Методы исследования**

В качестве основных методов исследования применены анализ, синтез, сравнение и моделирование. Практическая реализация поставленной задачи соответствует основным подходам к разработке программного обеспечения.

### **Информационная база исследования**

Информационной базой исследования являются открытые источники, в том числе доступные в сети Интернет, а также материалы курса «Создание программного обеспечения», доступные через систему дистанционного обучения РТУ МИРЭА.

# 1 Теоретическая часть

## 1.1 Анализ предметной области

Выставка — удивительное мероприятие и яркое зрелище для всех любителей собак. Именно на выставках собираются в одном месте профессионалы-кинологи и любители-собаководы. Это место для общения, обмена опытом и определения лучших представителей пород

Основная цель выставок собак — продвижение породистых собак, демонстрация их красоты и соответствия стандартам породы. Для заводчиков выставки — это возможность показать своих собак потенциальным покупателям.

Существует огромное разнообразие выставок на сегодняшний день. Они различаются по типу организации: международные, национальные, региональные. Также кроме всепородных выставок они могут быть посвящены конкретной породе или группам пород. Итогом выставки является определение медалистов по каждой породе.

Во время выставок собаки оцениваются экспертами согласно установленным стандартам породы. Оценивается внешний вид, строение тела, движение и характер собаки.

Оценки на выставках собак:

- 1) Отлично (excellent)
- 2) Очень хорошо (very good)
- 3) Хорошо (good)
- 4) Удовлетворительно (satisfactory)
- 5) Дисквалификация (disqualification)
- 6) Невозможно отсудить / Без оценки (without evaluation)

Выставки собак пользуются широкой популярностью среди любителей собак и профессионалов. Они способствуют развитию культуры обращения с животными, продвигают разведение породистых собак и способствуют обмену опытом между заводчиками и владельцами.

## 1.2 Анализ аналогов

### Infodog.com

Одна из популярных платформ для управления выставками собак является Infodog.com. Данная система предоставляет расписание мероприятий, результаты, онлайн-регистрацию участников и просмотр информацию о судьях.

Основной функционал системы:

- просмотр актуального расписание выставок по всей стране;
- регистрация участников;
- подача заявок на участие;
- просмотр результатов выставок;
- предоставление информационного материала;
- наличие и поддержка форума.

Недостатки системы:

- скудный дизайн пользовательского интерфейса;
- неактуально для жителей других стран, кроме Америки;
- ограниченный функционал бесплатной версии;
- сложность использования системы новыми пользователями.

Далее представлен интерфейс Infodog.com



рис. 1 Главная страница Infodog.com

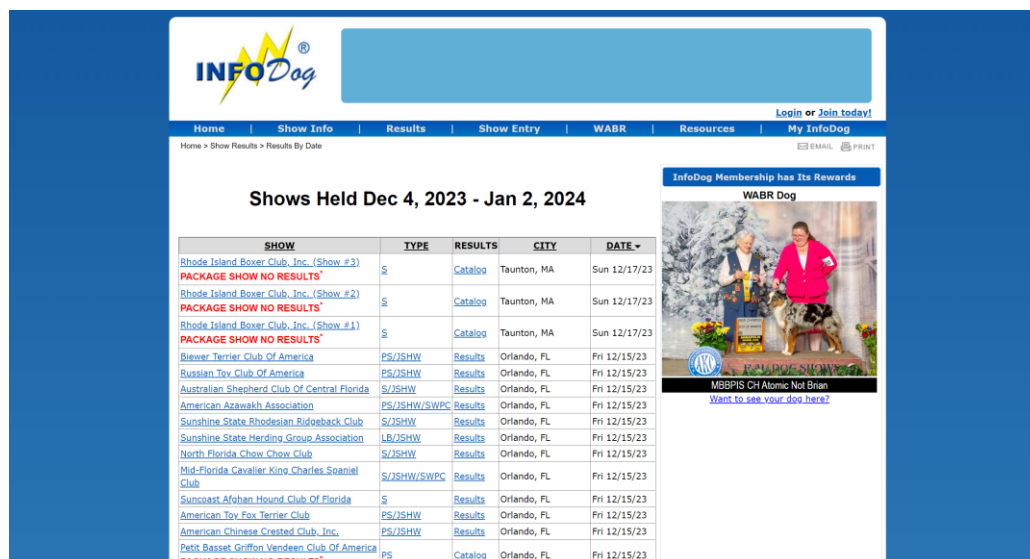


рис. 2 Страница результатов Infodog.com

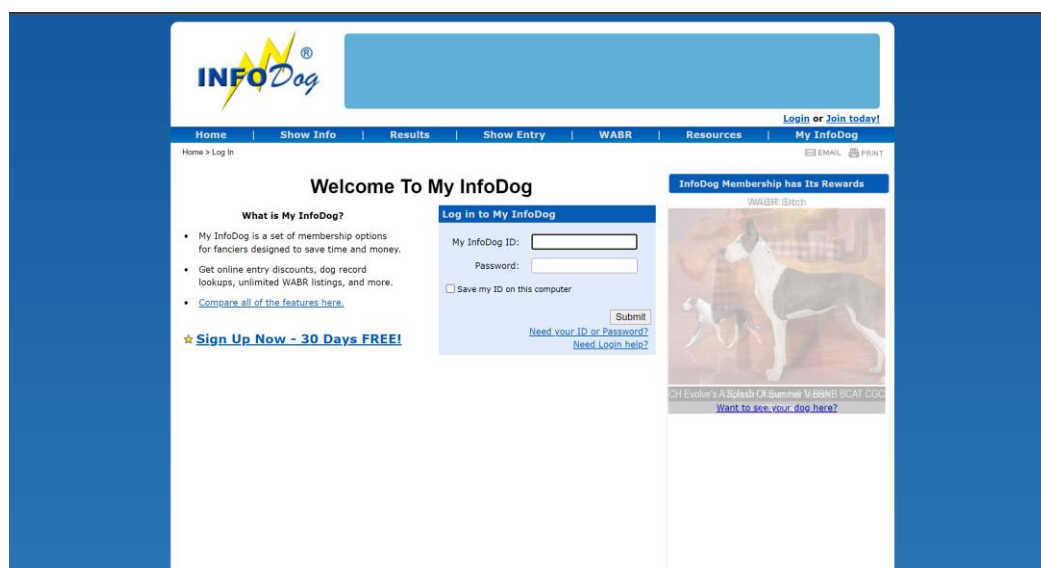


рис. 3 Страница входа Infodog.com

## ЗооПортал

Данная платформа создана для организации различных выставок собак по России. На сайте реализована онлайн-регистрация участников, календарь мероприятий, подача заявлений на участие в выставках, подача заявок в кинологические клубы, а также форум пользователей системы.

Недостатки системы:

- долгая загрузка компонентов (элементов) сайта;
- долгая загрузка картинок;
- есть проблемы у пользователей с загрузками своих картинок;

— отсутствует возможность цитирования и ответа на другие посты.

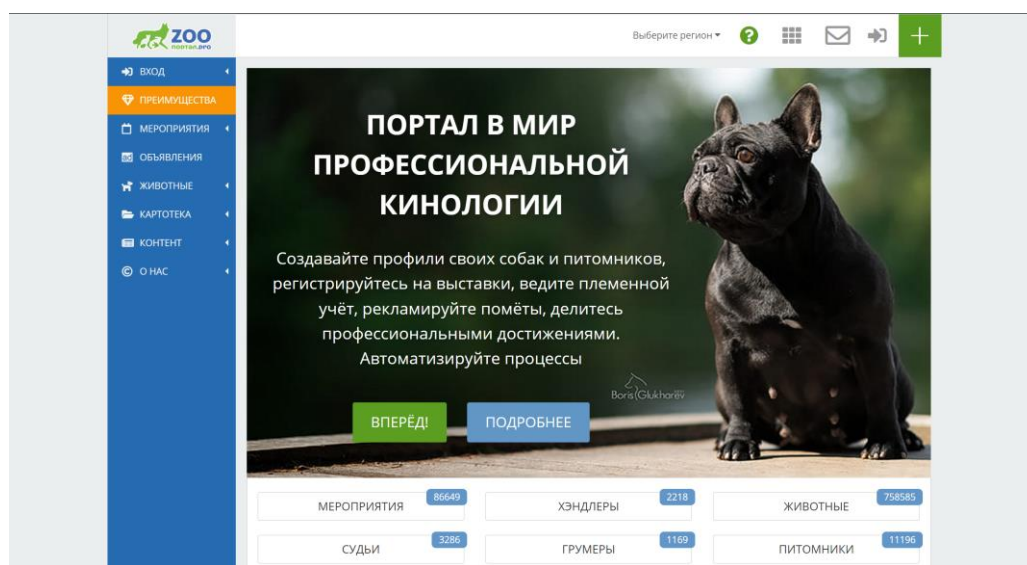


рис. 4 Главная страница ЗооПортал

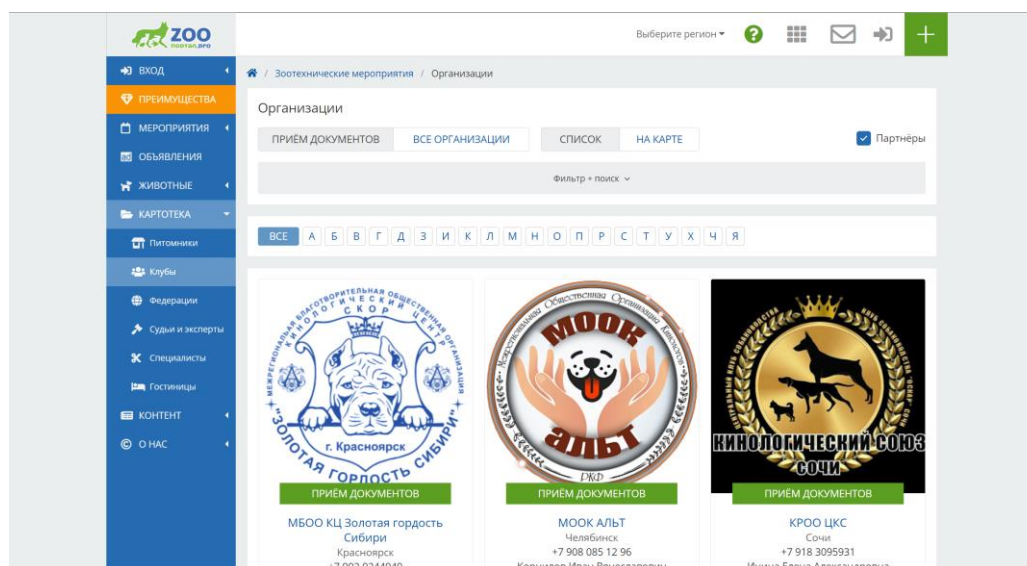


рис. 5 Страница клубов ЗооПортал



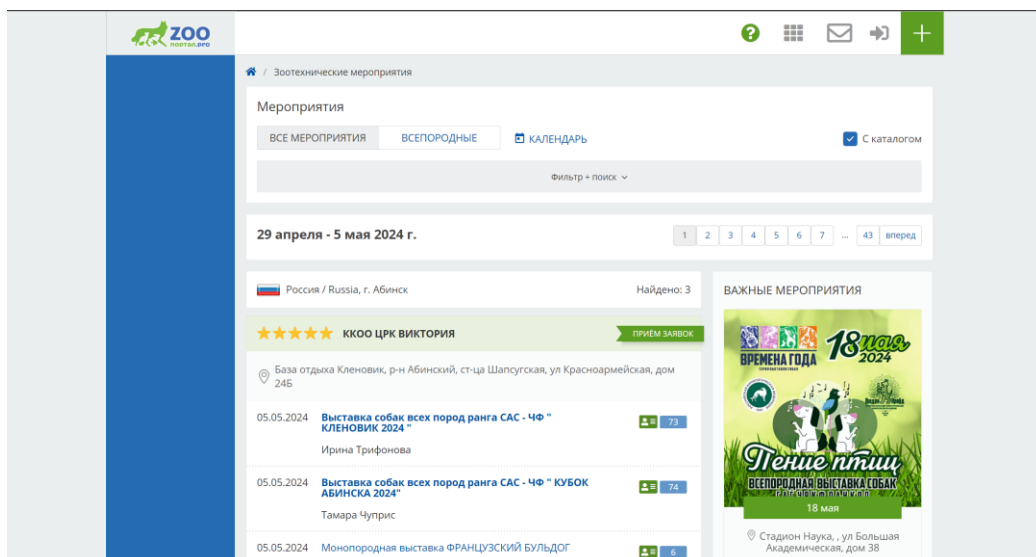


рис. 6 Страница мероприятий ЗооПортал

## SHOW-DOGS.RU

SHOW-DOGS.RU – система, созданная для поиска мероприятий и онлайн-регистрации участников на выставки. Функционал данного сайта поход на справочную службу с информацией.

Недостатки системы:

- яркий, режущий зрение пользовательский интерфейс;
- взаимодействия могут быть сложны для новых пользователей;
- нет реализации интерфейса для эксперта;
- нет поиска клубов и подачи заявлений в них.

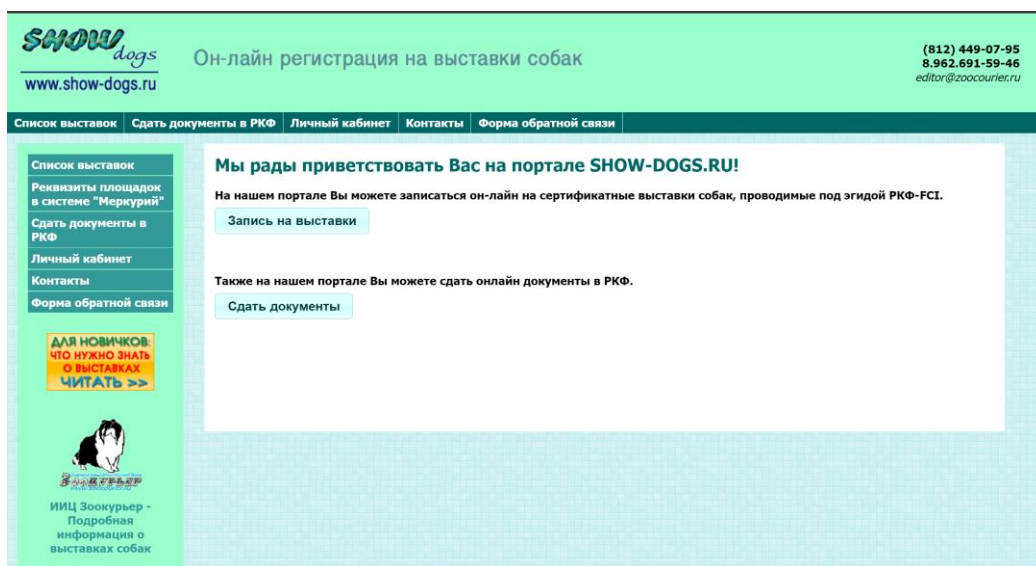


рис. 7 Главная страница SHOW-DOGS.RU

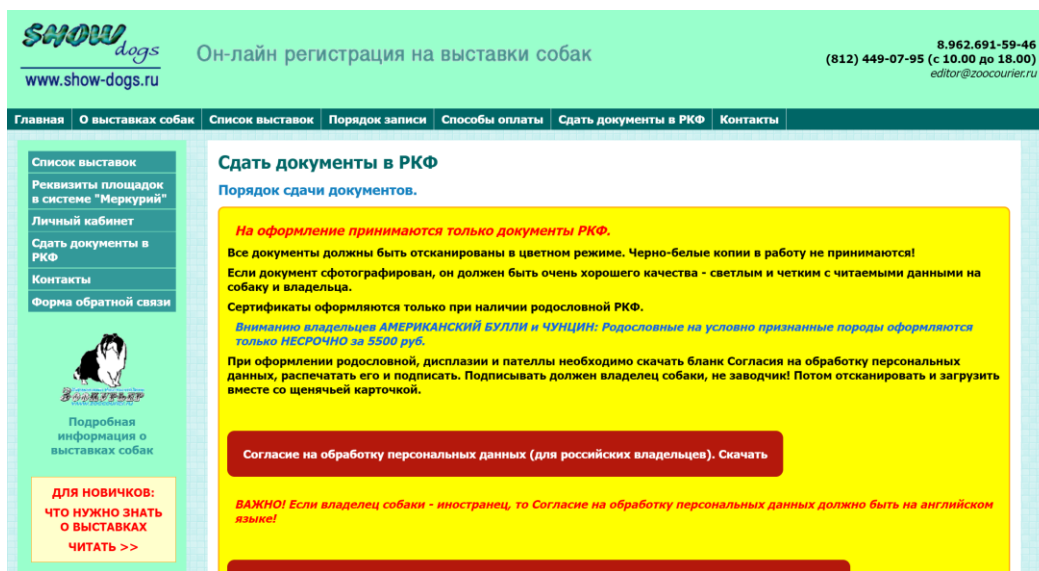


рис. 7 Страница регистрации на мероприятие SHOW-DOGS.RU

### 1.3 Описание требований к информационной системе

Разработанная система должна выполнять следующие требования.

- 1) Система обеспечивает возможность онлайн-регистрации участников выставки собак, включая владельцев собак, судей (экспертов).
- 2) Предусмотрена проверка и подтверждение заявок на участие.
- 3) Система позволяет вносить, сохранять и редактировать информацию о каждой собаке. Возможно указать породу, возраст, кличку собаки и данные о ее владельце.
- 4) Система обеспечивает хранение сведений о собаках.
- 5) Обеспечена возможность загрузки фотографий собак.
- 6) Система позволяет выбирать и назначать судей на выставку собак.
- 7) Система позволяет вводить информацию об оценках участника.
- 8) Система позволяет управлять доступом к информации и функционалу в зависимости от роли пользователей (администратор, эксперт, владелец собаки).
- 9) Система должна обеспечивать защиту персональных данных участников и другой конфиденциальной информации.

- 10) Должны быть созданы обобщенные списки.
  - Сведения о собаках.
  - Сведения о рингах.
  - Сведения об экспертах.
- 11) Организатору выставки могут потребоваться следующие сведения.
  - На каком ринге выступает заданный хозяин со своей собакой.
  - Какие эксперты обслуживают данную породу.
  - Распределение специализаций по рингам.
  - Рекордсмены по породам по количеству медалей.

## 1.4 Сравнение различных инструментов для реализации проекта

### Сравнение инструментов для базы данных

На сегодняшний день существует огромное количество программного обеспечения, которое позволяет проектировать базы данных. Далее представлен сравнительный анализ таких инструментов, как MySQL, MongoDB и PostgreSQL.

MySQL – одна из самых используемых и удобных систем управления базами данных. Это реляционная СУБД, значит данные в ней хранятся в виде таблиц. Система поддерживает JSON-формат данных, бесплатна для использования. Она кроссплатформенная работает на Linux, Windows, FreeBSD. Высокая производительность данной системы отлично подходит для веб-приложений, однако только для малых или средних проектов. Для больших проектов, где могут быть миллионы строк, производительность сильно падет. Еще одним недостатком является затруднительный переход на другие СУБД из-за того, что MySQL не соответствует требованиям SQL.

**MongoDB** – это файловая СУБД, данные могут храниться в форматах JSON, BSON, благодаря чему можно хранить фото, видео и аудиоматериалы и более эффективно хранить данные, например, об одном предмете/продукте/человеке. Можно использовать с Java, PHP, Python, Perl, C#. За счет того, что это NoSQL

СУБД, она подходит и для структурированных, и для неструктурированных данных. Однако данная СУБД плохо подходит для проектов, которые требуют высокой безопасности и соответствия требованиям ACID, проектов с бизнес-логикой и транзакциями.

**PostgreSQL** – объектно-реляционная СУБД с открытым исходным кодом. Поддерживает популярные языки программирования (Python, Java, Perl, PHP, C++), а также разные форматы данных (сетевые адреса, данные JSON, координаты геопозиций). Размер БД не ограничен, зависит только от свободной памяти в месте хранения. Полностью соответствует требованиям ACID. Лучше осуществляет частное обновление. PostgreSQL лучше подходит для корпоративных приложений с частыми операциями записи и сложными запросами, а также для организаций, которым приходится иметь дела с крупными и потенциально бесконечно масштабируемыми базами данных.

### **Сравнение фреймворков для фронтенда**

Ценность прогрессивных веб-технологий неоспорима, поэтому на рынке появляется все больше фреймворков и инструментов. Поскольку каждый из них обладает своими возможностями, ожидаемыми наборами функций будущих приложений, их размер, сложность, ожидаемая масштабируемость, а также наличие мультимедийных и интерактивных элементов — все это играет важную роль при выборе того или иного фреймворка. В ходе практической работы был проведен анализ популярных фреймворков для разработки веб-приложения.

**React** – это JS-библиотека, которая хорошо известна разработчикам благодаря своему широкому разнообразию инструментов. Данный фреймворк использует JSX для синхронизации с форматом HTML. Основными преимуществами является высокая скорость работы, реализация концепции функционального программирования, встроенное тестирование, виртуальный DOM и стабильность. Однако есть необходимость в дополнительных инструментах и отсутствует хорошая документация.

**Angular** – это JavaScript-фреймворк. Все компоненты Angular являются модульными, то есть совместимыми с другими фреймворками и инструментами

для разработки веб-приложений. Основные преимущества – подробная документация, масштабируемость, двусторонняя привязка данных, встроенные возможности тестирования. Недостатки инструмента – сложная архитектура, относительно низкая производительность.

**Vue** имеет некоторые сходства с **React**, поскольку они оба используют **Virtual DOM** для эффективного и облегченного представления реального **DOM**. **Vue** использует и улучшает **HTML** и **CSS**. Этому фреймворку удалось упростить код, обеспечив упрощенный процесс разработки и высокоскоростной рендеринг. Благодаря этому данный фреймворк отлично подходит для начинающих фронтенд-разработчиков. Однако он также имеет свои недостатки, в числе которых шаблонизация хуже, чем в **React** и система рендеринга менее функциональна, чем в **React**.

### **Сравнение фреймворков для бэкенда**

В данной курсовой работе было проведен сравнительный анализ наиболее популярных инструментов для серверной части веб-приложения, среди которых **Express.js**, **Django**, **Ruby on Rails**. Ниже представлена таблица сравнения (Таблица 1. Таблица сравнения фреймворков).

*Таблица 1. Таблица сравнения фреймворков*

Фреймворки	Достоинства	Недостатки
Express.js	<ul style="list-style-type: none"> <li>• Высокая производительность</li> <li>• Интегрируется с наиболее популярными СУБД (MongoDB, MySQL, PostgreSQL и др.)</li> <li>• Дополняет функциональные возможности Node.js</li> <li>• Поддержка большого количества шаблонизаторов</li> <li>• Совместим с другими Node.js фреймворками</li> </ul>	<ul style="list-style-type: none"> <li>• Требуется использование Node.js</li> <li>• Не подходит для объемных вычислительных процессов</li> <li>• Мало инструментов для тестирования</li> </ul>

Django	<ul style="list-style-type: none"> <li>• Интегрируется с наиболее популярными СУБД (MongoDB, MySQL, PostgreSQL и др.)</li> <li>• Высокий уровень безопасности</li> <li>• Подробная документация</li> <li>• Стандартизированная структура</li> </ul>	<ul style="list-style-type: none"> <li>• Сложность в адаптации Django ORM</li> <li>• Монолитная архитектура может снизить производительность</li> <li>• Возможна перегрузка функциональности</li> <li>• Менее эффективен в работе с веб-сокетами, чем Node.js</li> </ul>
Ruby on Rails	<ul style="list-style-type: none"> <li>• Высокая скорость разработки</li> <li>• Высокий уровень безопасности</li> <li>• Интегрируется с наиболее популярными СУБД (MongoDB, MySQL, PostgreSQL и др.)</li> <li>• Строгое соблюдение веб-стандартов</li> </ul>	<ul style="list-style-type: none"> <li>• Более медленная скорость работы, чем Node.js</li> <li>• Ограниченная гибкость при необходимости уникального подхода</li> </ul>
Laravel	<ul style="list-style-type: none"> <li>• Высокий уровень безопасности</li> <li>• Гибкая система маршрутизации</li> <li>• Просто проводить автоматизированное тестирование</li> </ul>	<ul style="list-style-type: none"> <li>• Более медленная скорость работы</li> <li>• Отсутствие встроенной технической поддержки</li> <li>• Поддерживает работу с MongoDB только с помощью доп. расширения</li> </ul>

## 2 Проектирование

### 2.1 Выбор стека технологий для разработки системы

Для моделирования и проектирования диаграмм были выбраны такие инструменты, как Excalidraw (Use Case), DrawSQL (ER-диаграмма), а также draw.io (Диаграмма последовательности и диаграмма активности). Для разработки дизайна пользовательского интерфейса было использовано Pixso, так как данное приложение полностью бесплатно, а интерфейс идентичен ранее используемой программе Figma.

Для разработки информационной системы для организаторов выставки собак был выбран стек технологий, состоящий из React (frontend), Node.js и Express.js (backend), PostgreSQL (база данных).

Ниже представлена таблица достоинств и недостатков данного набора инструментов (Таблица 2. Достоинства и недостатки выбранного стека).

Таблица 2. Достоинства и недостатки выбранного стека

	Достоинства	Недостатки
React	1 Виртуальный DOM обеспечивают высокую производительность веб-приложения. 2 Подходит для разработки как небольших, так и крупных проектов благодаря компонентной архитектуре. 3 Имеет активное сообщество разработчиков, множество библиотек и инструментов для улучшения разработки.	1 Поскольку React приложение загружается как единый JavaScript файл, его первоначальная загрузка может занимать некоторое время, особенно на медленных соединениях.
Node.js	1 Асинхронная и неблокирующая модель ввода-вывода позволяет создавать масштабируемые и производительные серверные приложения. 2 Использование JavaScript как на фронтенде, так и на бэкенде, позволяет создавать приложения с общим кодом и легким обменом данными между клиентом и сервером. 3 Большое количество пакетов в npm	1 Поскольку Node.js работает в одном потоке, блокирующие операции могут привести к замедлению всего приложения. 2 Вложенные обратные вызовы (callback hell) могут сделать код менее читаемым и поддерживаемым.

	репозитории делает Node.js очень гибким для решения различных задач.	
Express.js	<p>1 Простой и минималистичный фреймворк, который позволяет быстро создавать серверные приложения.</p> <p>2 Предоставляет достаточно свободы разработчику для выбора архитектуры и паттернов разработки.</p> <p>3 Возможность использования middleware позволяет легко добавлять дополнительную функциональность в приложение.</p>	<p>1 Для некоторых распространенных функциональных возможностей (например, аутентификация) может потребоваться использование сторонних middleware.</p>
PostgreSQL	<p>1 Обеспечивает надежность и целостность данных.</p> <p>2 Поддержка различных методов репликации и кластеризации делает PostgreSQL подходящим для масштабирования как маленьких, так и крупных проектов.</p> <p>3 PostgreSQL предоставляет множество функций, включая поддержку геоданных, полнотекстовый поиск, JSON.</p>	<p>1 Настройка PostgreSQL может быть сложной, особенно для начинающих разработчиков.</p> <p>2 В случае изменения структуры базы данных могут возникнуть проблемы с миграцией данных и обновлением приложения.</p>

## 2.2 Разработка проекта решения

В ходе курсовой работы разработана система управления выставки собак. Система обеспечивает хранение сведений о собаках – участниках выставки, экспертах и рингах.

### **Возможности взаимодействия пользователей с системой.**

Пользователи системы, являющиеся хозяевами собак, могут добавить сведения о своем питомце, подать заявку на участия в выставке, предварительно выбрав собаку и ринг. Хозяин за одну выставку может подать несколько заявок. Пользователи могут стать экспертом на выставке собак, указав специализацию (породу) в личном кабинете.

### **Возможности взаимодействия эксперта с системой.**

Эксперт может подать заявление на обслуживание ринга (каждый эксперт



имеет право на обслуживании нескольких рингов). Также эксперт может просматривать информацию об участниках ринга, который обслуживает. Он оценивает участников, у которых совпадает ринг с рингом эксперта, выставяя оценки (от 0 до 5).

#### **Возможности админа взаимодействия с системой.**

Админ является организатором выставки. В его обязанности входит создание/удаление рингов, одобрение заявок участников и экспертов. А также добавление новых участников и экспертов на ринги из созданных списков.

Далее наиболее наглядно представлено взаимодействия пользователей системы (рис 8. Use case). А также структура базы данных (рис 9. ER-диаграмма).

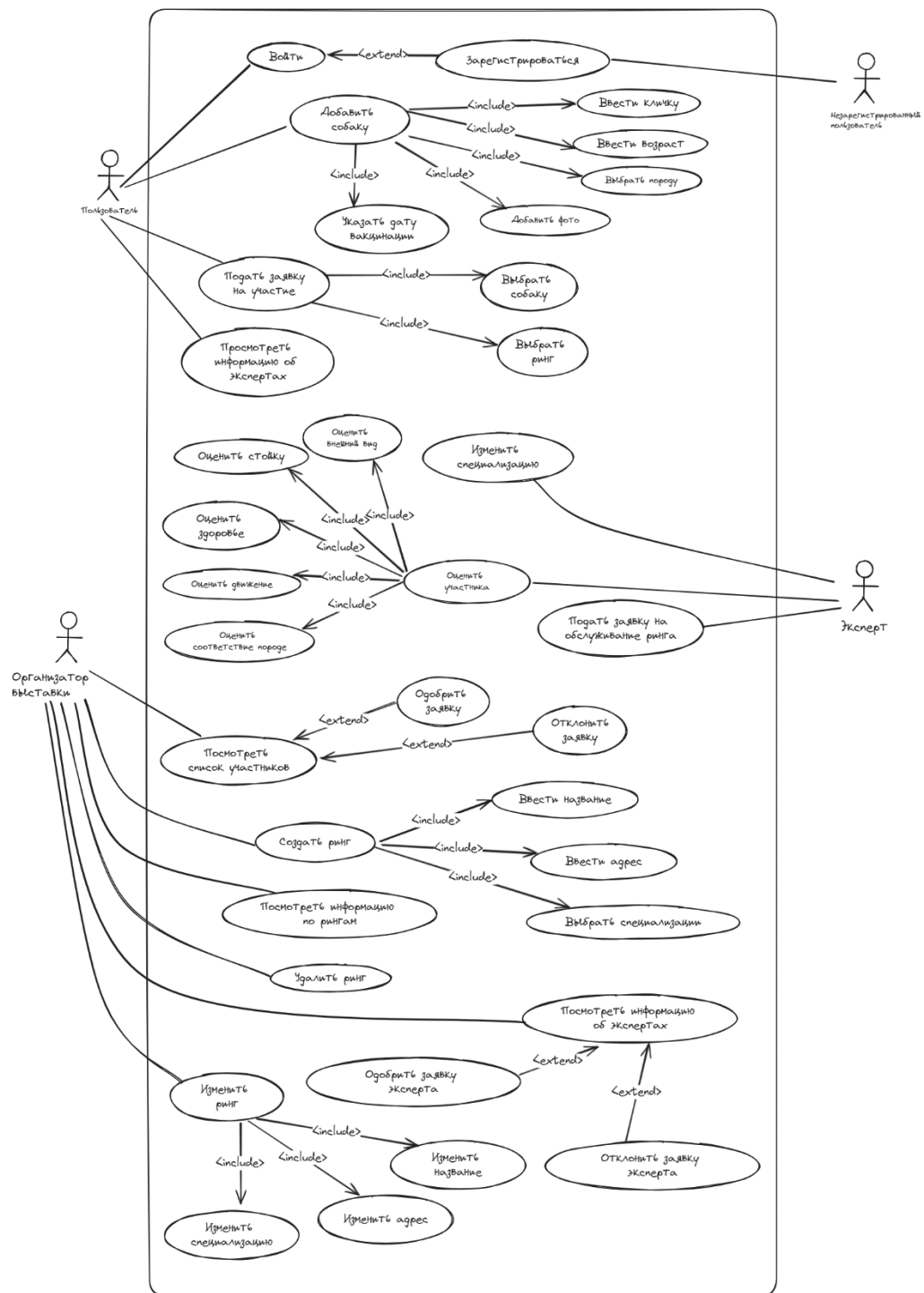


рис 8. Use Case



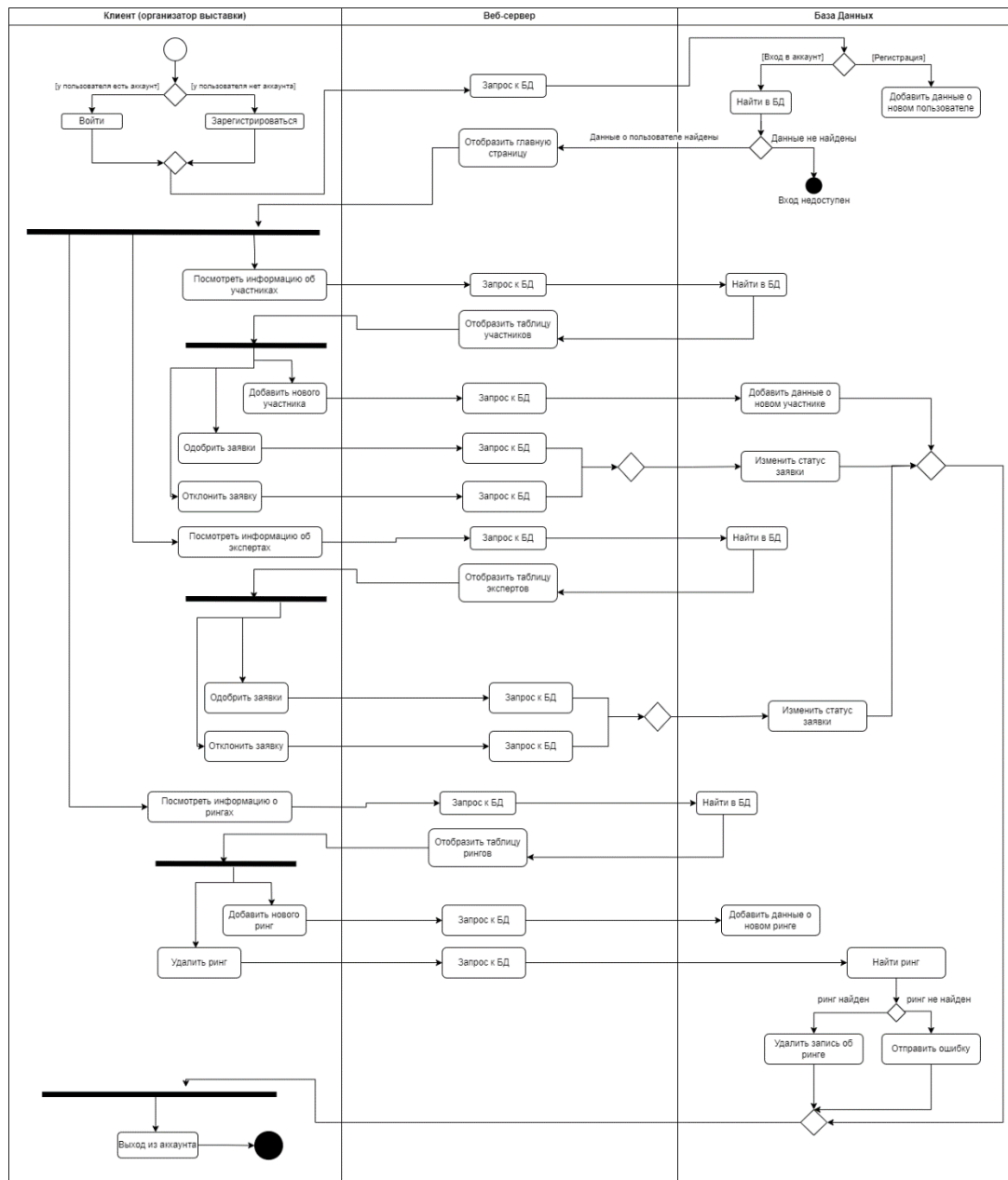


рис 11. Диаграмма активности

## Проектирование пользовательского интерфейса

Для пользовательского интерфейса была выбрана нейтральная цветовая гамма, чтобы не отвлекать пользователя от основного функционала приложения. На домашней странице администратор (как и любой пользователь) видит статистические данные о выставке. На последующих экранах происходит взаимодействие с данными из БД, их манипуляция.

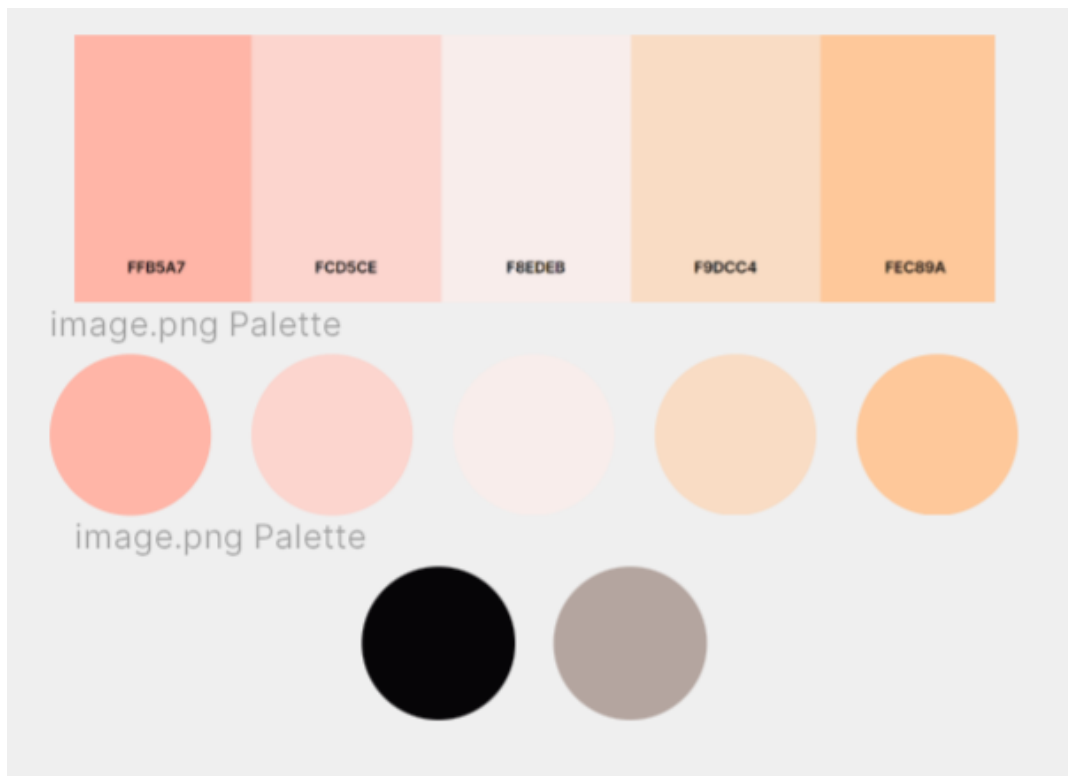
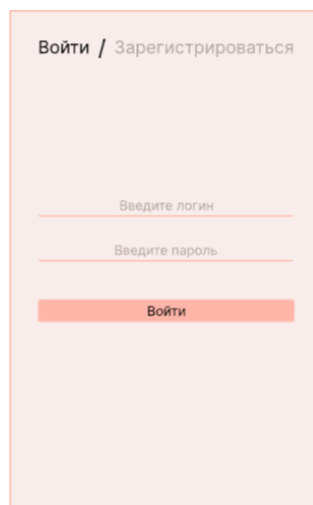


рис. 12 Цветовая гамма проекта



рис. 13 Приветственная страница



Войти / Зарегистрироваться

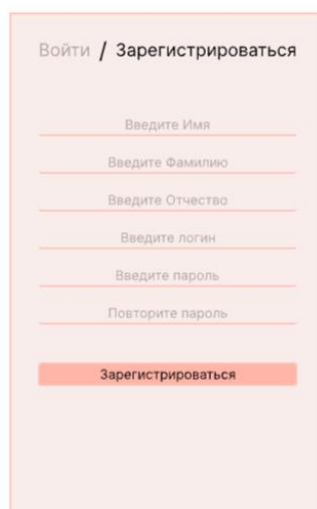
Введите логин

Введите пароль

Войти

This is a login form with a light pink background. At the top, it has a link 'Войти / Зарегистрироваться'. Below this are two input fields: 'Введите логин' and 'Введите пароль'. At the bottom is a red button labeled 'Войти'.

*рис. 14 Форма входа поверх приветственной страницы*



Войти / Зарегистрироваться

Введите Имя

Введите Фамилию

Введите Отчество

Введите логин

Введите пароль

Повторите пароль

Зарегистрироваться

This is a registration form with a light pink background. At the top, it has a link 'Войти / Зарегистрироваться'. Below this are six input fields: 'Введите Имя', 'Введите Фамилию', 'Введите Отчество', 'Введите логин', 'Введите пароль', and 'Повторите пароль'. At the bottom is a red button labeled 'Зарегистрироваться'.

*рис. 15 Форма регистрации поверх приветственной страницы*

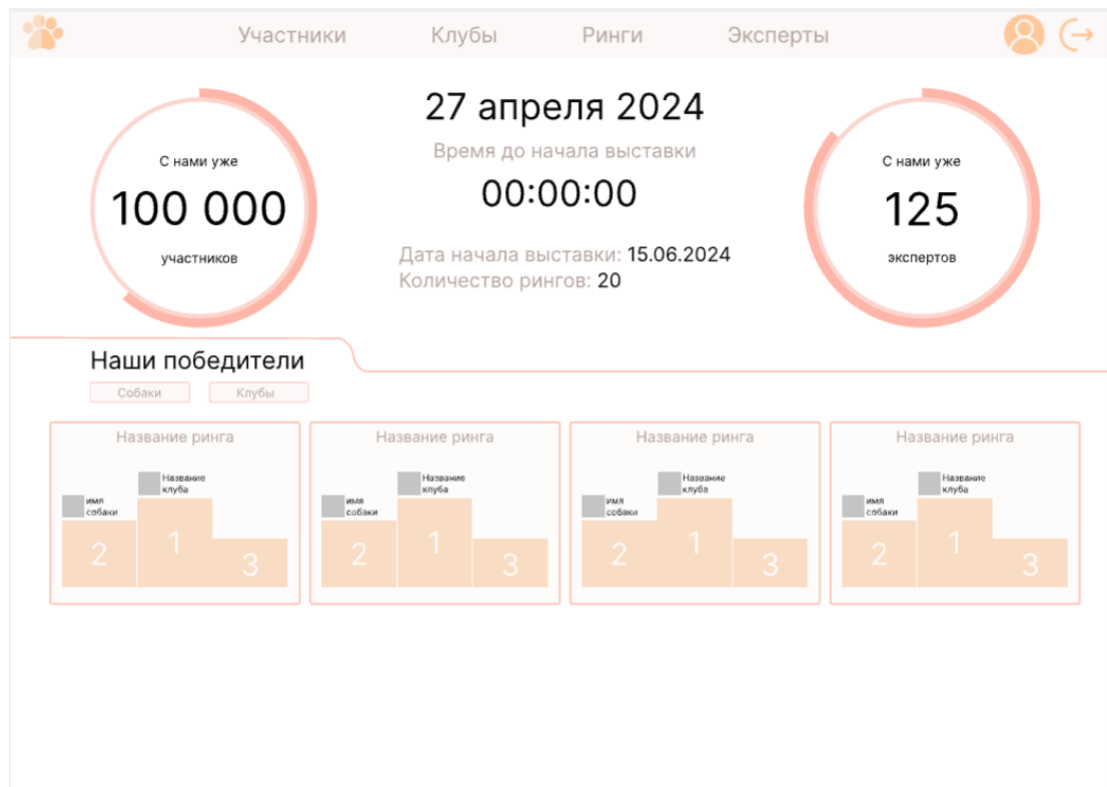


рис. 16 Домашняя страница администратора (организатора выставки)

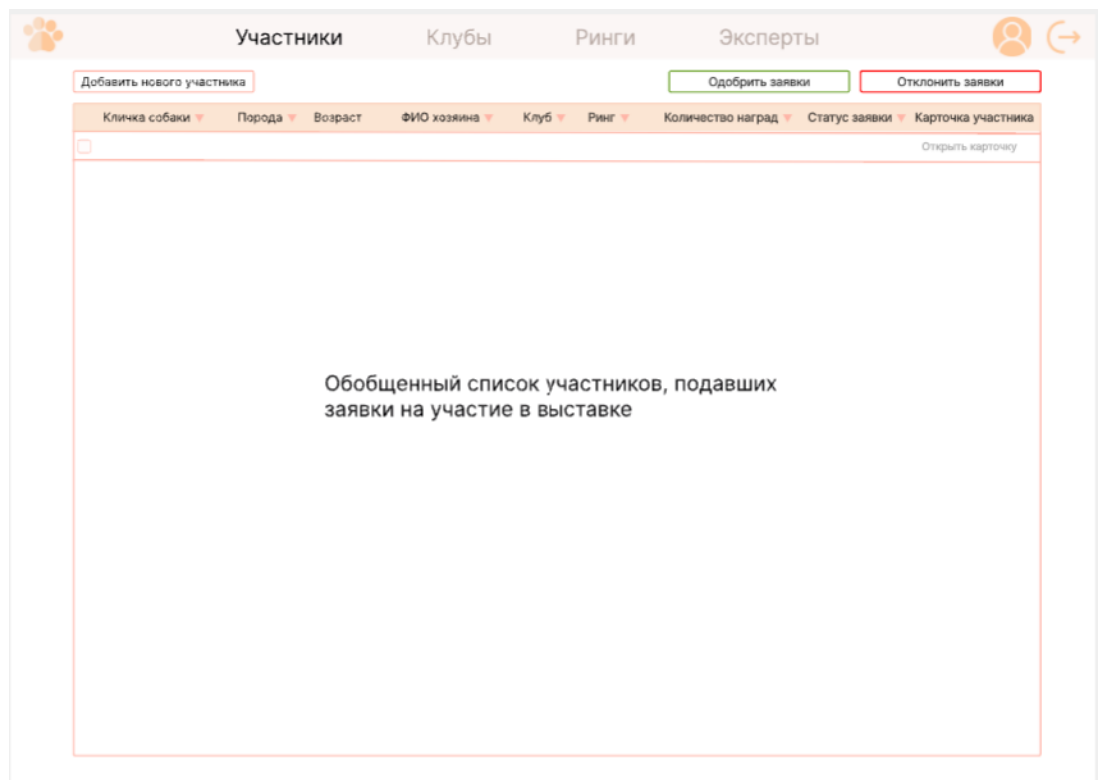


рис. 17 Пример экрана для манипуляции данными из БД и просмотра информации

## 3 Разработка

### 3.1 Разработка окна авторизации

Для разработки окна авторизации необходимы поля для ввода логина и пароля. Запрос к базе данных для проверки существования пользователя и проверки валидности пароля. Если пользователь ввел неправильные данные вывод ошибки.

Если пользователь не зарегистрирован в системе необходимо перейти в окно регистрации и корректно ввести свои данные. Система предупреждает о неправильном вводе пароля и несоответствии повторного пароля.

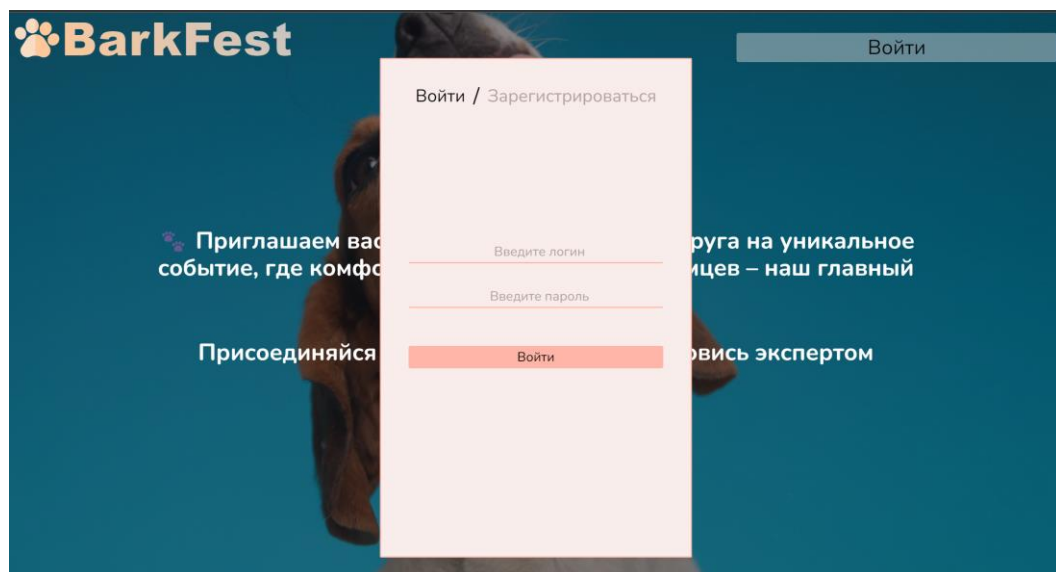


рис. 18 Страница входа

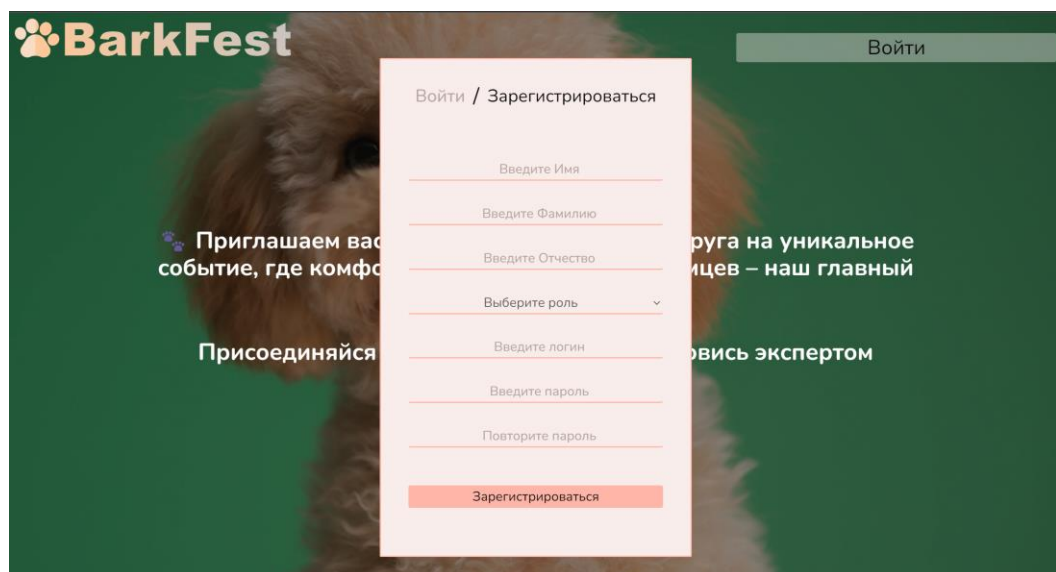


рис. 19 Форма регистрации

Листинг 1 – Код для валидации данных



```

import React from 'react'
import axios from 'axios';
import { useState } from 'react';
import style from './SignIn.module.css'
import { useForm } from 'react-hook-form';
import { useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
const SignIn = () => {
  const navigate = useNavigate()
  const [loggedIn, setLoggedIn] = useState(false);
  const [login, setLogin] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleLogin = async () => {
    try {
      await axios.get(`http://localhost:8082/auth?login=${login}`).then(data => {
        {
          localStorage.setItem('token', data.data.token);
        }
      });
      const token = localStorage.getItem('token')
      if(token){
        const decodedToken = jwtDecode(token);
        const role_id = decodedToken.role_id;
        const pass = decodedToken.password;

        if( password === pass) {
          setLoggedIn(true);
          if(role_id === '2') navigate('/home')
          else if(role_id === '3') navigate('/expert')
          else navigate('/admin')
        }
        else {setError('Неверное имя пользователя или пароль');}}
    } catch (error) {
      console.error('Error searching:', error);
    }
  };

  const [overlay, getOverlay] = useState(true)

  const handleClick = ()=>{
    getOverlay(!overlay)
  }
  const { register, handleSubmit, watch, formState: { errors } } = useForm();
  const onSubmit = async(data) => {
    if (data.role_id === "0"){setError('Не выбрана роль')}
    else{const new_data ={
      name: data.name,

```

```

    surname: data.surname,
    patronymic: data.patronymic,
    login: data.login,
    password: data.password,
    role_id: data.role_id
  }
  try{
    await axios.post('http://localhost:8082/register', new_data)
    .then(data=>{
      localStorage.setItem('token', data.data.token);
    });

    navigate("/")
  }
  catch (error){
    console.error('Error searching:', error);
  };
}}

return (
  <>
  <div className={style.container}>
    <div className={style.content}>
      <div className={style.nav}>
        <div onClick={handleClick} className={` ${overlay} &&
style.active}`>Войти</div>
        <span></span>
        <div onClick={handleClick} className={` ${!overlay} &&
style.active}`>Зарегистрироваться</div>
      </div>
      {overlay?
        (<div className={style.main}>
          {loggedIn ? (
            <div></div>
          ) : (
            <>
              {error && <p>{error}</p>}
              <input
                type="text"
                placeholder="Введите логин"
                value={login}
                onChange={(e) => setLogin(e.target.value)}
              />
              <input
                type="password"
                placeholder="Введите пароль"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
              />
            </>
          )}
        )
      }
    </div>
  </div>
)

```

```

        <button className={style.btn}
onClick={handleLogin}>Войти</button>

        </>
      )}
    </div>):
    (<form onSubmit={handleSubmit(onSubmit)}>
      <input type="text" placeholder="Введите Имя" {...register("name",
{required: true, maxLength: 100})} />
      <input type="text" placeholder="Введите Фамилию" {...register("surname",
{required: true, maxLength: 100})} />
      <input type="text" placeholder="Введите Отчество"
{...register("patronymic", {maxLength: 100})} />
      <select style={{color: 'rgb(98, 90, 87)'}} {...register("role_id",
{required: true})} >
        <option value="0">Выберите роль</option>
        <option style={{color: 'rgb(98, 90, 87)'}}value="2">Хозяин
собаки</option>
        <option style={{color: 'rgb(98, 90, 87)'}}value="3">Эксперт</option>
      </select>
      <input type="text" placeholder="Введите логин" {...register("login",
{required: true})} />
      <input type="password" placeholder="Введите пароль"
{...register("password", {required: true, minLength: 6, pattern: /^(?=.*[a-zA-
Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]+$/)}/>
      {errors.password && <span>Пароль должен состоять из букв латинского
алфавита, содержать цифры и спец. символы</span>}
      <input type="password" placeholder="Повторите
пароль" {...register('confirmPassword', { required: true })}/>
      {errors.confirmPassword && <span>Необходимо повторить пароль</span>}
      {watch('password') !== watch('confirmPassword') && <span style={{margin:
'0'}}> Пароли не совпадают</span>}

      <button type="submit">Зарегистрироваться</button>
    </form>)
  }

</div>
</div>
</>

)
}
export default SignIn

```

Асинхронная функция **handleLogin**, которая используется для обработки логина пользователя, выполняет следующие шаги:

1) асинхронный HTTP GET-запрос на локальный сервер:

```
await axios.get(`http://localhost:8082/auth?login=${login}`)
```

2) обработка ответа:

```
.then(data => {  
    localStorage.setItem('token', data.data.token);  
})
```

Если запрос успешен, в ответе ожидается объект с токеном аутентификации (data.data.token). Этот токен сохраняется в localStorage под ключом 'token'.

3) извлечение и декодирование токена:

```
const token = localStorage.getItem('token')  
if(token){  
    const decodedToken = jwtDecode(token);  
    const role_id = decodedToken.role_id;  
    const pass = decodedToken.password;
```

4) проверка пароля и авторизация:

```
if(password === pass) {  
    setLoggedIn(true);  
    if(role_id === '2') navigate('/home')  
    else if(role_id === '3') navigate('/expert')  
    else navigate('/admin')  
}  
else {  
    setError('Неверное имя пользователя или пароль');  
}
```

Проверяется, совпадает ли введенный пользователем пароль (password) с паролем из токена (pass). Если пароли совпадают, пользователь считается успешно залогиненным (setLoggedIn(true)), и в зависимости от его роли (role\_id) происходит перенаправление на разные страницы:

role\_id === '2': Перенаправление на /home.

role\_id === '3': Перенаправление на /expert.

В других случаях: Перенаправление на /admin.

Если пароли не совпадают, устанавливается сообщение об ошибке.

Асинхронную функция **onSubmit**, которая обрабатывает отправку данных формы для регистрации нового пользователя.

1) Проверка выбора роли пользователя:

```
if (data.role_id === "0") {  
    setError('Не выбрана роль');  
} else {
```

Функция начинает с проверки, выбрал ли пользователь роль. Если role\_id равно "0", вызывается функция **setError** с сообщением 'Не выбрана роль', и дальнейшее выполнение функции прекращается.

2) Создание объекта с данными для отправки:

```
const new_data = {  
    name: data.name,  
    surname: data.surname,  
    patronymic: data.patronymic,  
    login: data.login,  
    password: data.password,  
    role_id: data.role_id  
}
```

Если роль выбрана, создается объект new\_data, который содержит имя (name), фамилию (surname), отчество (patronymic), логин (login), пароль (password) и роль (role\_id) пользователя.

3) Асинхронный HTTP POST-запрос для регистрации:

```
try {  
    await axios.post('http://localhost:8082/register', new_data)
```

```
.then(data => {
    localStorage.setItem('token', data.data.token);
});
```

Проверка правильности ввода данных происходит за счет регулярных выражений и ограничений в **input**.

### 3.2 Разработка главной страницы

Для разработки главной страницы необходимо сделать таймер отсчета времени до начала выставки. Показ текущей даты.

Слайдер реализован с помощью библиотеки **swiper**.

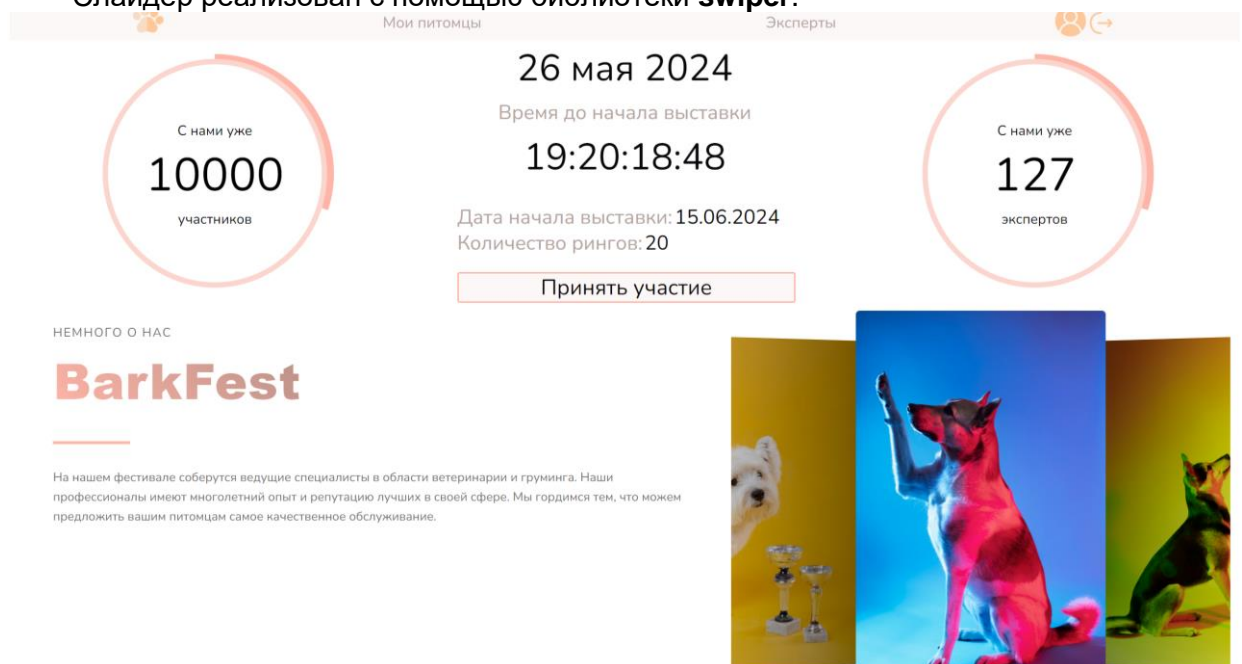


рис. 20 Главная страница.

#### Листинг 2 – Код для таймера

```
import React, { useState, useEffect } from 'react';

function Countdown() {
  const calculateTimeLeft = () => {
    const difference = +new Date("2024-06-15") - +new Date();
    let timeLeft = {};

    if (difference > 0) {
      timeLeft = {
        дней: Math.floor(difference / (1000 * 60 * 60 * 24)),
        часов: Math.floor((difference / (1000 * 60 * 60)) % 24),
        минут: Math.floor((difference / 1000 / 60) % 60),
        секунд: Math.floor((difference / 1000) % 60)
      };
    }
  };
}
```

```

    return timeLeft;
  };

  const [timeLeft, setTimeLeft] = useState(calculateTimeLeft());

  useEffect(() => {
    const timer = setTimeout(() => {
      setTimeLeft(calculateTimeLeft());
    }, 1000);

    return () => clearTimeout(timer);
  });

  const timerComponents = [];

  Object.keys(timeLeft).forEach(interval => {

    timerComponents.push(
      <div key={interval}>
        {interval === 'секунд' ? <>
{timeLeft[interval]}</>:<>{timeLeft[interval]}{":"}</> }
        </div>
      );
    });

  return (
    <div style={{display: 'flex', justifyContent: 'center'}}>
      {timerComponents.length ? timerComponents : <span>0:0:0:0</span>}
    </div>
  );
}

export default Countdown;

```

Функция **calculateTimeLeft** вычисляет оставшееся время до указанной даты. Она сначала определяет разницу между текущей датой и целевой датой в миллисекундах. Затем, если разница положительна (то есть указанная дата еще не наступила), вычисляются дни, часы, минуты и секунды до целевой даты, и возвращается объект `timeLeft`, содержащий эти значения.

Хук состояния (**useState**) используется для хранения оставшегося времени. Начальное значение состояния устанавливается вызовом функции `calculateTimeLeft`.

Хук эффекта (**useEffect**) используется для установки таймера, который

каждую секунду обновляет состояние timeLeft, вызывая функцию calculateTimeLeft.

Наиболее подробно со всем кодом можно ознакомиться в *Приложении 1*.

### 3.3 Разработка страницы «Мои питомцы» для пользователя

Для разработки страницы «Мои питомцы» необходимо реализовать запрос к базам данных на получение информации о собаках пользователя. Обработать результат и вывести в карточки на сайте.

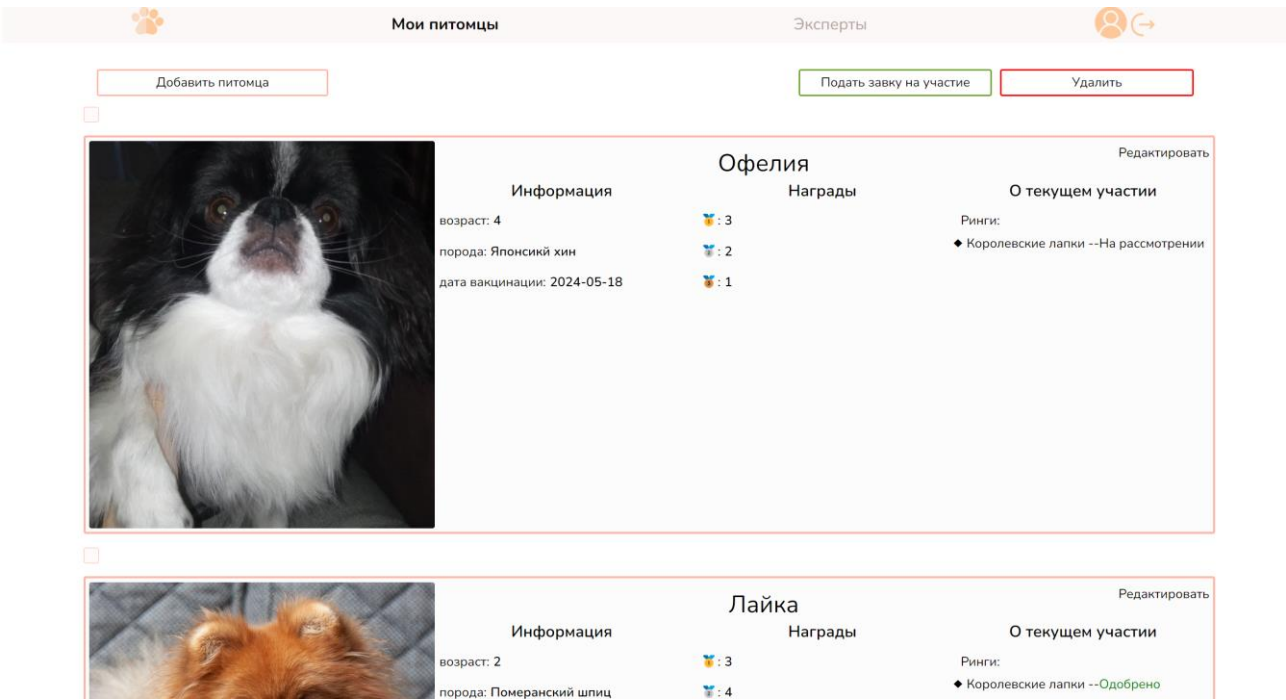


рис. 21 Страница «Мои питомцы»



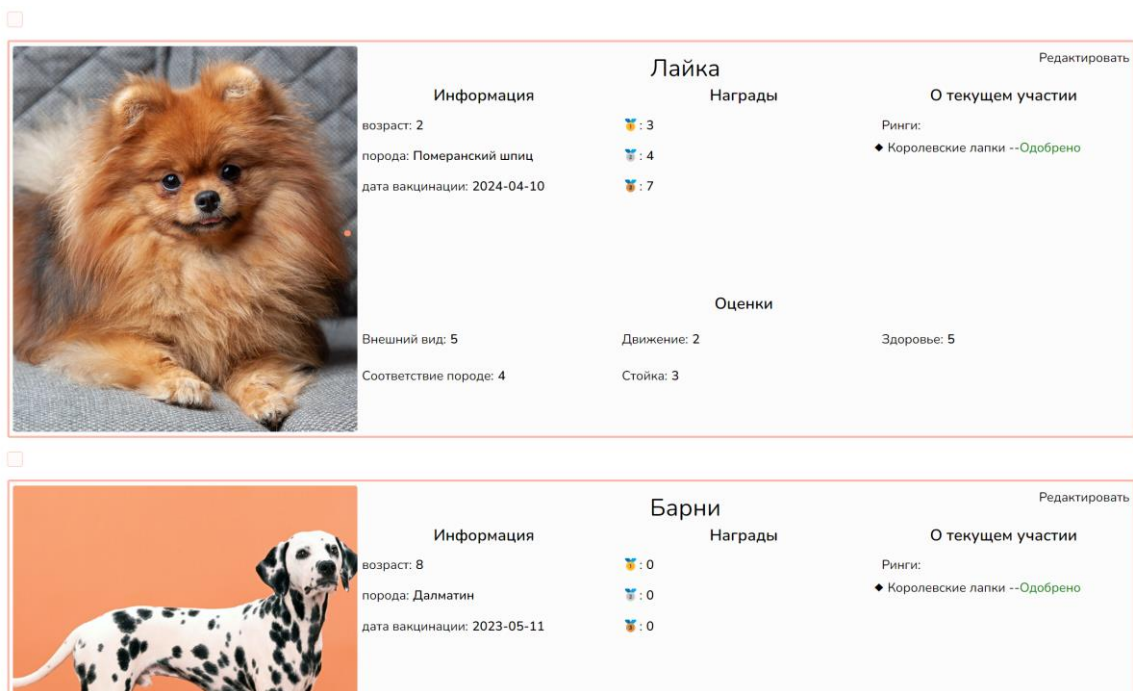


рис. 22 Отображение карточки собаки с оценками

### Листинг 3 – GET-запрос на получение данных о собаках

```
app.get('/dogs', async(req, res) =>{
  const ownerId = req.query.userId;
  try{
    const searchId = parseInt(ownerId);

    const query = {
      text: `SELECT dog.id, dog.name, dog.age, breed.name AS breed,
        vaccination, ring.name AS ring, application.status AS application_status,
        criterion.name AS criteria, mark.value,
        COALESCE(reward_counts.gold_count, 0) AS gold_count,
        COALESCE(reward_counts.silver_count, 0) AS silver_count,
        COALESCE(reward_counts.bronze_count, 0) AS bronze_count,
        ARRAY_AGG(DISTINCT photo.image) AS images
      FROM "dog"
      LEFT JOIN "photo" ON dog.id = photo.dog_id
      LEFT JOIN "breed" ON breed.id = dog.breed_id
      LEFT JOIN "application" ON application.dog_id = dog.id
      LEFT JOIN "ring" ON application.ring_id = ring.id
      LEFT JOIN "dog_reward" ON dog.id = dog_reward.dog_id
      LEFT JOIN "mark" ON mark.dog_id = dog.id
      LEFT JOIN "criterion" ON mark.criterion_id = criterion.id
      LEFT JOIN (
        SELECT
          dog_id,
          SUM(CASE WHEN reward_id = 1 THEN count ELSE 0 END) AS gold_count,
          SUM(CASE WHEN reward_id = 2 THEN count ELSE 0 END) AS silver_count,
          SUM(CASE WHEN reward_id = 3 THEN count ELSE 0 END) AS bronze_count
        FROM "dog_reward"
```

```

        GROUP BY dog_id
    ) AS reward_counts ON dog.id = reward_counts.dog_id
    WHERE dog.user_id = $1
    GROUP BY dog.id, dog.name, dog.age, breed.name, vaccination, ring.name,
    application.status, criterion.name, mark.value, gold_count, silver_count,
    bronze_count`;
    values: [searchId],
  };
  const result = await pool.query(query);
  const rows = result.rows;
  const dogs = {};

  rows.forEach(row => {
    if (!dogs[row.id]) {
      dogs[row.id] = {
        id: row.id,
        name: row.name,
        age: row.age,
        breed: row.breed,
        club: row.club,
        vaccination: row.vaccination,
        rings: [],
        gold_count: row.gold_count,
        silver_count: row.silver_count,
        bronze_count: row.bronze_count,
        images: row.images,
        marks: []
      };
    }

    if (row.criteria && row.value && row.application_status === 'Одобрено') {
      dogs[row.id].marks.push({
        criteria: row.criteria,
        value: row.value
      });
    }

    if (row.ring && row.application_status && !dogs[row.id].rings.some(r =>
r.ring === row.ring)) {
      dogs[row.id].rings.push({
        ring: row.ring,
        status: row.application_status
      });
    }
  });

  const data = Object.values(dogs);

  res.json(data);
} catch (error) {

```

```
console.error('Error searching in database:', error);
res.status(500).json({ error: 'Internal server error' });
}
})
```

Этот код представляет собой обработчик маршрута `/dogs` в Express.js, который используется для получения информации о собаках, принадлежащих определенному пользователю.

Маршрут `/dogs` ожидает параметр `userId` в строке запроса, который идентифицирует владельца собак.

Параметр `userId` преобразуется в целое число `searchId` для использования в запросе к базе данных.

Создается SQL-запрос, который выбирает данные о собаках, принадлежащих пользователю с идентификатором `searchId`. Запрос включает информацию о породе собаки, вакцинации, рингах, статусе заявки, критериях, оценках и наградах, а также массив изображений собак.

Запрос выполняется с использованием пула подключений к базе данных, и результат сохраняется в переменной `rows`.

После происходит преобразование данных. Перебираются все строки результата запроса, и данные группируются по идентификатору собаки. Создается объект `dogs`, где ключами являются идентификаторы собак, а значениями - объекты с информацией о собаках. Для каждой собаки также создаются массивы `marks` для оценок и `rings` для рингов.

Наиболее подробно с кодом можно ознакомиться в *Приложении 1*.

### 3.4 Разработка добавления собаки на странице «Мои питомцы»

В рамках проекта реализовать добавление собаки. Для этого реализована форма «Добавить питомца» и отправка POST-запроса с ее дальнейшей обработкой.

Все фотографии, загруженные пользователем, кодируются в Base64.

А выбор породы происходит с предварительным запросом к базе данных

на получение всех пород и с помощью Select библиотеки React.js.

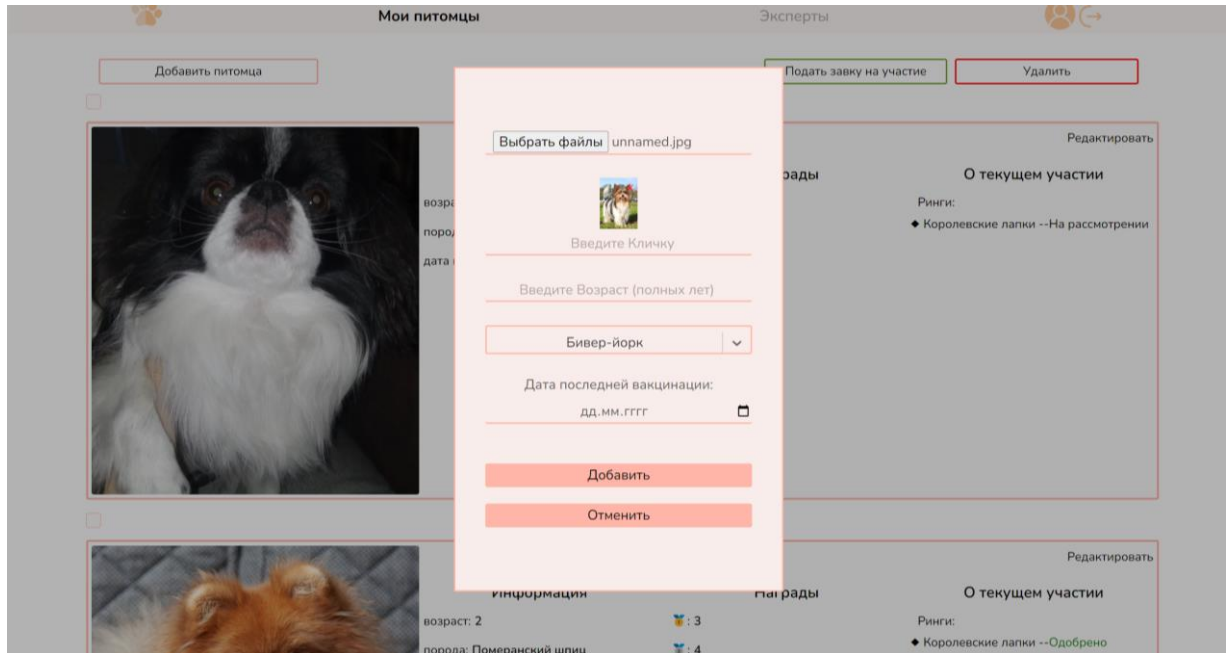


рис. 23 Форма добавления собаки с выбранной фотографией и породой

#### Листинг 4 – Функция добавления картинки

```
const [images, setImages] = useState([]);
const [error, setError] = useState('');
const handleImageUpload = (event) => {
  const files = Array.from(event.target.files);
  const totalImages = images.length + files.length;
  if (totalImages > 5) {
    setError('Вы не можете загрузить больше 5 изображений, картинки не будут загружены.');
```

```
    return;
  } else {
    setError('');
  }
  const fileReaders = files.map((file) => {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onload = () => resolve(reader.result);
      reader.onerror = (error) => reject(error);
      reader.readAsDataURL(file);
    });
  });
  Promise.all(fileReaders)
    .then((base64Images) => {
      setImages((prevImages) => [...prevImages, ...base64Images]);
    })
    .catch((error) => {
```

```

        console.error('Error converting images to Base64', error);
    });
};
const handleImageRemove = (index) => {
    setImages((prevImages) => prevImages.filter((_, i) => i !== index));
};

```

Этот код представляет собой компонент React, который позволяет пользователю загружать изображения, ограничивая их количество до пяти. Компонент использует хук состояния (useState) для управления массивом изображений и сообщением об ошибке. Используются два состояния: `images` для хранения массива изображений в формате Base64 и `error` для хранения сообщения об ошибке.

Функция **handleImageUpload** вызывается при выборе файлов для загрузки. Она сначала проверяет, не превышает ли общее количество изображений (уже загруженных плюс новых) пяти. Если превышает, устанавливается сообщение об ошибке и функция возвращается, предотвращая загрузку. Если нет, сообщение об ошибке сбрасывается.

Для каждого выбранного файла создается объект **FileReader**, который считывает файл как Data URL (Base64). Чтение файлов происходит асинхронно, поэтому используется **Promise.all** для ожидания завершения всех операций чтения. После успешного чтения, массив Base64-строк добавляется к текущему массиву изображений с помощью **setImages**. Если чтение файла завершается с ошибкой, ошибка выводится в консоль.

Функция **handleImageRemove** удаляет изображение по указанному индексу.

*Листинг 5 – Обработка POST-запроса на сервере*

```

app.post('/new_dog', async (req, res) => {
    try {
        const { name, owner_id, age, vaccination, breed_id, images } = req.body;
        const dogExists = await pool.query('SELECT * FROM "dog" WHERE name = $1 AND
user_id = $2 AND breed_id = $3', [name, owner_id, breed_id]);
        if (dogExists.rows.length > 0) {
            return res.status(400).json({ error: 'Dog with this name and breed already
exists for this owner' });
        }
    }
}

```

```

    const dogId = (await pool.query('SELECT MAX(id) AS max_id FROM
"dog"')).rows[0].max_id + 1;
    const insertDogQuery = 'INSERT INTO "dog" (id, name, age, vaccination,
user_id, breed_id) VALUES ($1, $2, $3, $4, $5, $6) RETURNING id';
    const insertDogValues = [dogId, name, parseInt(age), vaccination,
parseInt(owner_id), parseInt(breed_id)];
    await pool.query(insertDogQuery, insertDogValues);

    for (const base64Image of images) {
        const insertPhotoQuery = 'INSERT INTO "photo" (dog_id, image) VALUES ($1,
$2)';
        const insertPhotoValues = [dogId, base64Image];
        await pool.query(insertPhotoQuery, insertPhotoValues);
    }

    res.status(201).json({ message: 'Dog successfully registered' });
} catch (error) {
    console.error('Error registering dog:', error);
    res.status(500).json({ error: 'Internal server error' });
}
});

```

Этот код реализует обработчик маршрута POST  `'/new_dog'`  в Express.js, который добавляет новую собаку в базу данных.

Из тела запроса извлекаются данные о собаке, включая имя, идентификатор владельца, возраст, вакцинацию, идентификатор породы и массив изображений в формате Base64.

Выполняется запрос к базе данных, чтобы проверить, существует ли уже собака с таким именем и породой у данного владельца. Если такая собака найдена, возвращается статус 400 с сообщением об ошибке.

Новый идентификатор собаки вычисляется как максимальный существующий идентификатор плюс один. Затем выполняется запрос на вставку новой записи в таблицу  `'dog'` . После успешной вставки возвращается идентификатор новой записи.

Перебирается массив изображений и для каждого изображения выполняется запрос на вставку в таблицу  `'photo'` , связанного с идентификатором новой собаки.

Наиболее подробно с кодом можно ознакомиться в *Приложении 1*.

### 3.5 Реализация удаления собаки на странице «Мои питомцы»

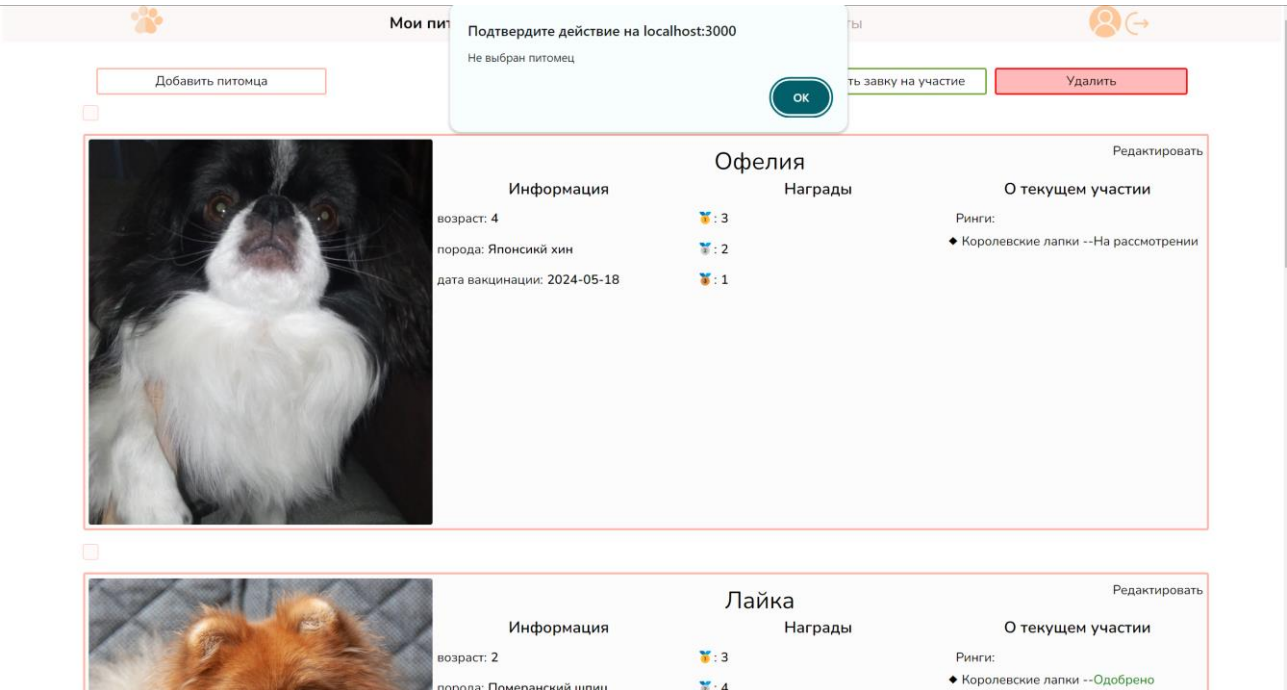


рис. 24 Уведомление об отсутствии выбранных питомцев для удаления

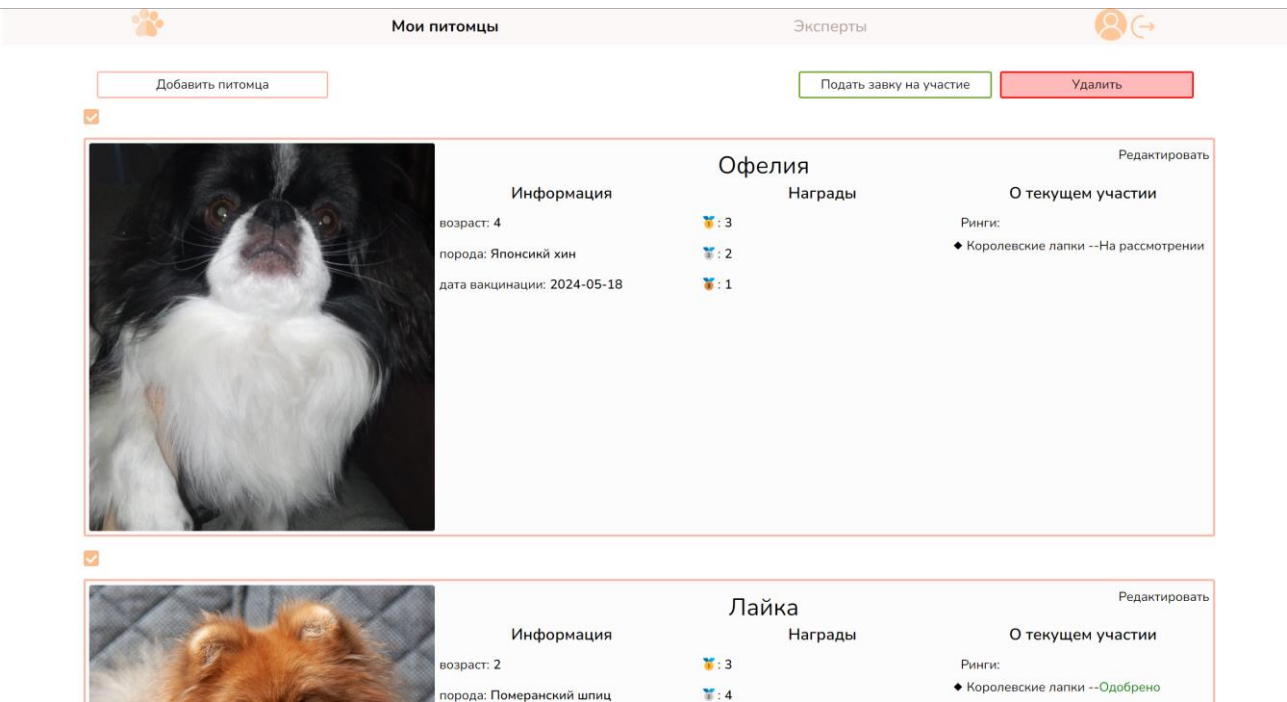


рис. 25 Выбор питомцев и удаление их

#### Листинг 6 – Отправка DELETE-запроса на сервер

```
const [checkedItems, setCheckedItems] = useState([]);
const handleCheckBoxChange = (index) => {
  setCheckedItems(prevState => {
    if (prevState.includes(index)) {
```



```

        return prevState.filter(item => item !== index);
      } else {
        return [...prevState, index];
      }
    });
  });
};

const handleDelete=async (checkedItems)=>{
  async function deleteData(id){
    try{
      await axios.delete(`http://localhost:8082/delete_dog?id=${id}` )
    }catch(error){
      console.log(error)
    }
  }
  if (checkedItems.length !== 0){
    await Promise.all(checkedItems.map(id => deleteData(id)));
    window.location.reload();
  }
  else{alert('Не выбран питомец')}
};

```

Этот код реализует функциональность для обработки изменений состояния чекбоксов и удаления выбранных элементов в React.

**checkedItems** — это состояние, которое хранит массив индексов или идентификаторов выбранных собак. Функция **handleCheckBoxChange** обновляет состояние **checkedItems** при изменении состояния чекбокса.

Функция **handleDelete** обрабатывает удаление выбранных элементов:

- вложенная функция **deleteData**: принимает идентификатор и отправляет запрос на удаление этого элемента с сервера;
- основная функция: проверяет, есть ли выбранные элементы. Если массив `'checkedItems'` не пуст, он вызывает `'deleteData'` для каждого элемента и ждет завершения всех запросов с помощью `'Promise.all'`. Затем обновляет страницу, чтобы отобразить изменения.

### *Листинг 7 – Обработка DELETE-запроса*

```

app.delete('/delete_dog', async(req, res) => {
  const dogId = req.query.id
  try {
    await pool.query('BEGIN');

```



```

const deleteDogRewardQuery = {
  text: `DELETE FROM "dog_reward" WHERE dog_id = $1`,
  values: [parseInt(dogId)]
};
await pool.query(deleteDogRewardQuery);

const deleteApplicationQuery = {
  text: `DELETE FROM "application" WHERE dog_id = $1`,
  values: [parseInt(dogId)]
};
await pool.query(deleteApplicationQuery);

const deletePhotoQuery = {
  text: `DELETE FROM "photo" WHERE dog_id = $1`,
  values: [parseInt(dogId)]
};
await pool.query(deletePhotoQuery);

const deleteMarkQuery = {
  text: `DELETE FROM "mark" WHERE dog_id = $1`,
  values: [parseInt(dogId)]
};
await pool.query(deleteMarkQuery);

const deleteDogQuery = {
  text: `DELETE FROM "dog" WHERE id = $1`,
  values: [parseInt(dogId)]
};
await pool.query(deleteDogQuery);

await pool.query('COMMIT');
res.status(200).json({ message: 'Dog deleted successfully' });

} catch (error) {
  await pool.query('ROLLBACK')
  console.error('Error deleting dog:', error);
  res.status(500).json({ error: 'Internal server error' });
}
})

```

Код реализует обработчик DELETE-запроса для удаления собаки и всех связанных данных из базы данных.

Идентификатор собаки извлекается из параметра запроса `id`.

Начинается транзакция (**await pool.query('BEGIN')**) для обеспечения атомарности операции удаления. Если что-то пойдет не так, транзакция будет откатана. Далее осуществляется удаление связанных таблиц. Каждое удаление

связано с конкретной таблицей, и все запросы используют один и тот же `dog\_id`, который передается в параметрах запроса. После успешного выполнения всех запросов транзакция фиксируется (`await pool.query('COMMIT')`) и возвращается ответ с сообщением об успешном удалении. Если в процессе выполнения любого из запросов возникает ошибка, транзакция откатывается (`await pool.query('ROLLBACK')`), ошибка логируется, и возвращается ответ с кодом 500 и сообщением об ошибке.

### 3.6 Разработка страницы «Эксперты» для владельца собаки

На данной странице реализован просмотр экспертов, зарегистрированных на сайте. Для этого на сервер осуществляется GET-запрос, а полученные данные преобразовываются в карточки экспертов.

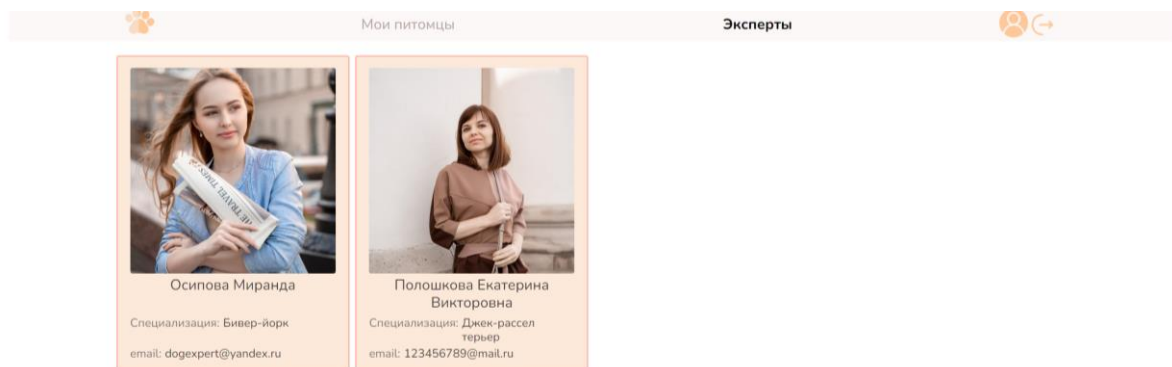


рис. 26 Страница «Эксперты» для владельца собак

Листинг 8 – React компонент для страницы

```
import React, {useEffect, useState} from 'react'
import style from './ExpertsPage.module.css';
import HeaderUser from '../../components/HeaderUser';
import { useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import ExpertUI from '../../components/ExpertUI';
import axios from 'axios';
const ExpertsPage = () => {
  const navigate=useNavigate();
```

```

    useEffect(()=>{
      const token = localStorage.getItem('token');
      if(token){
        const decodedToken = jwtDecode(token);
        if(decodedToken.role_id !== "2") navigate("/forbidden");
      }
    })
    return (
      <div className='page'>
        <HeaderUser />
        <main>
          <Experts />
        </main>
      </div>
    )
  }
}

export default ExpertsPage

const Experts =()=>{
  const [experts, setExperts] = useState([])
  useEffect(()=>{
    const fetchData = async () => {
      try {
        const data = await axios.get(`http://localhost:8082/experts`);
        setExperts(data.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchData();

  }, [])
  return(
    <div className={style.container}>
      <div className={style.main_content}>
        {experts.length > 0 ? experts.map(expert => <ExpertUI
expert={expert} />)
        :
        <div style={{width: '100%', textAlign: 'center', fontSize:
'20px', color: '#B4A59F', marginTop:'10px'}}>В базе нет ни одного эксперта</div>
      </div>
    </div>
  )
}

```

Аутентификация и авторизация: В компоненте ExpertsPage проверяется,

является ли пользователь владельцем собаки, используя токен из localStorage. Если роль пользователя не совпадает, он перенаправляется на страницу "forbidden".

Получение данных: компонент Experts выполняет запрос на сервер для получения данных об экспертах и сохраняет их в состоянии experts.

Отображение данных: если список экспертов не пуст, он отображается с использованием компонента ExpertUI. В противном случае отображается сообщение об отсутствии экспертов.

### 3.7 Разработка страницы «Участники» для администратора (организатора выставки)

В ходе разработки данной страницы был реализован GET-запрос на сервер для получения данных о заявках пользователей (владельцев собак) с дополнительной информацией. Также реализован функционал одобрения/отклонения заявок и добавления нового участника. Если участника добавляет администратор, то заявка автоматически получает статус ‘Одобрено’ с возможностью дальнейшего редактирования.

Участники							
Участники				Ринги		Эксперты	
Добавить нового участника				Одобрить заявку		Отклонить заявку	
Кличка собаки	Порода	Возраст	ФИО хозяина	Ринг	Специализация ринга	Кол-во наград	Статус заявки
<input type="checkbox"/> Миранда	Бивер-йорк	1	Полошкова В. Ю.	Эдем	<ul style="list-style-type: none"> <li>Немецкая овчарка</li> <li>Родезийский риджбек</li> <li>Шарпей</li> </ul>		Отклонено
<input type="checkbox"/> Барни	Далматин	8	Полошкова В. Ю.	Королевские лапки	<ul style="list-style-type: none"> <li>Йоркширский терьер</li> <li>Папильон</li> <li>Померанский шпиц</li> <li>Чихуахуа</li> <li>Японский хин</li> </ul>	0	Одобрено
<input type="checkbox"/> Ириска	Бивер-йорк	3	Полошкова В. Ю.	Королевские лапки	<ul style="list-style-type: none"> <li>Йоркширский терьер</li> <li>Папильон</li> <li>Померанский шпиц</li> <li>Чихуахуа</li> <li>Японский хин</li> </ul>		На рассмотрении
<input type="checkbox"/> Питер	Немецкая овчарка	6	Полошкова В. Ю.	Собачий Парк	<ul style="list-style-type: none"> <li>Австралийская овчарка</li> <li>Далматин</li> <li>Доберман</li> <li>Пудель</li> <li>Швейцарская овчарка</li> </ul>	0	Одобрено
<input type="checkbox"/> Офелия	Японский хин	4	Полошкова В. Ю.	Королевские лапки	<ul style="list-style-type: none"> <li>Йоркширский терьер</li> <li>Папильон</li> <li>Померанский шпиц</li> <li>Чихуахуа</li> <li>Японский хин</li> </ul>	6	На рассмотрении
<input type="checkbox"/> Лайка	Померанский шпиц	2	Полошкова В. Ю.	Королевские лапки	<ul style="list-style-type: none"> <li>Йоркширский терьер</li> <li>Папильон</li> <li>Померанский шпиц</li> <li>Чихуахуа</li> <li>Японский хин</li> </ul>	14	Одобрено

рис. 27 Страница «Участники» для администратора

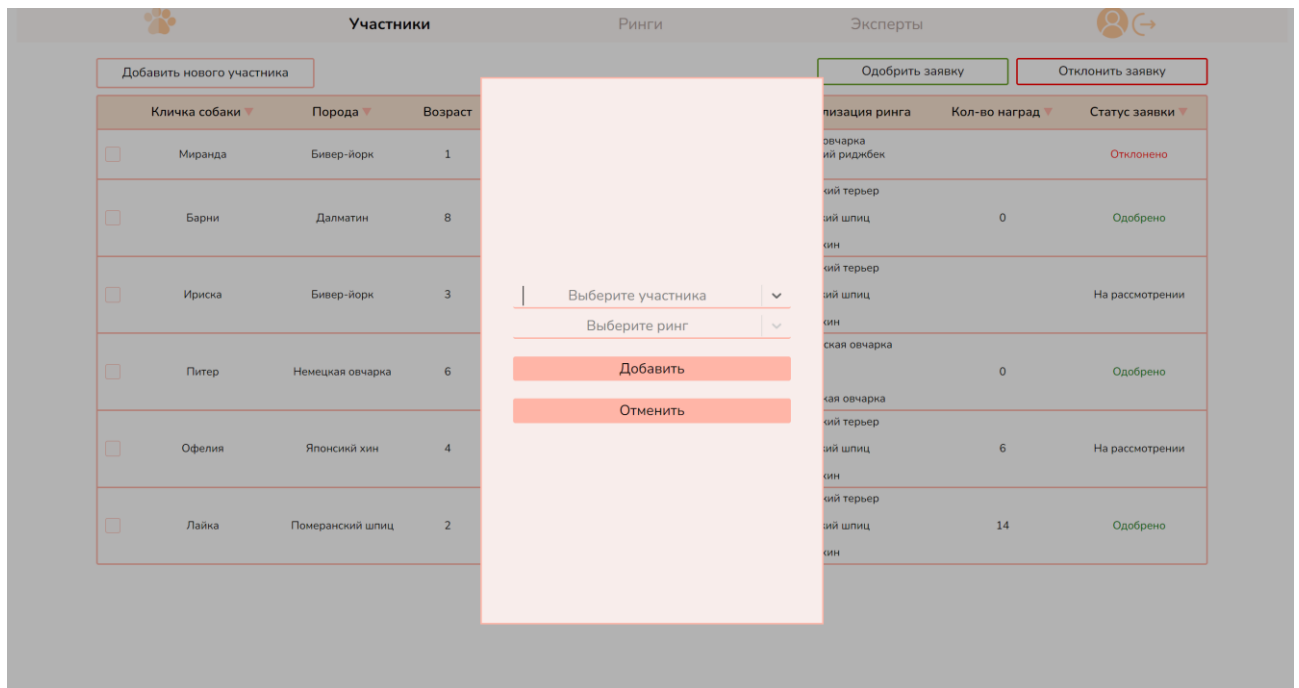


рис. 28 Окно добавления нового участника

### Листинг 9 – Код для обработки GET-запроса

```
app.get('/admin/participants', async(req, res) =>{
  try{
    const query={
      text: `SELECT application.id, dog.name AS nickname, breed.name AS breed,
        dog.age, CONCAT(u.surname, ' ', SUBSTRING(u.name, 1, 1), '. ',
SUBSTRING(u.patronymic, 1, 1), '.') AS fio,
        ring.name AS ring, specialization,
        SUM(dog_reward.count) AS reward_cnt, application.status FROM "application"
LEFT JOIN "dog" ON dog.id = application.dog_id
LEFT JOIN "breed" ON breed.id = dog.breed_id
LEFT JOIN "user" u ON u.id = dog.user_id
LEFT JOIN "ring" ON ring.id = application.ring_id
LEFT JOIN "dog_reward" ON dog.id = dog_reward.dog_id
LEFT JOIN (
  SELECT
    ring.id AS ring_id,
    ARRAY_AGG(DISTINCT breed.name) AS specialization
  FROM "ring"
  LEFT JOIN "ring_breed" ON ring.id = ring_breed.ring_id
  LEFT JOIN "breed" ON breed.id = ring_breed.breed_id
  GROUP BY ring.id
) AS ring_specializations ON ring.id = ring_specializations.ring_id
GROUP BY application.id, nickname, breed, dog.age, fio, ring.name,
specialization, application.status`
    }
    const result = await pool.query(query);
    res.json(result.rows);
  }catch(error){
    res.status(500).json({ error: 'Internal server error' });
  }
})
```

```
}  
})
```

Это обработчик маршрута `'GET'` Express предназначен для получения списка участников (питомцев), зарегистрированных на различных соревнованиях.

1) Обработка запроса.

- Маршрут `"/admin/participants"` обрабатывает GET-запросы.
- `'async(req, res) => {...}'` определяет асинхронную функцию обратного вызова, которая будет выполнена при получении запроса.

2) Запрос к базе данных.

- Используется объект `'query'`, который содержит SQL-запрос для выборки данных из базы данных.
- SQL-запрос выполняет JOIN операции для объединения таблиц `'application'`, `'dog'`, `'breed'`, `'user'`, `'ring'` и `'dog_reward'`.
- В запросе используются агрегатные функции `'SUM'` и `'GROUP BY'` для агрегации данных.
- Результаты запроса будут содержать информацию о питомцах, включая их клички, породы, возрасты, ФИО владельцев, названия рингов, специализации рингов, количество полученных наград и статусы заявок.

*Листинг 10 – Окно добавления нового участника*

```
import React, { useEffect, useState } from 'react'  
import { useForm } from 'react-hook-form';  
import axios from 'axios';  
import Select from 'react-select';  
const ContainerStyle = {  
  margin: '0',  
  padding: '0',  
  top: '0',  
  left: '0',  
  position: 'fixed',  
  zIndex: '1',  
  width: '100%',  
  height: '100%',  
  display: 'flex',  
  flexDirection: 'column',  
  justifyContent: 'center',
```

```

    alignItems: 'center',
    background: 'rgb(0, 0, 0, 0.3)'
  }

  const FormStyle = {
    width: '429px',
    height: '685px',
    backgroundColor: '#F8EDEB',
    border: '2px solid rgb(255, 181, 167)',
    marginTop: '0'
  }

  const customStyles = {
    control: (provided) => ({
      ...provided,
      width: '350px',
      height: '31px',
      border: 'none',
      borderBottom: '2px solid rgb(255, 181, 167)',
      borderRadius: '3px',
      background: 'rgb(248, 237, 235)',
      boxShadow: 'none',
      color: 'rgb(98, 90, 87)',
      textAlign: 'center',
      fontSize: '18px',
      fontWeight: '400',
      '&:hover': {
        border: '2px solid rgb(255, 181, 167)',
      },
      '&::placeholder': {
        color: 'rgb(180, 165, 159)',
      }
    }),
    option: (provided, state) => ({
      ...provided,
      margin: '0px',
      padding: '10px',
      backgroundColor: state.isSelected ? 'rgb(248, 237, 235)' : state.isFocused
? '#FFB5A7' : 'rgb(248, 237, 235)',
      color: state.isSelected ? 'rgb(98, 90, 87)' : 'rgb(98, 90, 87)',
      '&:hover': {
        backgroundColor: state.isSelected ? '#FFB5A7' : '#FFB5A7',
      },
    }),
    input: (provided) => ({
      ...provided,
    }),
    menu: (provided) => ({
      ...provided,
      marginTop: 0,

```

```

    })),
  };

const AddParticipant = () => {
  const {handleSubmit} = useForm();
  const [dogs, setDogs] = useState([]);
  const [rings, setRings] = useState([]);
  const [selectedOptionRing, setSelectedOptionRing] = useState('');
  const [selectedOptionDog, setSelectedOptionDog] = useState('');
  useEffect(()=>{
    const fetchDataDog = async () => {
      try {
        const data = await axios.get(`http://localhost:8082/admin/dogs`);
        setDogs(data.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchDataDog();
  }, [])
  useEffect(()=>{
    const fetchDataRing = async () => {
      try {
        const data = await
axios.get(`http://localhost:8082/admin/user_rings`);
        setRings(data.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchDataRing();
  }, [])
  const handleSelectChangeRing = (selected) => {
    setSelectedOptionRing(selected);
  };
  const handleSelectChangeDog = (selected) => {
    setSelectedOptionDog(selected);
  };
  const handleReload = ()=>{
    window.location.reload()
  }
  const onSubmit = async(data) => {
    const new_data = {
      dog_id: selectedOptionDog.value,
      ring_id: selectedOptionRing.value,
      status: 'Одобрено'
    }
    try{

```



```

        await axios.post('http://localhost:8082/admin/new_application',
new_data)
        window.location.reload();
    }
    catch (error){
        console.error('Error searching:', error);
    }
};
}
return (
    <div style={ContainerStyle}>
        <form style={FormStyle} onSubmit={handleSubmit(onSubmit)}>
            <Select
                styles={customStyles}
                value={selectedOptionDog}
                onChange={handleSelectChangeDog}
                options={dogs}
                placeholder="Выберите участника"
                isSearchable={true}
                noOptionsMessage={() => "Участник не найден"}
            />
            <Select
                styles={customStyles}
                value={selectedOptionRing}
                onChange={handleSelectChangeRing}
                options={rings}
                placeholder="Выберите ринг"
                isSearchable={true}
                noOptionsMessage={() => "Ринг не найден"}
            />
            <button type="submit">Добавить</button>
            <button onClick={handleReload}>Отменить</button>
        </form>
    </div>
)
}
}

export default AddParticipant

```

Компонент **AddParticipant** в React отвечает за добавление нового участника в ринг. В начале происходит инициализация состояния. Для хранения списков собак и рингов используется **useState**, а также для хранения выбранных опций.

Далее используется **useEffect** для выполнения HTTP-запросов к серверу для получения данных о собаках и рингах при монтировании компонента.

- **fetchDataDog** и **fetchDataRing** выполняют запросы к соответствующим

эндпоинтам и сохраняют данные в состоянии.

После происходит обработка выбранных опций в выпадающих списках, **handleSelectChangeRing** и **handleSelectChangeDog** обновляют состояние при выборе опций в выпадающих списках.

Функция **handleSubmit** из `react-hook-form` используется для обработки отправки формы. **onSubmit** формирует объект `new\_data` с выбранными значениями и отправляет его на сервер с помощью **axios**.

Компонент рендерит форму с двумя выпадающими списками для выбора собаки и ринга. Также есть две кнопки: одна для отправки формы, другая для отмены действия.

#### *Листинг 11 – Получение списка рингов для окна добавления*

```
app.get('/admin/user_rings', async(req, res) =>{
  try{
    const query={
      text: `SELECT ring.id AS value,
      ring.name || ' (' || STRING_AGG(breed.name, ', ') || ')' AS label
      FROM "ring"
      LEFT JOIN "ring_breed" ON ring.id = ring_breed.ring_id
      LEFT JOIN "breed" ON breed.id = ring_breed.breed_id
      GROUP BY ring.id;`
    }
    const result = await pool.query(query);
    res.json(result.rows);
  }catch(error) {
    console.error('Error searching in database:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
})
```

##### 1) Обработка запроса:

- Маршрут /admin/user\_rings обрабатывает GET-запросы.
- Обработчик асинхронный (async(req, res) => {...}), что позволяет выполнять асинхронные операции, такие как запросы к базе данных.

##### 2) Запрос к базе данных:

- Создается объект query, содержащий SQL-запрос для выборки данных о рингах и их специализациях.

- В SQL-запросе используются операции JOIN для объединения таблиц ring, ring\_breed и breed.

- С помощью агрегатной функции STRING\_AGG происходит объединение названий пород в одну строку с разделителем «,».

- Результаты запроса содержат идентификаторы рингов и их названия с указанием специализаций пород, например, «Название ринга (Порода1, Порода2, ...)».

### 3.8 Разработка страницы «Ринги» для администратора (организатора выставки)

Для разработки данной страницы был реализован GET-запрос к базе данных на получение данных о существующих рингах. Вывод полученных данных в виде таблицы для наглядности.

А также был реализован функционал создания нового ринга администратором (форма создания и POST-запрос) и функционал удаления ринга (DELETE-запрос по id ринга, выбранных галочками в чекбоксы).



<div>  <span>Участники</span> <span>Ринги</span> <span>Эксперты</span>  </div>				
<div> <div>Создать новый ринг</div> <div>Удалить ринг</div> </div>				
Название ринга ▼	Адрес	Специализация ринга ▼	Эксперты ринга	Карточка ринга
<input type="checkbox"/> Королевские лапки	г. Москва, ул. Гагарина, д. 34	<ul style="list-style-type: none"> <li>♦ Японский хин</li> <li>♦ Чихуахуа</li> <li>♦ Померанский шпиц</li> <li>♦ Йоркширский терьер</li> <li>♦ Папильон</li> </ul>		Открыть карточку
<input type="checkbox"/> Мир Сладостей	г. Москва, ул. Родинка, д. 1	<ul style="list-style-type: none"> <li>♦ Джек-рассел терьер</li> <li>♦ Японский хин</li> <li>♦ Ротвейлер</li> <li>♦ Йоркширский терьер</li> <li>♦ Шiba-ину</li> </ul>	♦ Полошкова Е В	Открыть карточку
<input type="checkbox"/> Собачий Парк	г. Москва, ул. Волковская, д. 7	<ul style="list-style-type: none"> <li>♦ Далматин</li> <li>♦ Пудель</li> <li>♦ Доберман</li> <li>♦ Швейцарская овчарка</li> <li>♦ Австралийская овчарка</li> <li>♦ Доберман</li> <li>♦ Швейцарская овчарка</li> <li>♦ Австралийская овчарка</li> <li>♦ Далматин</li> <li>♦ Пудель</li> </ul>	♦ Осипова М ♦ Полошкова Е В	Открыть карточку
<input type="checkbox"/> Эдем	г. Москва, ул. Лебедева, д. 48	<ul style="list-style-type: none"> <li>♦ Родезийский риджбек</li> <li>♦ Немецкая овчарка</li> <li>♦ Шарпей</li> </ul>	♦ Полошкова Е В	Открыть карточку

рис. 29 Страница «Ринги» для администратора

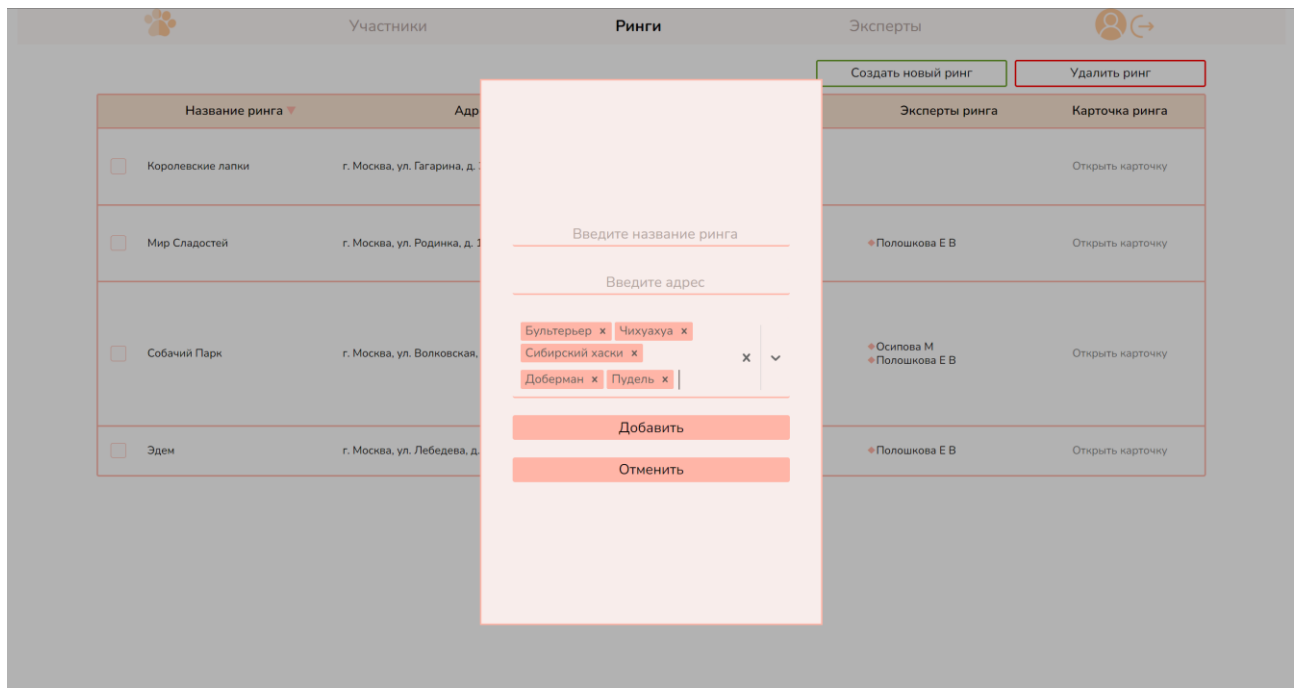


рис. 30 Форма создания нового ринга с множественным выбором пород (специализации)

### Листинг 12 – Форма создания нового ринга

```
const AddRing = () => {
  const {register, handleSubmit} = useForm();
  const [breeds, setBreed] = useState(['']);
  const [selectedBreeds, setSelectedBreeds] = useState([]);
  useEffect(() => {
    const fetchData = async () => {
      try {
        const data = await axios.get(`http://localhost:8082/breeds`);
        setBreed(data.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };
    fetchData();
  }, []);
  const handleReload = () => {
    window.location.reload();
  };

  const handleBreedsChange = (selected) => {
    setSelectedBreeds(selected || []);
  };

  const onSubmit = async (data) => {
    const selectedIds = selectedBreeds.map(breed => breed.value);

    const formData = {
```

```

        name: data.name,
        address: data.address,
        breed_id: selectedIds
    };
    console.log(formData)
    try {
        const response = await
axios.post('http://localhost:8082/admin/create_ring', formData);
        console.log('Server response:', response.data);
        window.location.reload();
    } catch (error) {
        console.error('Error sending form data:', error);
    }
    console.log(formData.selectedIds)
    };
    return (
        <div style={ContainerStyle}>
            <form style={FormStyle} onSubmit={handleSubmit(onSubmit)}>
                <input type="text" placeholder="Введите название ринга"
{...register("name", {required: true, maxLength: 30})} />
                <input type="text" placeholder="Введите адрес" {...register("address",
{required: true})} />
                <MultiSelectCheckbox options={breeds} onChange={handleBreedsChange}
placeholder={'Специализация...'} noOption={'Порода не найдена'} />
                <button type="submit">Добавить</button>
                <button onClick={handleReload}>Отменить</button>
            </form>
        </div>
    )
}

export default AddRing

```

```

const Option = (props) => {
    return (
        <components.Option {...props}>
            <input
                style={{width: '18px', height: '18px', paddingRight: '6px', marginTop:
'28px'}}
                type="checkbox"
                checked={props.isSelected}
                onChange={() => null}
            />{ " "}
            <label>{props.label}</label>
        </components.Option>
    );
};

const MultiSelectCheckbox = ({ options, onChange, placeholder, noOption }) => {
    return (

```

```

    <Select
      closeMenuOnSelect={false}
      hideSelectedOptions={false}
      isMulti
      onChange={onChange}
      options={options}
      styles={customStyles}
      components={{ Option }}
      placeholder={placeholder}
      noOptionsMessage={() => `${noOption}`}
    />
  );
};

export default MultiSelectCheckbox;

```

Компонент **AddRing** использует **useForm** для обработки форм и **axios** для выполнения HTTP-запросов.

**MultiSelectCheckbox** — это пользовательский компонент для выбора нескольких опций с использованием библиотеки react-select.

### *Листинг 13 – Обработка POST-запроса на создание ринга*

```

app.post('/admin/create_ring', async(req, res)=>{
  const { name, address, breed_id } = req.body;
  const existingRing = await pool.query(
    'SELECT * FROM "ring" WHERE name = $1 AND address = $2',
    [name, address]
  );

  if (existingRing.rows.length > 0) {
    return res.status(400).json({ error: 'Ring with the same name and address already exists' });
  }

  const result = await pool.query(
    'INSERT INTO "ring" (name, address) VALUES ($1, $2) RETURNING id',
    [name, address]
  );

  const ringId = result.rows[0].id;
  const insertPromises = breed_id.map(breedId => {
    return pool.query(
      'INSERT INTO "ring_breed" (ring_id, breed_id) VALUES ($1, $2)',
      [ringId, breedId]
    );
  });

```

```

});
try{

    await Promise.all(insertPromises);

    res.status(201).json({ message: 'Ring created successfully', ringId });
} catch(error) {
    console.error('Error executing query', error);
    res.status(500).json({ error: 'Internal server error' });
}
});

```

1) Обработка запроса.

- Маршрут **/admin/create\_ring** обрабатывает POST-запросы.
- Используется асинхронная функция для выполнения асинхронных операций с базой данных.

2) Получение данных из запроса.

- Извлекаются данные из тела запроса: name, address и breed\_id.

breed\_id предполагается массивом идентификаторов пород.

3) Проверка на существование ринга.

- Выполняется запрос к базе данных для проверки, существует ли ринг с тем же именем и адресом.
- Если такой ринг существует, возвращается ответ с кодом 400 и сообщением об ошибке.

4) Вставка нового ринга.

- Если ринг с таким именем и адресом не найден, выполняется вставка нового ринга в таблицу ring.
- Используется запрос с RETURNING id, чтобы получить идентификатор нового ринга.

5) Вставка связей между рингом и породами.

- После получения идентификатора нового ринга, создаются запросы для вставки записей в таблицу ring\_breed, чтобы связать ринг с породами.
- Для каждого идентификатора породы из массива breed\_id создается соответствующий запрос. Все запросы для вставки в таблицу ring\_breed

выполняются параллельно с помощью Promise.all.

### 3.9 Разработка страницы «Эксперты» для администратора (организатора выставки)

Для реализации функционала данной страницы использовались наработки предыдущих страниц. Так как рендеринг страницы происходит аналогично странице «Ринги» с предварительной отправкой GET-запроса. Одобрение/отклонение заявки осуществляется аналогично функционалу страницы «Участники» и добавление нового эксперта аналогично странице «Участники» осуществляется через GET-запросы экспертов и рингов и рендеринга формы из двух селектов.

Участники

Ринги

Эксперты

Добавить нового эксперта

Одобрить заявку

Отклонить заявку

ФИО эксперта	Специализация эксперта	Ринг	Специализация ринга	Статус заявки ▾
<div><input type="checkbox"/></div> <div>Полошкова Екатерина Викторовна</div>	Джек-рассел терьер	Мир Сладостей	<div><div>♦ Джек-рассел терьер</div><div>♦ Йоркширский терьер</div><div>♦ Ротвейлер</div><div>♦ Шиб-ину</div><div>♦ Японский хин</div></div>	Одобрено
<div><input type="checkbox"/></div> <div>Осипова Миранда</div>	Бивер-Йорк	Королевские лапки	<div><div>♦ Йоркширский терьер</div><div>♦ Папильон</div><div>♦ Померанский шпиц</div><div>♦ Чихуахуа</div><div>♦ Японский хин</div></div>	Отклонено
<div><input type="checkbox"/></div> <div>Осипова Миранда</div>	Бивер-Йорк	Собачий Парк	<div><div>♦ Австралийская овчарка</div><div>♦ Далматин</div><div>♦ Доберман</div><div>♦ Пудель</div><div>♦ Швейцарская овчарка</div></div>	Одобрено
<div><input type="checkbox"/></div> <div>Полошкова Екатерина Викторовна</div>	Джек-рассел терьер	Собачий Парк	<div><div>♦ Австралийская овчарка</div><div>♦ Далматин</div><div>♦ Доберман</div><div>♦ Пудель</div><div>♦ Швейцарская овчарка</div></div>	Одобрено
<div><input type="checkbox"/></div> <div>Осипова Миранда</div>	Бивер-Йорк	Мир Сладостей	<div><div>♦ Джек-рассел терьер</div><div>♦ Йоркширский терьер</div><div>♦ Ротвейлер</div><div>♦ Шиб-ину</div><div>♦ Японский хин</div></div>	На рассмотрении
<div><input type="checkbox"/></div> <div>Полошкова Екатерина Викторовна</div>	Джек-рассел терьер	Эдем	<div><div>♦ Немецкая овчарка</div><div>♦ Родзийский риджбек</div><div>♦ Шарпей</div></div>	Одобрено

рис. 31 Страница «Эксперты» для администратора

Листинг 14 – Рендеринг страницы

```
import React, {useEffect, useState} from 'react';
import { useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import HeaderAdmin from '../../components/HeaderAdmin';
import axios from 'axios';
import style from './AdminExpertPage.module.css';
import Checkbox from '../../components/Checkbox';
import AddExpert from '../../components/AddExpert';
```



```

const AdminExpertPage =() =>{
  const navigate=useNavigate();
  useEffect(()=>{
    const token = localStorage.getItem('token');
    if(token){
      const decodedToken = jwtDecode(token);
      if(decodedToken.role_id !== "1") navigate("/forbidden");
    }
  })
  return(
    <div className='page'>
      <HeaderAdmin />
      <main>
        <AdminExpert />
      </main>
    </div>
  )
}

export default AdminExpertPage

const AdminExpert =()=>{
  const [overlay, setOverlay] = useState(false)
  const [experts, setExpert] = useState([])
  useEffect(()=>{
    const fetchData = async () => {
      try {
        const data = await
axios.get(`http://localhost:8082/admin/experts`);
        setExpert(data.data);
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };
    fetchData()
  }, [])
  const [checkedItems, setCheckedItems] = useState([]);

  const handleCheckBoxChange = (index) => {
    setCheckedItems(prevState => {
      if (prevState.includes(index)) {
        return prevState.filter(item => item !== index);
      } else {
        return [...prevState, index];
      }
    });
  };

  const handleReject=async (checkedItems)=>{

```

```

    async function updateData(id){
      try{
        await axios.put(`http://localhost:8082/admin/reject_expert?id=${id}`
)
      }catch(error){
        console.log(error)
      }
    }
    if (checkedItems.length !== 0){
      await Promise.all(checkedItems.map(id => updateData(id)));
      window.location.reload();
    }

    else{alert('Не выбран эксперт')}}
  }
  const handleApprove =async(checkedItems)=>{
    async function updateData(id){
      try{
        await axios.put(`http://localhost:8082/admin/approve_expert?id=${id}`
)
      }catch(error){
        console.log(error)
      }
    }
    if (checkedItems.length !== 0){
      await Promise.all(checkedItems.map(id => updateData(id)));
      window.location.reload();
    }

    else{alert('Не выбран эксперт')}}
  }
  return(
    <div className={style.container}>
      {overlay?<AddExpert /> : <></>}
      <div className={style.btn}>
        <button className={style.participant}
onClick={()=>setOverlay(!overlay)}>Добавить нового эксперта</button>
        <div className={style.menu}>
          <button className={style.add}
onClick={()=>handleApprove(checkedItems)}>Одобрить заявку</button>
          <button className={style.delete} onClick={() =>
handleReject(checkedItems)}>Отклонить заявку</button>
        </div>
      </div>
      <div className={style.table}>
        <table>
          <tr className={style.title}>
            <td></td>

```

```

        <td>ФИО эксперта</td>
        <td>Специализация эксперта</td>
        <td>Ринг</td>
        <td>Специализация ринга</td>
        <td>Статус заявки <img src='/triangle.svg' alt='' /></td>
      </tr>
      {experts.map(el=><tr className={style.line} key={el.id}>
        <td className={style.check}>
          <Checkbox key={el.id}
            index={el.id}
            onChange={handleCheckBoxChange}/>
        </td>
        <td>{el.fio}</td>
        <td>{el.expert_specialization}</td>
        <td>{el.ring}</td>
        <td>{el.ring_specialization.map(e=><div style={{display:
'flex'}}><span style={{color: 'rgb(255, 181, 167)'}}>◆</span> {e} </div>)}</td>
          {el.status === 'Одобрено'? <td style={{color:
'green'}}>{el.status}</td> : el.status === 'Отклонено'? <td style={{color:
'red'}}>{el.status}</td> : <td>{el.status}</td>}
        </tr>)}
      </table>
    </div>
  </div>
)
}

```

При загрузке страницы проверяется наличие токена в **localStorage**. Если токен существует, он декодируется с помощью **jwt-decode**, и проверяется, имеет ли пользователь роль администратора (**role\_id** равен "1"). Если нет, пользователя перенаправляют на страницу **/forbidden**. Если проверка прошла успешно, рендерится компонент **HeaderAdmin** и основной контент **AdminExpert**.

Значения констант для состояний **useState**:

- **overlay**: Отвечает за отображение компонента **AddExpert** для добавления нового эксперта.
- **experts**: Содержит список экспертов, полученный с сервера.
- **checkedItems**: Содержит список ID выбранных экспертов.

Функция **fetchData**, которая делает GET-запрос к серверу для получения списка экспертов и сохраняет его в **experts**.

Функция **handleCheckBoxChange** обновляет состояние **checkedItems** при изменении состояния чекбоксов.

Функции **handleReject** и **handleApprove** выполняют PUT-запросы к серверу для обновления статуса экспертов. Если нет выбранных экспертов, выводится предупреждение.

Компонент рендерит таблицу с данными экспертов, где каждая строка содержит информацию об эксперте, включая ФИО, специализацию, ринг и статус заявки. Используются условные стили для отображения статуса заявки.

#### *Листинг 15 – Обработка PUT-запроса на отклонение заявки*

```
app.put('/admin/reject_expert', async(req, res) =>{
  const expert_ring_id = req.query.id
  try {
    await pool.query('BEGIN');

    const rejectStatusQuery = {
      text: `UPDATE "expert_ring" SET "status" = 'Отклонено' WHERE id = $1`,
      values: [parseInt(expert_ring_id)]
    };
    await pool.query(rejectStatusQuery);

    await pool.query('COMMIT');
    res.status(204).json({ message: 'Status updated successfully' });

  } catch (error) {
    await pool.query('ROLLBACK')
    console.error('Error updating status:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
});
```

HTTP PUT запрос для отклонения эксперта. Он обновляет статус конкретного эксперта в таблице **expert\_ring** и включает обработку ошибок и транзакции.

- **const expert\_ring\_id = req.query.id**: извлечение ID эксперта из строки запроса.
- **await pool.query('BEGIN')**: запуск транзакции, чтобы обеспечить атомарность операции.
- **const rejectStatusQuery**: создание запроса на обновление статуса эксперта в

таблице **expert\_ring** на значение 'Отклонено'.

- **await pool.query(rejectStatusQuery)**: выполнение запроса.
- **await pool.query('COMMIT')**: подтверждение транзакции после успешного выполнения запроса на обновление.

Если произошла ошибка, транзакция откатывается с помощью **await pool.query('ROLLBACK')**, чтобы изменения не были сохранены.

### 3.10 Разработка страницы «Участники» для эксперта

В ходе разработки данной страницы необходимо было реализовать основной функционал, который состоял в просмотре только тех участников, которые получили одобрение по заявке на ринг эксперта. Для этого были созданы GET-запросы, в которых обрабатываются данные условия.

Также реализован функционал оценивания участника (выставление оценки по пяти критериям: стойка, движение, внешний вид, здоровье и соответствие породе). Оценки ограничены 0 и 5.

Участники			Ринги		
Кличка собаки	Порода	Возраст	ФИО хозяина	Ринг	Карточка
Питер	Немецкая овчарка	6	Полошкова В. Ю	Собачий Парк	Открыть карточку
Ириска	Бивер-йорк	3	Полошкова В. Ю	Мир Сладостей	Открыть карточку

рис. 32 Страница «Участники» для эксперта

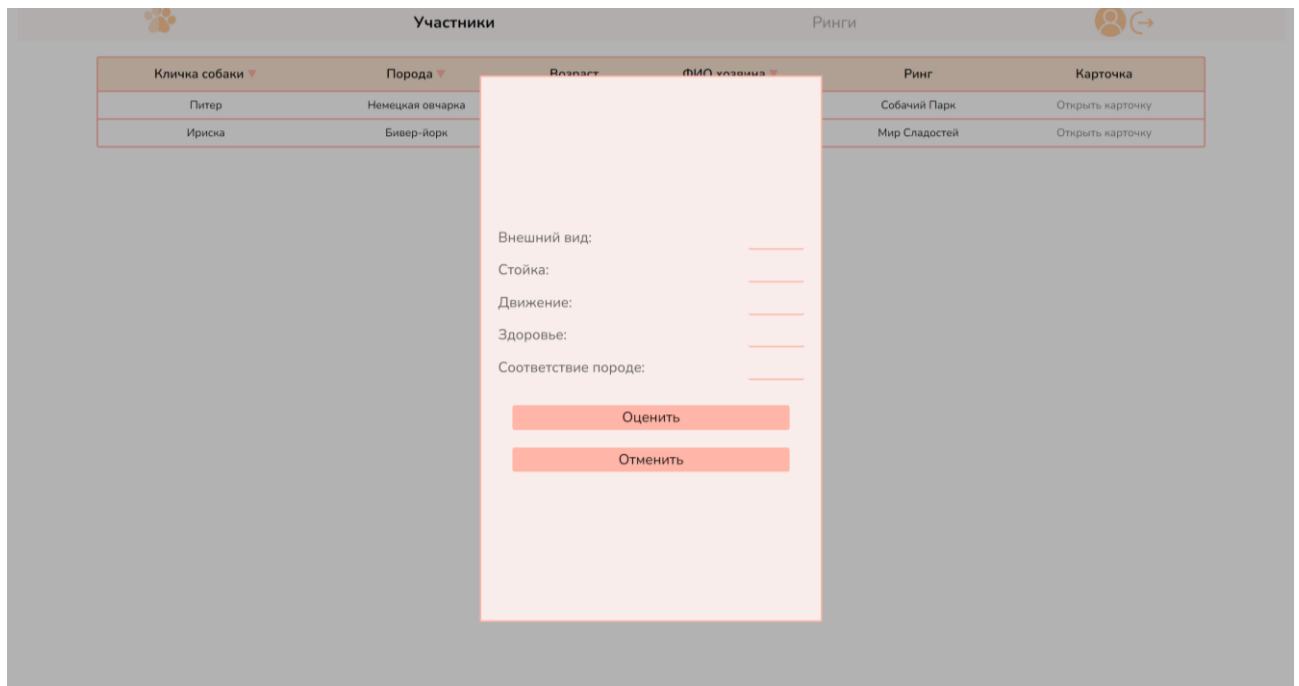


рис. 33 Форма для выставления оценок экспертом

Листинг 16 – Рендеринг формы для выставления оценок

```
const AddMark = ({dog_id, user_id}) => {
  const dogId = dog_id;
  const userId = user_id
  const {register, handleSubmit } = useForm();
  const handleReload =()=>{
    window.location.reload()
  }
  const onSubmit = async(data) => {
    try{
      await
      axios.post(`http://localhost:8082/expert/add_mark?dog_id=${dogId}&user_id=${userId}`, data)
      window.location.reload();
    }
    catch (error){
      console.error('Error searching:', error);
    }
  };
  return (
    <div style={ContainerStyle}>
      <form style={FormStyle} onSubmit={handleSubmit(onSubmit)}>
        <div style={{display: 'flex', justifyContent: 'space-between' , width: '90%'}}>
          <label style={{textAlign: 'left', color: 'rgb(98, 90, 87)', marginTop: '5px', fontSize: '18px'}}>Внешний вид:</label>
          <input style={{width: '70px', marginBottom: '10px'}} type='number'
            required {...register("criterion1", {max: 5, min: 0})} />
        </div>
      </form>
    </div>
  )
}
```

```

        <div style={{display: 'flex', justifyContent: 'space-between' , width:
'90%'}}>
            <label style={{textAlign: 'left', color: 'rgb(98, 90, 87)', marginTop:
'5px', fontSize: '18px'}}>Стойка:</label>
            <input style={{width: '70px', marginBottom: '10px'}} type='number'
required {...register("criterion2", {max: 5, min: 0})} />
        </div>
        <div style={{display: 'flex', justifyContent: 'space-between' , width:
'90%'}}>
            <label style={{textAlign: 'left', color: 'rgb(98, 90, 87)', marginTop:
'5px', fontSize: '18px'}}>Движение:</label>
            <input style={{width: '70px', marginBottom: '10px'}} type='number'
required {...register("criterion3", {max: 5, min: 0})} />
        </div>
        <div style={{display: 'flex', justifyContent: 'space-between' , width:
'90%'}}>
            <label style={{textAlign: 'left', color: 'rgb(98, 90, 87)', marginTop:
'5px', fontSize: '18px'}}>Здоровье:</label>
            <input style={{width: '70px', marginBottom: '10px'}} type='number'
required {...register("criterion4", {max: 5, min: 0})} />
        </div>
        <div style={{display: 'flex', justifyContent: 'space-between' , width:
'90%'}}>
            <label style={{textAlign: 'left', color: 'rgb(98, 90, 87)', marginTop:
'5px', fontSize: '18px'}}>Соответствие породе:</label>
            <input style={{width: '70px', marginBottom: '10px'}} type='number'
required {...register("criterion5", {max: 5, min: 0})} />
        </div>
        <button type="submit">Оценить</button>
        <button onClick={handleReload}>Отменить</button>
    </form>
</div>
)
}

export default AddMark

```

В данном компоненте используются следующие инструменты:

- **useForm**: Хук из библиотеки react-hook-form для управления формами.
- **axios**: Библиотека для выполнения HTTP-запросов.
- **dog\_id** и **user\_id**: Идентификаторы собаки и пользователя, передаваемые

в качестве пропсов.

Функция **handleReload**, которая перезагружает страницу, когда пользователь нажимает кнопку "Отменить".

Асинхронная функция **onSubmit** отправляет данные формы на сервер с использованием **axios.post**. Если запрос успешен, страница перезагружается. В

случае ошибки она логируется в консоль.

Каждый **input** зарегистрирован с помощью **register** из **react-hook-form** и имеет ограничения по минимальному и максимальному значению.

#### *Листинг 17 – Обработка POST-запроса выставления оценок*

```
app.post('/expert/add_mark', async(req, res) =>{
  const { dog_id, user_id } = req.query;
  const { criterion1, criterion2, criterion3, criterion4, criterion5 } =
req.body;

  try {
    const expertResult = await pool.query('SELECT expert.id FROM "expert"
WHERE user_id = $1', [user_id]);
    const expertId = expertResult.rows[0].id;

    await pool.query(
      'INSERT INTO "mark" (dog_id, expert_id, criterion_id, value) VALUES
($1, $2, $3, $4), ($1, $2, $5, $6), ($1, $2, $7, $8), ($1, $2, $9, $10), ($1, $2,
$11, $12)',
      [dog_id, expertId, 1, criterion1, 2, criterion2, 3, criterion3, 4,
criterion4, 5, criterion5]
    );

    res.status(201).json({ message: 'Марки успешно добавлены' });
  } catch (error) {
    console.error('Ошибка при добавлении марок:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
})
```

Извлекаем **dog\_id** и **user\_id** из строки запроса (req.query). Извлекаем оценки (criterion1, criterion2, criterion3, criterion4, criterion5) из тела запроса (req.body). Выполняем запрос к базе данных для получения идентификатора эксперта на основе **user\_id**. Сохраняем **expertId** из результата запроса.

Далее выполняем запрос для вставки пяти строк в таблицу mark, где:

- dog\_id и expert\_id остаются одинаковыми для всех строк.
- criterion\_id варьируется от 1 до 5 для каждой строки.
- value соответствует оценкам по каждому критерию (criterion1, criterion2, criterion3, criterion4, criterion5).



### 3.11 Разработка страницы «Ринги» для эксперта

Данная страница отвечает за просмотр информации о рингах выставки экспертами. Также на данной странице реализован функционал подачи заявления на обслуживания ринга. Однако в одно время можно подать заявление только на один ринг.

Все заявления экспертов и их статусы отображаются в профиле эксперта.

Участники		Ринги	
		<div>Подать заявку</div>	
Название ринга	Адрес	Специализация ринга	
<input type="checkbox"/> Собачий Парк	г. Москва, ул. Волковская, д. 7	♦ Далматин ♦ Пудель ♦ Доберман ♦ Швейцарская овчарка ♦ Австралийская овчарка	
<input type="checkbox"/> Квартет	г. Москва, ул. Ленина, д. 10	♦ Бультерьер ♦ Чихуахуа ♦ Сибирский хаски ♦ Доберман ♦ Пудель	
<input type="checkbox"/> Мир Сладостей	г. Москва, ул. Родинка, д. 1	♦ Джек-рассел терьер ♦ Японский хин ♦ Ротвейлер ♦ Йоркширский терьер ♦ Шиб-ину	
<input type="checkbox"/> Эдем	г. Москва, ул. Лебедева, д. 48	♦ Немецкая овчарка ♦ Родезийский риджбек ♦ Шарпей	
<input type="checkbox"/> Королевские лапки	г. Москва, ул. Гагарина, д. 34	♦ Японский хин ♦ Чихуахуа ♦ Померанский шпиц ♦ Йоркширский терьер ♦ Папильон	

рис. 34 Страница «Ринги» для эксперта

#### Листинг 18 – Обработка POST-запроса для сохранения объявления

```
app.post('/expert/new_application', async (req, res) => {
  const user_id = req.query.expert_id;
  const ring_id = req.query.ring_id;

  const parsedUserId = parseInt(user_id, 10);
  const parsedRingId = parseInt(ring_id, 10);

  try {
    const expertResult = await pool.query(`SELECT expert.id FROM "expert" WHERE
user_id = $1`, [parsedUserId]);

    if (expertResult.rows.length === 0) {
      return res.status(404).json({ error: 'Expert not found' });
    }

    const expert_id = expertResult.rows[0].id;
    const existingPairResult = await pool.query(
```

```

    `SELECT * FROM "expert_ring" WHERE expert_id = $1 AND ring_id = $2`,
    [expert_id, parsedRingId]
  );

  if (existingPairResult.rows.length > 0) {
    return res.status(409).json({ error: 'This expert-ring pair already exists'
  });
  }
  const query = {
    text: `INSERT INTO "expert_ring" (expert_id, ring_id) VALUES ($1, $2)`,
    values: [expert_id, parsedRingId]
  };

  await pool.query(query);
  res.status(201).json({ message: 'Application created successfully' });
} catch (error) {
  console.error('Error creating application:', error);
  res.status(500).json({ error: 'Internal server error' });
}
});

```

Обработчик POST запроса для создания новой заявки. Сперва извлекаются **expert\_id** и **ring\_id** из строки запроса (req.query). Далее преобразуются значения параметров из строки в целое число (parseInt).

После происходит проверка существования эксперта. Выполняется запрос к базе данных для получения идентификатора эксперта на основе **user\_id**. Если эксперт не найден (expertResult.rows.length === 0), возвращаем клиенту статус 404 и сообщение об ошибке.

Если эксперт найден, то происходит проверка существования пары эксперт-ринг. Выполняется запрос к базе данных для проверки наличия существующей записи в таблице **expert\_ring** с указанными **expert\_id** и **ring\_id**. Если такая запись существует (existingPairResult.rows.length > 0), возвращаем клиенту статус 409 и сообщение об ошибке.

Если записи еще такой нет, то выполняется вставка новой записи в таблицу **expert\_ring**.

### 3.12 Разработка страницы «Профиль» для администратора

В ходе разработки данной страницы был реализован просмотр данных о пользователе. Каждая строчка с фамилией, именем и отчеством, почтой и

данными о паспорте активны и изменяемы. Также область с картинкой тоже активна для загрузки новой фотографии. После нажатия кнопки «Сохранить данные» новые данные будут обновлены в базе данных.

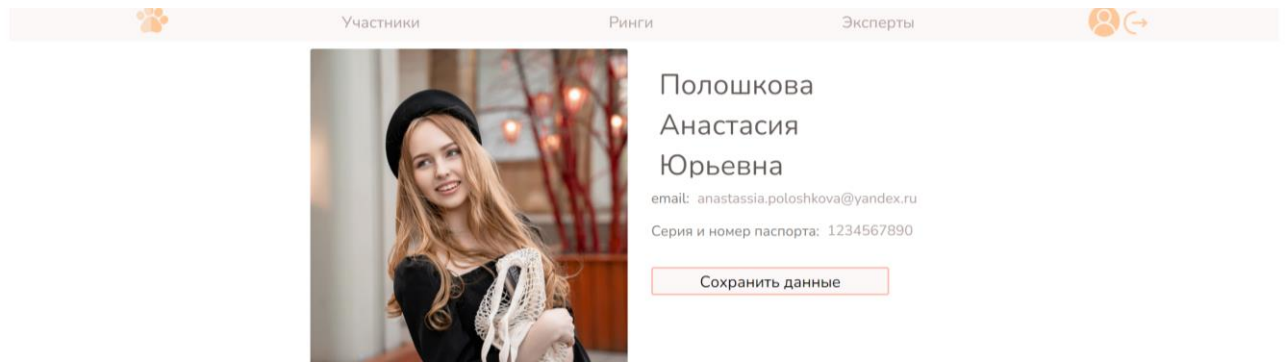


рис. 35 Страница «Профиль» для администратора

Листинг 19 – Форма для загрузки фотографии

```
const handleImageUpload = (event) => {
  const files = Array.from(event.target.files);
  const fileReaders = files.map((file) => {
    return new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onload = () => resolve(reader.result);
      reader.onerror = (error) => reject(error);
      reader.readAsDataURL(file);
    });
  });

  return Promise.all(fileReaders);
};

const handleFileInput = () => {
  fileInputRef.current.click();
};

const handleChangeImage = async (event) => {
  try {
    const convertedImages = await handleImageUpload(event);
    setImage(convertedImages);
  } catch (error) {
    console.log(error);
  }
};
```

```

    }
  };

<div className={style.photo}>
  <input
    type="file"
    accept="image/*"
    ref={fileInputRef}
    style={{ display: 'none' }}
    onChange={handleChangeImage}
  />
  {image === "" || image === null ? (
    <div className={style.image} style={{ backgroundImage:
'url(/photo.svg)' }} onClick={handleFileInput} />
  ) : (
    <div className={style.image} style={{ backgroundImage: `url(${image})` }}
onClick={handleFileInput} />
  )}
</div>

```

Обработка загрузки изображений **handleImageUpload**. В данной функции происходит конвертация файлов в массив: `Array.from(event.target.files)` конвертирует список файлов из `event.target.files` в массив. Для каждого файла создается новый промис, который использует **FileReader** для чтения содержимого файла как URL данных (data URL).

**reader.onload** и **reader.onerror** обрабатывают успешное и ошибочное чтение файла соответственно.

**Promise.all(fileReaders)** возвращает промис, который разрешается, когда все промисы файлов разрешаются.

**handleChangeImage** – это асинхронная функция вызывается при изменении ввода файла. Она использует `handleImageUpload` для обработки загруженных изображений и затем обновляет состояние компонента с помощью `setImage`.

Элемент `input` скрыт (`style={{ display: 'none' }}`) и предназначен для выбора изображения. Он принимает только изображения (`accept="image/*"`). В зависимости от состояния `image`, отображается либо дефолтное изображение (`/photo.svg`), либо загруженное изображение. Когда пользователь кликает на изображение, вызывается `handleFileInput`, который открывает диалоговое окно

для выбора файла.

### 3.13 Разработка страницы «Профиль» для владельца собак

На основе профиля для администратора реализован функционал страницы «Профиль» для владельца собаки. Также на данной странице реализован просмотр всех заявок и их статусов с возможностью отменить (удалить) заявку.

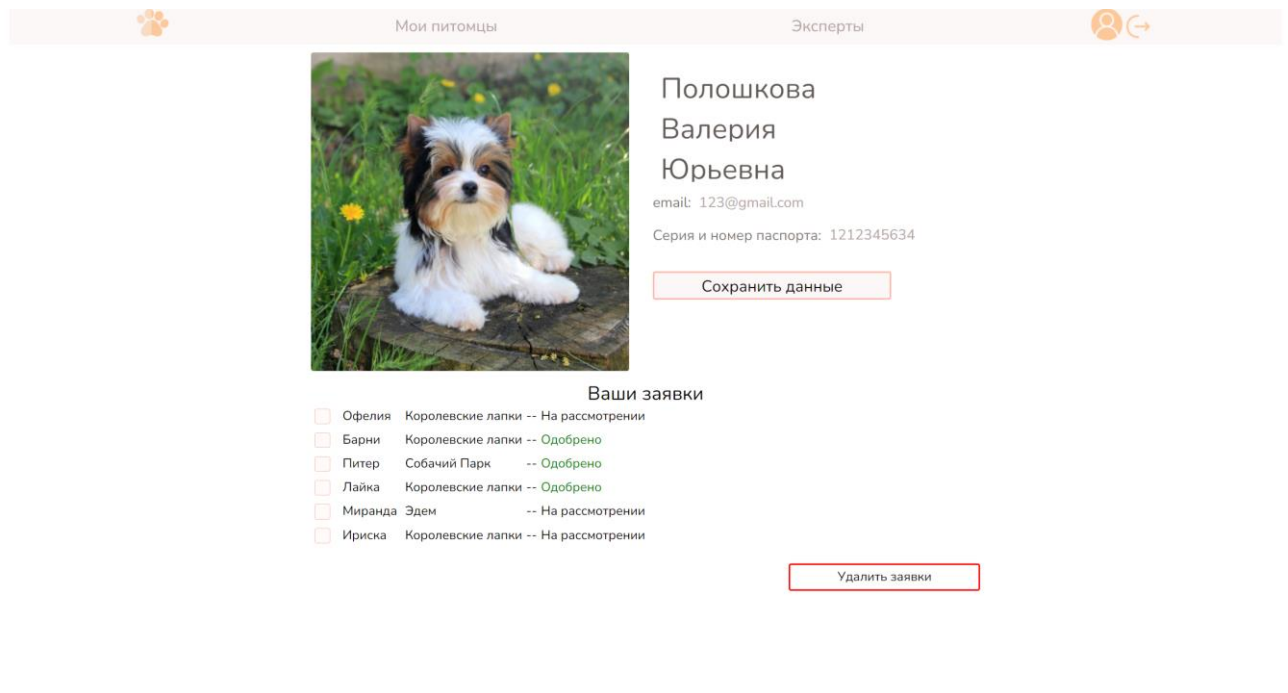


рис. 36 Страница «Профиль» для владельца собак

Листинг 20 – Код для рендеринга заявок пользователя

```
<div className={style.title}>Ваши заявки</div>
  {applications.length !== 0 ? applications.map(el =><tr key={el.id}>
    <td> <Checkbox key={el.id}
      index={el.id}
      onChange={handleCheckBoxChange}/></td>
    <td>{el.nickname}</td>
    <td>{el.ring}</td>
    -- {el.status === 'Одобрено'? <td style={{color:
'green'}}>{el.status}</td> : el.status === 'Отклонено'? <td style={{color:
'red'}}>{el.status}</td> : <td>{el.status}</td>}
  </tr>): <></>
  <div style={{display:'flex', justifyContent:'right'}}>
    <button className={style.delete} onClick={() =>
handleDelete(checkedExceptions)}>Удалить заявки</button>
  </div>
```

</div>

```
const [applications, setApplication] = useState([])
useEffect(()=>{
  const token = localStorage.getItem('token');
  if(token){
    const decodedToken = jwtDecode(token);
    setId(decodedToken.userId)
    setName(decodedToken.name);
    setSurname(decodedToken.surname);
    setPatronymic(decodedToken.patronymic);
    setImage(decodedToken.image);
    setEmail(decodedToken.email);
    setPassport(decodedToken.passport)
  }
  const fetchData = async() =>{
    try{
      const data = await
    axios.get(`http://localhost:8082/user/applications?id=${id}`)
      setApplication(data.data);
    }catch(error){
      console.log(error)
    }
  }
  fetchData()
}, [id])

const [checkedItems, setCheckedItems] = useState([]);

const handleCheckBoxChange = (index) => {
  setCheckedItems(prevState => {
    if (prevState.includes(index)) {
      return prevState.filter(item => item !== index);
    } else {
      return [...prevState, index];
    }
  });
};
```

<div className={style.title}>Ваши заявки</div> - отображает заголовок "Ваши заявки".

Отображение заявок:

Если applications.length !== 0, отображаются заявки. Используется метод map для отображения каждой заявки в строке таблицы (<tr>). Каждая заявка включает чекбокс, никнейм, ринг и статус. Статус заявки отображается цветом:

зеленым для "Одобрено" и красным для "Отклонено".

Кнопка удаления отображается и вызывает функцию **handleDelete** с выбранными элементами (**checkedItems**).

В данном компоненте используется **useState**:

- **applications**: массив заявок, начальное значение — пустой массив.
- **checkedItems**: массив выбранных заявок, начальное значение — пустой массив.
- **id**: идентификатор пользователя, начальное значение — null.

В данном компоненте используется **useEffect** для получения токена из localStorage, декодирования токена с помощью jwtDecode, установки данных пользователя (id, name, surname, patronymic, image, email, passport).

#### *Листинг 21 – Обработка DELETE-запроса для отмены заявки*

```
app.delete('/delete_application', async(req, res)=>{
  const id = req.query.id;
  try{
    await pool.query(`DELETE FROM "application" WHERE application.id = $1`,
[id]);
    res.status(200).json({message: 'Application deleted successfully'})
  }catch(error){
    console.error('Error deleting application:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
})
```

Запрос принимает идентификатор заявки из параметра запроса, выполняет SQL-запрос для удаления заявки из таблицы **application** с указанным идентификатором, отправляет ответ с кодом состояния 200 (Успешно), если удаление прошло успешно, и сообщением об успешном удалении. Если произошла ошибка при выполнении запроса к базе данных, обрабатывает исключение, отправляя ответ с кодом состояния 500 (Внутренняя ошибка сервера) и сообщением об ошибке.

### 3.14 Разработка страницы «Профиль» для эксперта

Данная страница разработана на основе профиля для эксперта. Также добавлен функционал выбора специализацию и изменения его.

Участники Ринги

Осипова  
Миранда

email: dogexpert@yandex.ru  
Серия и номер паспорта: null  
Бивер-Йорк

Сохранить данные

Ваши заявки

☐ Собачий Парк -- Одобрено  
☐ Мир Сладостей -- Одобрено

Удалить заявки

рис. 37 Страница «Профиль» для эксперта

Листинг 22 – Обработка PUT-запроса для эксперта

```
app.put('/expert/update_specialization', async (req, res)=>{
  const id = req.query.id;
  const {specialization} = req.body
  try{
    if (specialization){
      const query = {
        text: `UPDATE "expert" SET breed_id = $1 WHERE user_id = $2`,
        values: [specialization, id]
      };
      await pool.query(query);
      res.status(204).json({ message: 'Specialization updated successfully' });
    } else {
      res.status(400).json({ error: 'breed_id is required' });
    }
  } catch(error){
    console.error('Error:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
})
```

Данный запрос принимает идентификатор эксперта из параметра запроса и новую специализацию из тела запроса. Проверяет, предоставлена ли



специализация в теле запроса. Если нет, возвращает ошибку с кодом состояния 400 (Неверный запрос) и сообщением об отсутствии необходимых данных.

Если специализация предоставлена, выполняет SQL-запрос для обновления записи эксперта в таблице **expert**, устанавливая новое значение поля **breed\_id** для указанного идентификатора пользователя (**user\_id**).

Отправляет ответ с кодом состояния 204 (Успешное обновление), если обновление прошло успешно, и сообщением об успешном обновлении. Если произошла ошибка при выполнении запроса к базе данных, обрабатывает исключение, отправляя ответ с кодом состояния 500 (Внутренняя ошибка сервера) и сообщением об ошибке.

Наиболее подробно с кодом можно ознакомиться в *Приложении 1*.

## Результаты

В ходе курсового проекта разработана информационно-управляющая система для выставки собак. Выполнены следующие задачи:

- проведен анализ предметной области для выявления основных потребностей будущих пользователей информационной системы;
- проанализированы аналоги, выявлены их достоинства и недостатки;
- составлен список основных функциональных и нефункциональных требований;
- проведен сравнительный анализ различных фреймворков и выбран стек технологий для реализации проекта (React.js, Node.js, Express.js, PostgreSQL);
- спроектированы диаграммы (Use Case, диаграмма активности);
- спроектирована база данных (ER-диаграмма);
- разработан пользовательский интерфейс;
- разработана информационно-управляющая система, реализован функционал;

Разработанная информационная система обеспечивает возможность онлайн-регистрации участников выставки собак, включая владельцев собак и судей (экспертов). Предусмотрена проверка и подтверждение заявок на участие. Система позволяет вносить, сохранять и редактировать информацию о каждой собаке. Возможно указать породу, возраст, кличку собаки и данные о ее владельце. Система обеспечивает хранение сведений о собаках. Есть возможность загрузки фотографий собак. Система позволяет выбирать и назначать судей на выставку собак. Система позволяет вводить информацию об оценках участника. Система позволяет управлять доступом к информации и функционалу в зависимости от роли пользователей (администратор, эксперт,

владелец собаки). Система обеспечивает защиту персональных данных участников и другой конфиденциальной информации.

## Список литературы

1. Swiper React Component // URL: <https://swiperjs.com/react> (дата обращения: 10.05.2024).
2. О титулах на выставках собак, оценках, рангах выставок и принятых сокращениях // URL: <https://kormadv.ru/stati/o-titulakh-na-vystavkakh-sobak-otsenkakh-rangakh-vystavok-i-prinyatykh-sokrashcheniyakh> (дата обращения: 29.04.2024).
3. Какие бывают ранги, оценки и титулы на выставках собак // URL: <https://smart-dogs.ru/blog/vystavka-sobak-rangi-otsenki-tituly/> (дата обращения: 29.04.2024).
4. URL: <https://stock.adobe.com/ru/> (дата обращения: 23.04.2024).
5. React Router URL: <https://reactrouter.com/en/main/components/routes> (дата обращения: 07.05.2024).
6. Маршрутизация в большом приложении на React // URL: [https://habr.com/ru/companies/rambler\\_and\\_co/articles/424025/](https://habr.com/ru/companies/rambler_and_co/articles/424025/) (дата обращения: 07.05.2024).
7. Как мы переходили на React-router v6: подводные камни и альтернативы // URL: <https://habr.com/ru/companies/alfa/articles/686954/> (дата обращения: 06.05.2024).
8. React Hook Form URL: <https://react-hook-form.com/> (дата обращения: 03.05.2024).
9. Node JS и React - как создать фулстек приложение. Полное руководство // URL: <https://it-dev-journal.ru/articles/node-js-i-react-kak-napisat-fulstek-prilozhenie-polnoe-rukovodstvo> (дата обращения: 02.05.2024).
10. react-router-dom // URL: <https://www.npmjs.com/package/react-router->

dom (дата обращения: 04.05.2024).

11. InfoDog URL: <https://infodog.com/> (дата обращения: 02.05.2024).
12. Инструкция по работе на форуме ЗооПортала // ZOOПортал.pro URL: [https://zoportal.pro/forum/forum2/topic5511/?PAGEN\\_1=2](https://zoportal.pro/forum/forum2/topic5511/?PAGEN_1=2) (дата обращения: 02.05.2024).
13. URL: <https://bestrussian.dog/en/> (дата обращения: 02.05.2024).
14. URL: [https://www.show-dogs.ru/scripts/list\\_show03.php](https://www.show-dogs.ru/scripts/list_show03.php) (дата обращения: 02.05.2024).
15. Dog Show Club URL: <https://dogshowclub.ru/> (дата обращения: 02.05.2024).
16. JWT Authentication in React // URL: <https://permify.co/post/jwt-authentication-in-react/> (дата обращения: 24.04.2024).
17. React // URL: <https://react.dev/> (дата обращения: 07.05.2024).
18. Express - Node.js web application framework // URL: <https://expressjs.com/> (дата обращения: 07.05.2024).
19. RESTful API на Node.js + MongoDB // URL: <https://habr.com/ru/articles/193458/> (дата обращения: 20.04.2024).
20. node-postgres // URL: <https://node-postgres.com/> (дата обращения: 20.04.2024).