МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Web-приложение для поиска и получения животных из приютов»

Выполнил студент Песецкий Н.А.
(Ф.И.О.)

Руководитель проекта преп.-стажер. Некрасова А.П.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доцент Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стажер. Некрасова А.П.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стажер. Некрасова А.П.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовая работа защищена с оценкой _____

Минск 2024

Содержание

Введение

Веб-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP [1].

Платформа для управления приютами для животных — это программное обеспечение, предназначенное для организации работы приютов в цифровой среде. Она предоставляет пользователям инструменты для учета животных, управления их состоянием, организации процесса распределения животных новым владельца.

Для достижения цели были поставлены следующие задачи:

1. Провести анализ существующих интернет-приютов для животных и определить ключевые требования к разрабатываемому веб-приложению (раздел 1);
2. Разработать архитектуру веб-приложения (раздел 2);
3. Реализовать функционал веб-приложения с учетом поставленных требований (раздел 3);
4. Провести тестирование для выявления и устранения ошибок, а также для проверки соответствия требованиям (раздел 4);
5. Разработать руководство пользователя веб-приложения (раздел 5)

Целевая аудитория веб-приложения — люди, желающие приобрести животное. Помимо этого, целевая аудитория также включает приюты для животных, которые заинтересованы в упрощении процесса распределения животных.

В качестве программной платформы было решено использовать Node.js [2].

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

Веб-приложение «PetFirst» для интернет-приютов должно упростить процесс передачи животных из приютов, включая создание запроса на получение животного и поиск животного в каталоге.

У всех пользователей должна быть возможность просмотреть каталог животных, изучить страницы с их описанием, использовать фильтры и поиск для поиска интересующих животных.

1.2 Обзор аналогичных решений

1.2.1 «Приют Бирюлево»

Платформа «Приют Бирюлево» [3] представляет собой интернет-приют собак и кошек. Главная страница представлена на рисунке 1.1.

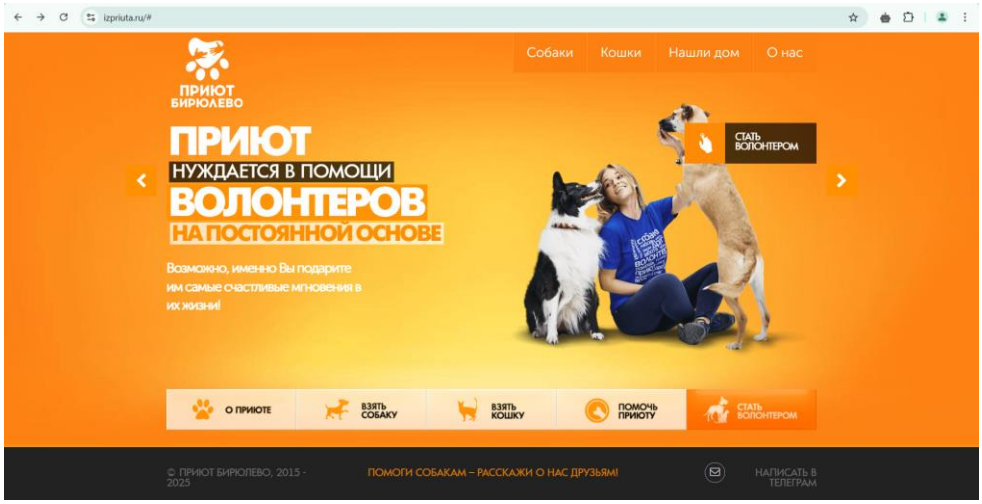

Рисунок 1.1 – Главная страница сайта Приют Бирюлево

Достоинства:
– Простой интерфейс;
– Поиск по критериям;
– Подробная информация о животном.
Недостатки:
– Частично неудобная навигация по сайту;
– Интернет-приют ограничен 1 приютом.

1.2.2 «Преданное сердце»

Сайт интернет-приют котов «Преданное сердце» [4] – это платформа, которая обеспечивает возможность помочь приюту денежными средствами, а также позволяет осуществлять поиск животных. Главная страница представлена на рисунке 1.2.

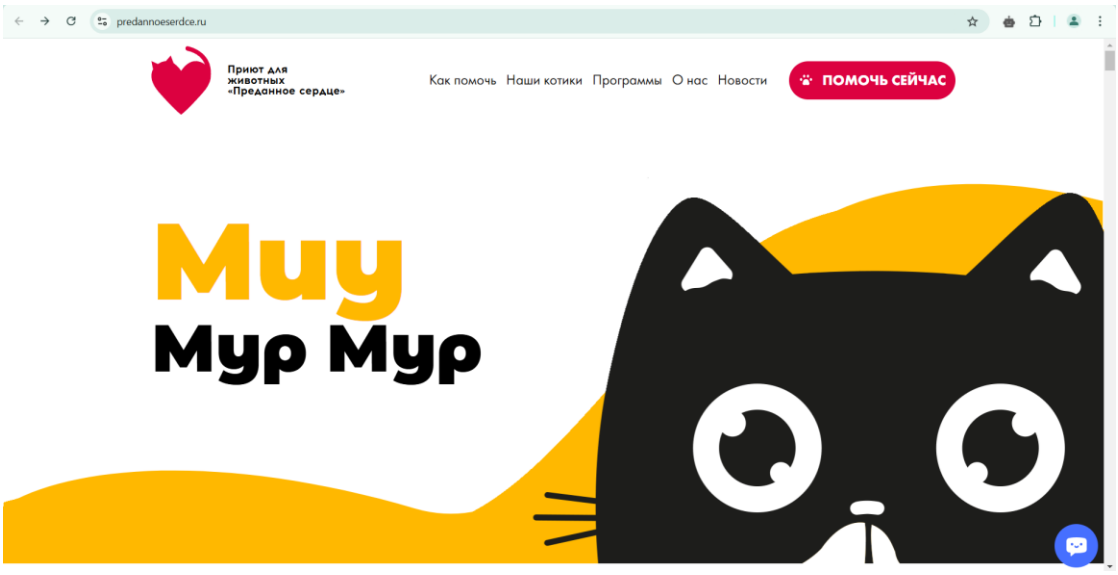

Рисунок 1.2 – Главная страница сайта Преданное сердце

Достоинства:

– Удобная система поиска и фильтров для быстрого нахождения животных с нужными характеристиками;

– Возможность пожертвовать средства приюту;

– Возможность связаться со службой поддержки и получить обратную связь.

Недостатки:

– Однообразие животных в каталоге;

– Отсутствие онлайн коммуникации;

– Отсутствие информации о заболеваниях.

1.2.3 «Искра»

«Искра» [5] – это платформа, предоставляющая получить животное из приюта, а также сдать его в приют. На сайте присутствуют статьи и полезные ссылки для будущих хозяев животных. Главная страница представлена на рисунке 1.3.

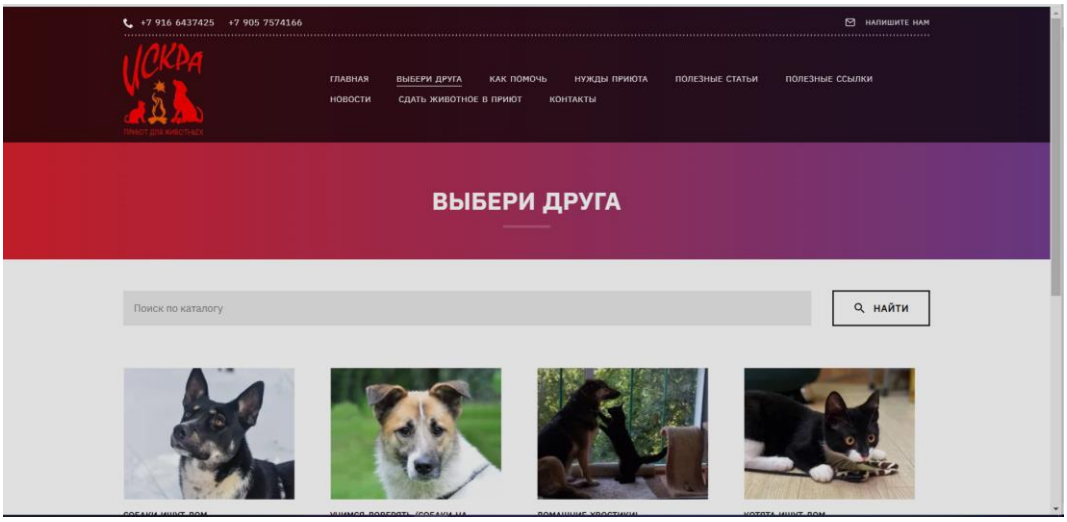

Рисунок 1.3 – Главная страница сайта DNS

Достоинства:
– Адаптивность сайта под планшеты и мобильные устройства;
– Возможность переключения между светлой и темной темой;
– Возможность оставлять отзывы;
– Возможность оплатить заказ прямо на сайте.
Недостатки:
– Однообразие животных.

1.3 Выводы по разделу

1. Анализ аналогичных решений показал, что сайты предлагают базовый функционал, включая поиск животных и просмотр подробной информации о них, но при этом не лишены недостатков, из которых можно отметить малое разнообразие животных, большинство приютов предлагают котов и собак, отсутствие информации о заболеваниях животных.

2. Поставленные задачи требуют разработки веб-приложения с поддержкой четырёх ролей: гостя, клиента, управляющего приютом и администратора, каждая из которых будет обладать своим перечнем функциональных возможностей, рассмотренных в пункте 1.1. В результате разработки веб-приложения «PetFirst» будет создана платформа, которая упрощает процессы интернет-приюта, включая просмотр и управление каталога животных, управление запросами на получения животных, обработку запросов, просмотр и управление списком заболеваний.

2 Проектирование веб-приложения

2.1 Функциональность веб-приложения

Функциональные возможности веб-приложения отображены в диаграмме вариантов использования (рисунок 2.1).

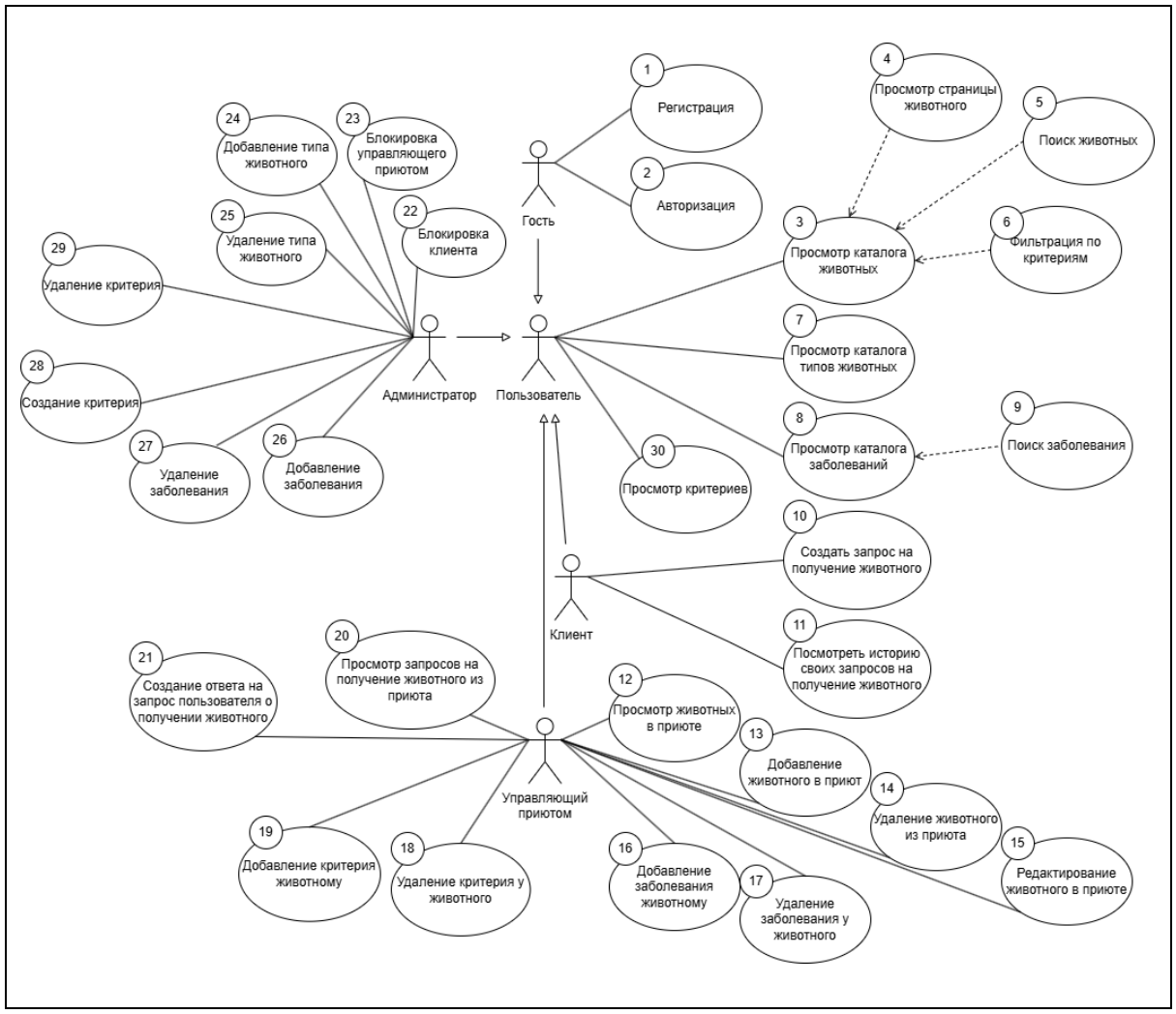

Рисунок 2.1 – Диаграмма вариантов использования

На диаграмме отражены все действующие в приложении роли и функции, которые доступны этим ролям. Описание ролей представлено в таблице 2.1.

Таблица 2.1 – Описание ролей

Роль	Описание
Гость	Может регистрироваться в системе и авторизоваться. Имеет доступ к просмотру каталога животных, заболеваний, типов животных.

Продолжение таблицы 2.1

Клиент	Может создавать запросы на получение животного и просматривать историю запросов.
Управляющий приютом	Может добавлять животных в приют, обновлять о них информацию, добавлять и удалять заболевания у животных в приюте, добавлять и удалять критерии у животных в приюте, обрабатывать запросы клиентов.
Администратор	Обладает правами для администрирования системы: управление типами животных, управление заболеваниями, управление критериями, блокировка пользователей и приютов.

Функциональные возможности пользователя с ролью «Гость» представлены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «Гость»

№	Вариант использования	Пояснение
1	Регистрация	Позволяет создать учетную запись клиента или управляющего приютом.
2	Авторизация	Позволяет войти в систему как клиент или управляющий приютом.
3	Просмотр каталога животных	Обеспечивает возможность просмотра животных информации о животных в приютах.
4	Просмотр каталога типов животных	Обеспечивает возможность читать информационные материалы о типах животных.
5	Просмотр каталога заболеваний	Обеспечивает возможность читать информационные материалы о заболеваниях животных.
6	Просмотр критериев	Обеспечивает возможность просмотра критериев для поиска.

Далее рассмотрим функциональные возможности для пользователя, которые представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя с ролью «Клиент»

№	Вариант использования	Пояснение
1	Просмотр каталога животных	Обеспечивает возможность просмотра животных информации о животных в приютах.
2	Просмотр каталога типов животных	Обеспечивает возможность читать информационные материалы о типах животных.
3	Просмотр каталога заболеваний	Обеспечивает возможность читать информационные материалы о заболеваниях животных.
4	Создание запроса на получение животного	Обеспечивает возможность создавать запрос на получение животного.
5	Просмотр истории запросов на получение животных	Обеспечивает возможность просматривать созданные запросы на получение животного.
6	Просмотр критериев	Обеспечивает возможность просмотра критериев для поиска.

Функциональные возможности пользователя с ролью «Управляющий приютом» представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности пользователя с ролью «Управляющий приютом»

№	Вариант использования	Пояснение
1	Просмотр каталога животных	Обеспечивает возможность просмотра животных информации о животных в приютах.
2	Просмотр каталога типов животных	Обеспечивает возможность читать информационные материалы о типах животных.
3	Просмотр каталога заболеваний	Обеспечивает возможность читать информационные материалы о заболеваниях животных.

Продолжение таблицы 2.4

4	Просмотр животных в приюте	Просмотр списка животных, принадлежащих приюту.
5	Добавление животного в приют	Добавление животного в список животных приюта.
6	Удаление животного из приюта	Удаление животного из списка животных приюта.
7	Редактирование животного в приюте	Редактирование животного в списке животных приюта.
8	Добавление заболевания животному	Добавление заболевания из списка заболеваний животному в приюте.
9	Удаление заболевания у животного	Удаление заболевания у животного в приюте.
10	Добавление критерия животному	Добавление критерия из списка критериев животному в приюте.
11	Удаление критерия у животного	Удаление критерия у животного в приюте.
12	Просмотр запросов на получение животного из приюта	Просмотр запросов на получение животного из приюта.
13	Создание ответа на запрос пользователя о получении животного	Создание ответа на запрос пользователя.
14	Просмотр критериев	Обеспечивает возможность просмотра критериев для поиска.

Функциональные возможности пользователя с ролью «Администратор» представлены в таблице 2.5.

Таблица 2.5 – Функциональные возможности пользователя с ролью «Администратор»

№	Вариант использования	Пояснение
1	Просмотр каталога животных	Обеспечивает возможность просмотра животных информации о животных в приютах.
2	Просмотр каталога типов животных	Обеспечивает возможность читать информационные материалы о типах животных.
3	Просмотр каталога заболеваний	Обеспечивает возможность читать информационные материалы о заболеваниях животных.
4	Блокировка клиентов	Блокировка аккаунта клиента.
5	Блокировка управляющего приютом	Блокировка аккаунта управляющего приютом.
6	Добавление типов животного	Добавление типа животного в список типов животных.
7	Удаление типов животного	Удаление типа животного из списка типов животных.
8	Создание критериев	Добавление критерия в список критериев животных.
9	Удаление критериев	Удаление критерия из списка критериев животных.
10	Добавление заболеваний	Добавление заболеваний в список существующих заболеваний.

Продолжение таблицы 2.5

11	Удаление заболеваний	Удаление заболеваний из списка возможных заболеваний животного.
12	Просмотр критериев	Обеспечивает возможность просмотра критериев для поиска.

Такое распределение функций позволяет обеспечить четкое разграничение прав доступа, что способствует удобству использования и безопасности системы.

2.2 Логическая схема базы данных

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных.

В проекте была выбрана система управления базами данных MSSQL [6] в качестве основной базы данных. Для реализации функциональности приложения была разработана структура базы данных, которая состоит из 9 таблиц. Диаграмма логической схемы базы данных представлена на рисунке 2.2.

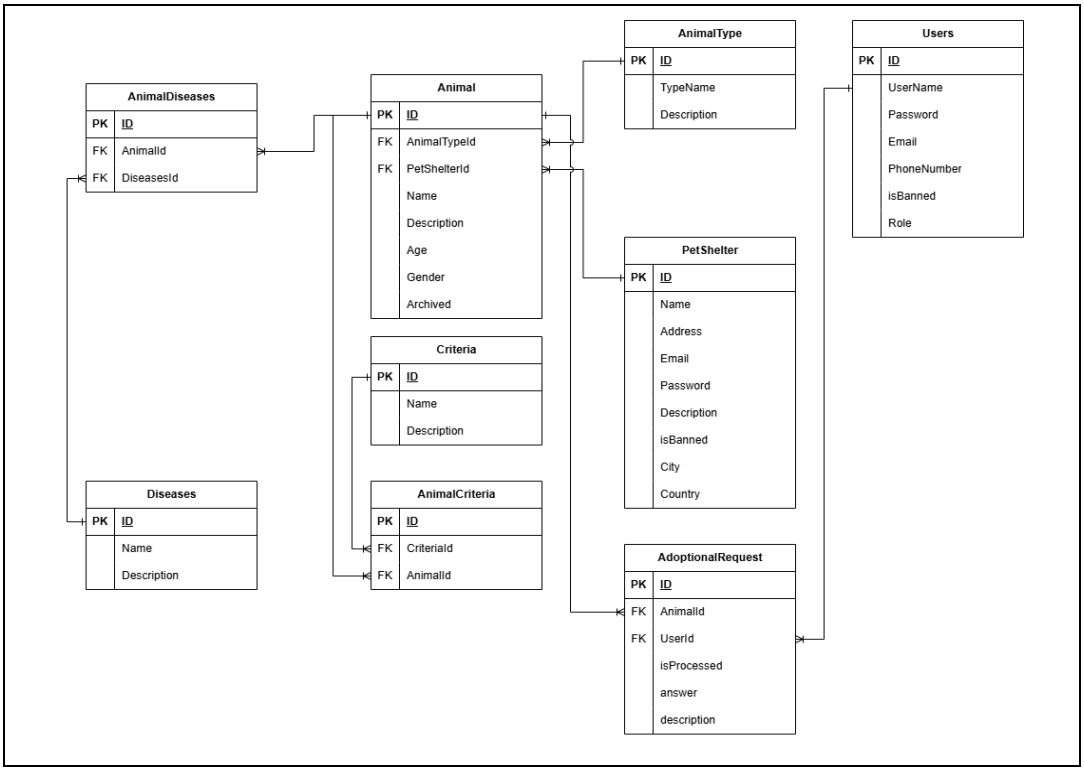

Рисунок 2.2 – Логическая схема базы данных

Для удобного взаимодействия с базой данных был выбран Prisma – ORM (Object-Relational Mapping) [7] для Node.js, который предоставляет удобный интерфейс для работы с базой данных через объекты и модели.

Таблица 2.6 – Описание таблиц базы данных

Название таблицы	Описание таблицы
Users	Содержит информацию о клиентах приложения.
PetShelters	Содержит информацию о управляющих приютами приложения.
AnimalType	Содержит информацию о типах животных.
Animal	Содержит информацию о животных.
Diseases	Хранит данные о заболеваниях.
AnimalDiseases	Реализует связь «многие-ко-многим» между таблицами Animal и Diseases, указывая, какие заболевания имеются у конкретных животных.
Criteria	Содержит критерии.
AnimalCriteria	Реализует связь «многие-ко-многим» между таблицами Animal и Criteria, указывая, какие критерии применяются к конкретным животным.
AdoptionalRequest	Хранит запросы на получения животных от пользователей, включая идентификаторы животного и пользователя, статус обработки, описание и ответ.

В таблице 2.7 показано описание полей таблицы «users».

Таблица 2.7 – Описание структуры таблицы «users»

Название	Тип данных	Описание
ID	Int	Идентификатор пользователя (первичный ключ)
UserName	nvarchar(1000)	Имя пользователя
Password	nvarchar(1000)	Захешированный пароль
Email	nvarchar(1000)	Почта
PhoneNumber	nvarchar(1000)	Номер телефона
Role	nvarchar(1000)	Роль пользователя
isBanned	Int	Заблокирован или нет

В таблице 2.8 показано описание полей таблицы «PetShelters».

Таблица 2.8 – Описание структуры таблицы «PetShelters»

Название	Тип данных	Описание
ID	Int	Идентификатор управляющего приютом (первичный ключ)
Name	nvarchar(1000)	Название приюта
Email	nvarchar(1000)	Почта
Password	nvarchar(1000)	Захешированный пароль
Description	nvarchar(1000)	Описание приюта
isBanned	Int	Заблокирован или нет
Address	nvarchar(1000)	Адрес приюта
Country	nvarchar(1000)	Страна в которой расположен приют
City	nvarchar(1000)	Город в котором расположен приют

В таблице 2.9 показано описание полей таблицы «AnimalType».

Таблица 2.9 – Описание структуры таблицы «AnimalType»

Название	Тип данных	Описание
ID	Int	Идентификатор типа животного (первичный ключ)
TypeName	nvarchar(1000)	Название типа животного
Description	nvarchar(1000)	Описание типа животного

В таблице 2.10 показано описание полей таблицы «Animal».

Таблица 2.10 – Описание структуры таблицы «Animal»

Название	Тип данных	Описание
ID	Int	Идентификатор животного (первичный ключ)
PetShelterId	Int	Идентификатор управляющего приютом (внешний ключ)
AnimalTypeId	Int	Идентификатор типа животного (внешний ключ)
Name	nvarchar(1000)	Кличка животного
Description	nvarchar(1000)	Описание животного
Age	Int	Возраст животного
Gender	nvarchar(1000)	Пол животного
Archived	Int	Архивирован или нет

В таблице 2.11 показано описание полей таблицы «Diseases».

Таблица 2.11 – Описание структуры таблицы «Diseases»

Название	Тип данных	Описание
ID	Int	Идентификатор заболевания (первичный ключ)
Name	nvarchar(1000)	Название заболевания
Description	nvarchar(1000)	Описание заболевания

В таблице 2.12 показано описание полей таблицы «AnimalDiseases».

Таблица 2.12 – Описание структуры таблицы «AnimalDiseases»

Название	Тип данных	Описание
ID	Int	Идентификатор записи (первичный ключ)
AnimalId	Int	Идентификатор животного (внешний ключ)
DiseasesId	Int	Идентификатор заболевания (внешний ключ)

В таблице 2.13 показано описание полей таблицы «Criteria».

Таблица 2.13 – Описание структуры таблицы «Criteria»

Название	Тип данных	Описание
ID	Int	Идентификатор критерия (первичный ключ)
Name	nvarchar(1000)	Название критерия
Description	nvarchar(1000)	Описание критерия

В таблице 2.14 показано описание полей таблицы «AnimalCriteria».

Таблица 2.14 – Описание структуры таблицы «AnimalCriteria»

Название	Тип данных	Описание
ID	Int	Идентификатор записи (первичный ключ)
AnimalId	Int	Идентификатор животного (внешний ключ)
CriteriaId	Int	Идентификатор критерия (внешний ключ)

В таблице 2.15 показано описание полей таблицы «AdoptionalRequest».

Таблица 2.15– Описание структуры таблицы «AdoptionalRequest»

Название	Тип данных	Описание
ID	Int	Идентификатор записи (первичный ключ)
AnimalId	Int	Идентификатор животного (внешний ключ)
UserId	Int	Идентификатор клиента (внешний ключ)
isProcessed	Int	Обработан или нет
description	nvarchar(1000)	Текст запроса
answer	nvarchar(1000)	Ответ на запрос

В таблице 2.16 описаны связи таблиц друг с другом.

Таблица 2.16 – Описание связей сущностей

Таблица источник	Связанная таблица	Тип связи	Пояснение
Users	AdoptionalRequest	Один ко многим	Один пользователь может создавать много запросов
Diseases	AnimalDiseases	Один ко многим	Одна болезнь может быть у многих животных
Animal	AnimalDiseases	Один ко многим	Одно животное может иметь множество заболеваний
Criteria	AnimalCriteria	Один ко многим	Один критерий может быть у множества животных
Animal	AnimalCriteria	Один ко многим	Одно животное может иметь множество критериев
PetShelter	Animal	Один ко многим	Один приют может иметь множество животных
AnimalType	Animal	Один ко многим	Один тип может быть у множества животных
Animal	AdoptionalRequest	Один ко многим	Одно животное может быть у множества запросов.

Скрипт создания баз данных представлен в приложении А.

2.3 Архитектура веб-приложения

Архитектура приложения представляет собой распределённую систему, развернутую с использованием современных технологий, обеспечивающих производительность, масштабируемость и удобство управления. Архитектура веб-приложения представлена на рисунке 2.3.

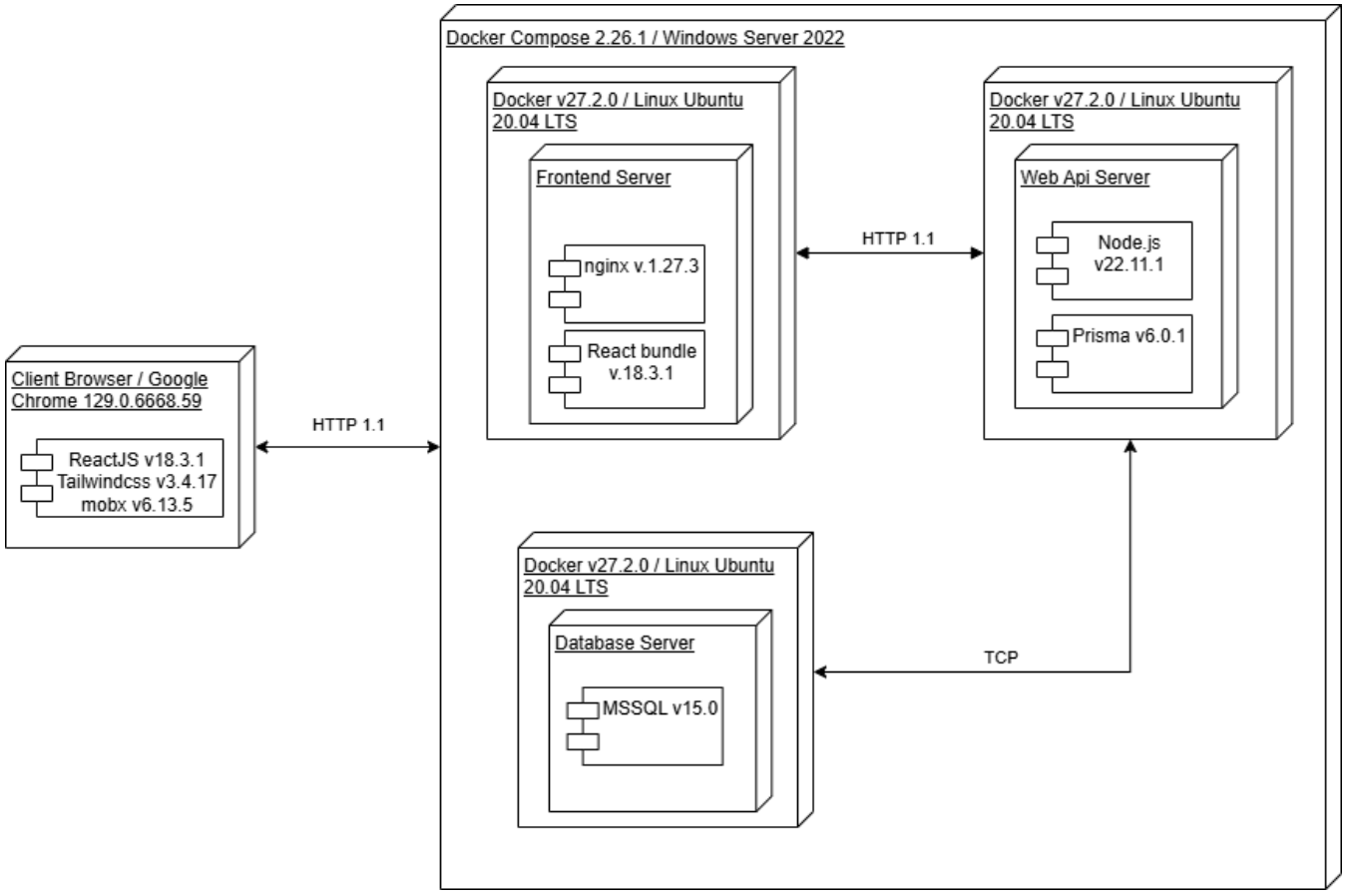

Рисунок 2.3 – Диаграмма развертывания

Пояснение назначения каждого элемента веб-приложения представлено в таблице 2.17.

Таблица 2.17 – Назначение элементов архитектурной схемы веб-приложения

Элемент	Назначение
Frontend Server (nginx [8])	Предоставляет доступ к статическим ресурсам фронтенд -части веб-приложения
Database Server (MSSQL)	Используется для хранения и предоставления доступа к данным, которые необходимы для работы веб-приложения.
Backend Server (Node.js, Prisma)	Обрабатывает запросы пользователя, запрашивает данные из базы данных с помощью ORM Prisma.
Client Browser	Отображает фронтенд-часть веб-приложения, отправляет запросы пользователя, отображает ответы сервера.

Описание протоколов, используемых при работе веб-приложений, представлено в таблице 2.18.

Таблица 2.18 – Описание используемых протоколов

Протокол	Назначение
HTTP	Обмен данными между Client Browser и Frontend Server, Frontend Server и Backend Server.
TCP [9]	Обмен данными между Database Server и Веб API Server.

2.4 Выводы по разделу

В процессе проектирования веб-приложения «PetFirst» определены основные роли пользователей, функции, архитектура, а также схемы базы данных и связи между сущностями. Приложение разработано с учётом современных технологий и стандартов, что обеспечивает его функциональность, масштабируемость и удобство использования.

Основные выводы представлены в следующем списке:

1. Поддержка четырёх ролей с четко разграниченными правами доступа и функциональными возможностями: гость, клиент, управляющий приютом, администратор. Гость может зарегистрироваться и авторизоваться, Пользователь имеет доступ к основным функциям, Клиент имеет возможность создавать запросы на получение животных, Управляющий приютом имеет возможность управлять животными, а Администратор управляет системой.
2. Количество функций: 30. Функционал приложения охватывает регистрацию и авторизацию, просмотр каталогов, управление запросами и животными.
3. Количество таблиц в базе данных: 9. Таблицы охватывают основные сущности приложения, такие как пользователи, управляющие приютом, животные, болезни, типы животных, запросы на получения животных.
4. Веб-приложение имеет монолитную архитектуру с применением Nginx в качестве веб-сервера, MSSQL для хранения данных, Node.js для серверной части и Docker Compose [10] для запуска многоконтейнерных Docker-приложений.

Таким образом, веб-приложение «PetFirst» спроектировано с учётом всех ключевых аспектов, включая современные технологии и практики проектирования, что делает его эффективным инструментом для решения задач пользователей.

3 Реализация веб-приложения

3.1 Программная платформа Node.js

Для серверной части проекта была выбрана платформа Node.js, которая характеризуется событийно-ориентированной архитектурой и моделью однопоточного выполнения. В рамках проекта Node.js используется совместно с фреймворком Express.js [11], который упрощает настройку серверной части и разработку API.

3.2 Система управления базами данных MSSQL

Для хранения и управления данными в проекте была выбрана система управления базами данных MSSQL. MSSQL является одной из самых популярных реляционных СУБД, которая обеспечивает высокую производительность, надежность и масштабируемость.

3.3 Object-Relational Mapping Prisma

Для взаимодействия с базой данных MSSQL в проекте используется ORM Prisma. Prisma — это мощная и гибкая библиотека для Node.js, которая предоставляет удобный интерфейс для работы с реляционными базами данных. Она поддерживает MSSQL, PostgreSQL, SQLite и MySQL.

Сопоставление моделей, используемых в Prisma, с их реальными структурами представлено в таблице 3.1.

Таблица 3.1 – Сопоставление моделей с их реальной структурой

Название модели	Название таблицы
Users	Users
PetShelter	PetShelter
AnimalType	AnimalType
Diseases	Diseases
Animal	Animal
AnimalDiseases	AnimalDiseases
AdoptionRequest	AdoptionRequest
Criteria	Criteria
AnimalCriteria	AnimalCriteria

Код, описывающий модель Users, приведён в листинге 3.1.

```
model Users {
        ID          Int      @id @default(autoincrement())
```

```
        UserName    String  @unique
        Password    String
        Email       String  @unique
        PhoneNumber String? @unique
        Role        String
        isBanned    Int

        AdoptionRequests AdoptionRequest[]
}
```

Листинг 3.1 – Модель «Users»

Модель «Users» представляет собой описание таблицы пользователей в базе данных. Она включает в себя поля для уникального идентификатора пользователя (ID), имени пользователя (UserName), электронной почты (Email), пароля (Password), номер телефона (PhoneNumber), роли пользователя (Role), блокировка аккаунта (isBanned). Поле ID автоматически инкрементируется и является первичным ключом. Поля UserName и Email должны быть уникальными. Роль пользователя может быть либо 'user', либо 'admin'.

Код, описывающий модель PetShelter, приведён в листинге 3.2.

```
model PetShelter {
        ID          Int     @id @default(autoincrement())
        Name        String  @unique
        Address     String  @unique
        Email       String  @unique
        Password    String
        Description String
        isBanned    Int
        Country     String
        City        String
        Animals     Animal[]
}
```

Листинг 3.2 – Модель «PetShelter »

Модель «PetShelter» представляет собой описание таблицы приютов для животных в базе данных. Она включает в себя поля для уникального идентификатора приюта (ID), названия приюта (Name), адреса (Address), электронной почты (Email), пароля (Password), описания (Description), статуса блокировки (isBanned), страны (Country) и города (City). Поле ID автоматически инкрементируется и является первичным ключом. Поля Name, Address и Email

должны быть уникальными. Модель также содержит связь с таблицей животных (Animals), указывая на животных, находящихся в данном приюте.

Код, описывающий модель AnimalType, приведён в листинге 3.3.

```
model AnimalType {
        ID          Int      @id @default(autoincrement())
        TypeName    String @unique
        Description String
        Diseases    AnimalTypeDiseases[]
        Animals     Animal[]
}
```

Листинг 3.3 – Модель «AnimalType»

Модель «AnimalType» представляет собой описание таблицы типов животных в базе данных. Она включает в себя поля для уникального идентификатора типа животного (ID), названия типа (TypeName) и описания типа (Description). Поле ID автоматически инкрементируется и является первичным ключом. Поле TypeName должно быть уникальным.

Код, описывающий модель Diseases, приведён в листинге 3.4.

```
model Diseases {
        ID          Int      @id @default(autoincrement())
        Name        String   @unique
        Description String
        AnimalTypes AnimalTypeDiseases[]
        Animals     AnimalDiseases[]
}
```

Листинг 3.4 – Модель «Diseases»

Модель «Diseases» представляет собой описание таблицы заболеваний в базе данных. Она включает в себя поля для уникального идентификатора заболевания (ID), названия заболевания (Name) и описания заболевания (Description). Поле ID автоматически инкрементируется и является первичным ключом. Поле Name должно быть уникальным.

Код, описывающий модель AnimalDiseases, приведён в листинге 3.5.

```
model Diseases {
        ID          Int      @id @default(autoincrement())
        AnimalId    Int
        DiseasesId Int



        Animal      Animal   @relation(fields: [AnimalId], references: [ID])
```

```
        Diseases   Diseases @relation(fields: [DiseasesId], references: [ID])

        @@unique([AnimalId, DiseasesId])
}
```

Листинг 3.5 – Модель «AnimalDiseases»

Модель «AnimalDiseases» представляет собой описание таблицы-связки, реализующей отношение «многие-ко-многим» между таблицами Animal и Diseases. Она включает в себя поля для уникального идентификатора записи (ID), идентификатора животного (AnimalId) и идентификатора заболевания (DiseasesId). Поле ID автоматически инкрементируется и является первичным ключом. Поля AnimalId и DiseasesId вместе образуют уникальный индекс, обеспечивая уникальность каждой комбинации животного и заболевания.

Код, описывающий модель Animal, приведён в листинге 3.6.

```
model Animal {
        ID          Int       @id @default(autoincrement())
        PetShelterId Int?
        AnimalTypeId Int
        Name        String
        Description  String
        Age         Int
        Gender       String
        Archived     Int

        PetShelter   PetShelter? @relation(fields: [PetShelterId], references: [ID])
        AnimalType    AnimalType  @relation(fields: [AnimalTypeId], references:
[ID])
        Diseases     AnimalDiseases[]
        Criteria     AnimalCriteria[]
        AdoptionRequests AdoptionRequest[]
}
```

Листинг 3.6 – Модель «Animal»

Модель «Animal» представляет собой описание таблицы животных в базе данных. Она включает в себя поля для уникального идентификатора животного (ID), идентификатора приюта (PetShelterId), идентификатора типа животного (AnimalTypeId), имени животного (Name), описания (Description), возраста (Age), пола (Gender) и статуса архивирования (Archived). Поле ID автоматически инкрементируется и является первичным ключом. Поле PetShelterId является опциональным и может быть NULL, указывая на отсутствие приюта. Модель

содержит связи с таблицей приютов (PetShelter), таблицей типов животных (AnimalType), таблицей заболеваний (AnimalDiseases), таблицей критериев (AnimalCriteria) и таблицей запросов на усыновление (AdoptionRequest), обеспечивая комплексную информацию о каждом животном.

Код, описывающий модель AdoptionRequest, приведён в листинге 3.7.

```
model AdoptionRequest {
        ID          Int    @id @default(autoincrement())
        AnimalId    Int
        UserId      Int
        isProcessed Int
        description String
        answer      String?

        Animal      Animal @relation(fields: [AnimalId], references: [ID])
        User        Users  @relation(fields: [UserId], references: [ID])
}
```

Листинг 3.7 – Модель «AdoptionRequest»

Модель «AdoptionRequest» представляет собой описание таблицы запросов на усыновление животных в базе данных. Она включает в себя поля для уникального идентификатора запроса (ID), идентификатора животного (AnimalId), идентификатора пользователя (UserId), статуса обработки запроса (isProcessed), описания запроса (description) и ответа на запрос (answer). Поле ID автоматически инкрементируется и является первичным ключом. Поля AnimalId и UserId являются внешними ключами, связывающими запрос с конкретным животным и пользователем соответственно. Поле answer является опциональным и может быть NULL. Модель содержит связи с моделями Animal и Users, указывая на соответствующие записи в этих таблицах.

Код, описывающий модель Criteria, приведён в листинге 3.8.

```
model Criteria {
        ID          Int      @id @default(autoincrement())
        Name        String   @unique
        Description String
        Animals     AnimalCriteria[]
}
```

Листинг 3.8 – Модель «Criteria»

Модель «Criteria» представляет собой описание таблицы критериев оценки животных в базе данных. Она включает в себя поля для уникального идентификатора критерия (ID), названия критерия (Name) и описания критерия

(Description). Поле ID автоматически инкрементируется и является первичным ключом. Поле Name должно быть уникальным. Модель содержит связь с таблицей AnimalCriteria, указывая на критерии, применяемые к конкретным животным.

Код, описывающий модель AnimalCriteria, приведён в листинге 3.9.

```
model AnimalCriteria {
        ID        Int      @id @default(autoincrement())
        AnimalId  Int
        CriteriaId Int

        Animal    Animal   @relation(fields: [AnimalId], references: [ID])
        Criteria  Criteria @relation(fields: [CriteriaId], references: [ID])

        @@unique([AnimalId, CriteriaId])
}
```

Листинг 3.9 – Модель «AnimalCriteria»

Модель «AnimalCriteria» представляет собой описание таблицы-связки, реализующей отношение «многие-ко-многим» между таблицами Animal и Criteria. Она включает в себя поля для уникального идентификатора записи (ID), идентификатора животного (AnimalId) и идентификатора критерия (CriteriaId). Поле ID автоматически инкрементируется и является первичным ключом. Поля AnimalId и CriteriaId вместе образуют уникальный индекс, обеспечивая уникальность каждой комбинации животного и критерия. Модель содержит связи с моделями Animal и Criteria, указывая на соответствующие записи в этих таблицах.

3.4 Программные библиотеки

В процессе разработки серверной части веб-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

Библиотека	Версия	Назначение
bcrypt [14]	5.1.1	Библиотека для хеширования паролей.
cookie-parser [15]	1.4.7	Библиотека для работы с cookie.
jsonwebtoken [16]	9.0.2	Библиотека для работы с JSON Веб Tokens (JWT).

Продолжение таблицы 3.2

@prisma/client [17]	6.0.1	ORM Для работы с базой данных.
cors [18]	2.8.5	Библиотека для настройки CORS.
nodemon [19]	3.1.9	Библиотека, следящая за обновлениями файлов и перезапускающая сервер.
swagger-jsdoc [20]	6.2.8	Библиотека для генерации документации API на основе JSDoc комментариев.
swagger-ui-express [21]	5.0.1	Middleware для интеграции Swagger UI с Express.

В процессе разработки клиентской части веб-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

Библиотека	Версия	Назначение
Axios[22]	1.7.9	Библиотека для выполнения HTTP-запросов.
Mobx[23]	6.13.5	Библиотека для управления состоянием приложения.
react-toastify[24]	11.0.2	Библиотека для отображения уведомлений в React.
React[25]	18.3.1	Библиотека для создания пользовательских интерфейсов.
react-router-dom	6.27.0	Библиотека для маршрутизации в приложениях React.
react-dom	18.3.1	Библиотека для взаимодействия React с DOM, обеспечивающая рендеринг компонентов в браузере.
mobx-react-lite	4.1.0	Библиотека для интеграции MobX с React, оптимизированная для функциональных компонентов.
Tailwindcss[26]	3.3.1	Фреймворк для быстрого создания пользовательских интерфейсов.

Программные библиотеки позволяют упростить реализацию веб-приложения.

3.5 Структура серверной части

Основные компоненты структуры серверной части включают в себя несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Роутеры — управляют маршрутами и направляют запросы к соответствующим контроллерам.
2. Контроллеры — обрабатывают запросы от клиента, выполняют бизнес-логику через сервисы и возвращают ответы.
3. Middleware — промежуточные обработчики, используемые для валидации данных и обеспечения безопасности.
4. Сервисы — промежуточные обработчики, используемые для обработки данных пред работай с базой данных.
5. Модели — промежуточные обработчики, используемые для работы с базой данных.

В таблице 3.4 приведён список директорий серверной части проекта.

Таблица 3.4 – Директории серверной части проекта

Директория	Назначение
adoptionalRequest	Содержит контроллер и сервис для взаимодействия с моделью запросов.
animalDiseases	Содержит контроллер и сервис для взаимодействия с моделью болезней.
animals	Содержит контроллер и сервис для взаимодействия с моделью животных.
animalsType	Содержит контроллер и сервис для взаимодействия с моделью типами животных.
authorization	Содержит контроллер и сервис для взаимодействия с jwt токенами и моделью пользователей.
exceptions	Содержит модели ошибок.
database	Содержит объект базы данных и функции DTO (Data transfer object).
diseases	Содержит контроллер и сервис для взаимодействия с моделью болезней.
middlewares	Содержит промежуточные обработчики (middleware) для обработки запросов, аутентификации, логирования и обработки ошибок.
models	Содержит классы для взаимодействия с базой таблицой в базе данных

Продолжение таблицы 2.11

petShelters	Содержит контроллер и сервис для взаимодействия с моделью управляющих приютом.
router	Содержит маршруты (роутеры), которые определяют URL-пути и соответствующие контроллеры для обработки запросов.
prisma	Содержит схему базы данных.

Таблица соответствия маршрутов контроллерам в исходном коде представлена в таблице 3.5.

Таблица 3.5 – Контроллеры и функции маршрутов

URL	Метод	Контроллер	Метод контроллера	№ функции	Описание
/user/register	POST	UserController	registration	1	Регистрация пользователя
/user/login	POST	UserController	login	2	Авторизация пользователя
/petshelter /register	POST	PetShelterController	create	1	Регистрация управляющего приютом
/petshelter/login	POST	PetShelterController	registration	2	Авторизация управляющего приютом
/animals	GET	AnimalsController	search	3	Просмотр каталога животных
/animals/:id	GET	AnimalsController	getById	4	Просмотр страницы животного
/ animals?page=${ page }	GET	AnimalsController	search	5	Поиск животных
/ animals?page=${ page }&type={ type }		AnimalsController	search	6	Фильтрация по критериям

Продолжение таблицы 3.5

/animal-type	GET	AnimalTypeController	getAll	7	Просмотр каталога типов животных
/diseases	GET	DiseaseController	search	8	Просмотр каталога заболеваний
/diseases?page=${page}&search=${search}	GET	DiseaseController	search	9	Поиск заболевания
/adoption-requests	POST	AdoptionRequestController	create	10	Создать запрос на получение животного
/adoption-requests	GET	AdoptionRequestController	getAll	11	Посмотреть историю своих запросов на получение животного
/animals?page=${page }&PetShelter=${PetShelter }	GET	AnimalsController	search	12	Просмотр животных в приюте
/animals	POST	AnimalsController	create	13	Добавление животного в приют
/animals	DELETE	AnimalsController	delete	14	Удаление животного из приюта
/animals/:id	PUT	AnimalsController	update	15	Редактирование животного в приюте
/animal-diseases	POST	AnimalDiseasesController	create	16	Добавление заболевания животному
/animal-diseases/:id	DELETE	AnimalDiseasesController	delete	17	Удаление заболевания у животного

Продолжение таблицы 3.5

/criteria/:id	DELETE	CriteriaController	delete	18	Удаление критерия у животного
/criteria	POST	CriteriaController	create	19	Добавление критерия животному
/adoption-requests	GET	AdoptionRequestController	getAll	20	Просмотр запросов на получение животного из приюта
/adoption-requests/:id	PUT	AdoptionRequestController	update	21	Создание ответа на запрос пользователя о получении животного
/user/ban	PUT	UserController	update	22	Блокировка клиента
/petshelters/ban	PUT	PetShelterController	update	23	Блокировка управляющего приютом
/animal-type	POST	AnimalTypeController	create	24	Добавление типа животного
/animal-type/:id	DELETE	AnimalTypeController	delete	25	Удаление типа животного
/diseases	POST	DiseaseController	create	26	Добавление заболевания
/diseases/:id	DELETE	DiseaseController	delete	27	Удаление заболевания
/criteria	POST	CriteriaController	create	28	Создание критерия
/criteria/:id	DELETE	CriteriaController	delete	29	Удаление критерия
/criteria	GET	CriteriaController	getAll	30	Просмотр критериев

При передаче данных между клиентом и сервером используется формат JSON (JavaScript Object Notation).

3.6 Реализация функций для гостя

3.6.1 Реализация функции регистрации клиента

Функция registration в UserController предназначена для регистрации новых пользователей в системе. Это асинхронный метод, который обрабатывает HTTP POST-запросы. Его задача — принять данные, отправленные клиентом, обработать их, создать нового клиент в базе данных и отправить ответ клиенту. Функция представлена в листинге 3.13.

```
static async registration(req, res, next) {
            try {
              const { username, email, password, phonenumber } = req.body;
              const userData = await UserService.registration(
                username,
                email,
                password,
                phonenumber
              );
              res.cookie("refreshToken", userData.refreshToken, {
                maxAge: 30 * 24 * 60 * 60 * 1000,
                httpOnly: true,
              });
              return res.json(userData);
            } catch (e) {
              console.log(e);
              next(e);
            }
  }
```

Листинг 3.13 – Функция регистрации Клиента

Пароль шифруется с использованием библиотеки bcrypt. Затем выполняется проверка уникальности имени и email пользователя в базе данных. Если пользователь с таким именем или email уже существует, возвращается сообщение об ошибке. Если проверка проходит успешно, создаётся новая запись пользователя с ролью клиент. Клиенту возвращаются данные об аккаунте и jwt токен.

3.6.2 Реализация функции авторизации клиента

Функция login в UserController отвечает за процесс авторизации пользователя в системе под ролью клиента. Она обрабатывает HTTP POST-запросы, принимая email и пароль пользователя. В первую очередь выполняется поиск пользователя в базе данных, проверяя указанный email и подтверждённый статус аккаунта. Если пользователь не найден, возвращается сообщение об ошибке с указанием недействительных данных. Функция представлена в листинге 3.14.

```
static async login(req, res, next) {
        try {
          const { email, password } = req.body;
          const userData = await UserService.login(email, password);

          if (userData.user.isBanned == 1) {
            return res.status(403).json({ message: "Пользователь заблокирован" });
          }

          res.cookie("refreshToken", userData.refreshToken, {
            maxAge: 30 * 24 * 60 * 60 * 1000,
            httpOnly: true,
          });
          return res.json(userData);
        } catch (e) {
          next(e);
        }
  }
```

Листинг 3.14 – Функция авторизации Клиента

Если пользователь найден, осуществляется проверка правильности введённого пароля с использованием библиотеки bcrypt. В случае некорректного пароля возвращается сообщение об ошибке. При успешной проверке генерируются токены Access Token и Refresh Token, которые упаковываются в cookies для последующей работы клиента с системой.

Успешная авторизация возвращает объект, содержащий сообщение о результате, сгенерированный Access Token, а также информацию о пользователе, включая его идентификатор, имя, роль и дополнительные данные. В случае сбоев в процессе работы сервера или некорректных данных отправляется общее сообщение об ошибке.

3.6.3 Реализация функции регистрации управляющего приютом

Функция create в PetShelterController отвечает за процесс авторизации пользователя в системе под ролью управляющего приютом. Она обрабатывает HTTP POST-запросы, принимая данные приюта, такие как название, адрес, электронная почта, пароль, описание, город и страна. В начале функции выполняется проверка наличия всех обязательных полей в теле запроса. Если какое-либо из обязательных полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).
Если все необходимые поля присутствуют, функция вызывает сервис PetShelterService.create, передавая ему данные для создания нового приюта. Функция представлена в листинге 3.15.

```
  static async create(req, res, next) {
    try {
      const { name, address, email, password, description, city, country } =
        req.body;

      if (
        !name ||
        !address ||
        !email ||
        !password ||
        !description ||
        !city ||
        !country
      ) {
        return res
          .status(400)
          .json({ error: "Все поля обязательны для создания приюта" });
      }

      const petShelter = await PetShelterService.create({
        name,
        address,
        email,
        password,
        description,
        isBanned: 0,
        city,
        country,
```

```
            });
            return res.status(201).json(petShelter);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при создании приюта", message: e.message });
          }
}
```

Листинг 3.15 – Функция авторизации Управляющего приютом

При успешном создании приюта возвращается объект созданного приюта с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.6.4 Реализация функции авторизации управляющего приютом

Функция login в PetShelterController реализует процесс извлечения списка статей из базы данных и их предоставления гостю. Она обрабатывает HTTP GET-запросы и возвращает данные о статьях, включая идентификатор, заголовок, текст, дату публикации, URL изображения, количество лайков, а также информацию об авторе. Функция представлена в листинге 3.16.

```
        static async login(req, res, next) {
          try {
            const { email, password } = req.body;

            if (!email || !password) {
              return res
                .status(400)
                .json({ error: "Email и пароль обязательны для входа" });
            }

            const petShelterData = await PetShelterService.authenticate(
              email,
              password
            );

            // Проверка, заблокирован ли пользователь
            if (petShelterData?.user?.isBanned == 1) {
```

```
            return res.status(403).json({ message: "Пользователь заблокирован" });
          }

          res.cookie("refreshToken", petShelterData.refreshToken, {
            maxAge: 30 * 24 * 60 * 60 * 1000,
            httpOnly: true,
          });
          return res.status(200).json(petShelterData);
        } catch (e) {
          return res
            .status(401)
            .json({ error: "Ошибка при авторизации", message: e.message });
        }
  }
```

Листинг 3.16 – Функция авторизации Управляющего приютом

Если пользователь найден, осуществляется проверка правильности введённого пароля с использованием библиотеки bcrypt. В случае некорректного пароля возвращается сообщение об ошибке. При успешной проверке генерируются токены Access Token и Refresh Token, которые упаковываются в cookies для последующей работы клиента с системой.

Успешная авторизация возвращает объект, содержащий сообщение о результате, сгенерированный Access Token, а также информацию о управляющем приютом, включая его идентификатор, имя, роль и дополнительные данные. В случае сбоев в процессе работы сервера или некорректных данных отправляется общее сообщение об ошибке.

3.7 Реализация функций для пользователя

3.7.1 Реализация функции просмотра каталога животных

Функция search в AnimalController отвечает за процесс получения списка животных из базы данных для отображения гостям или пользователям системы. Она обрабатывает HTTP GET-запросы, принимая параметры фильтрации и поиска из строки запроса (req.query). Функция представлена в листинге 3.17.

```
static async search(req, res, next) {
        try {
          let animals;
          animals = await AnimalService.getAll({
            ...req.query,
```

```
            });
            return res.status(200).json(animals);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при получении животных", message: e.message
});
          }
  }
```

Листинг 3.17 – Функция просмотра каталога животных

В начале функции вызывается сервис AnimalService.getAll для получения списка животных. Сервис обрабатывает запрос и возвращает список животных, соответствующих заданным параметрам.

Если операция успешна, функция возвращает список животных с кодом состояния 200 (OK). В случае возникновения ошибки при получении данных из базы данных или во время обработки запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.2 Реализация функции просмотра страницы животного

Функция getById в AnimalController отвечает за получение детальной информации о конкретном животном по его идентификатору (ID). Она обрабатывает HTTP GET-запросы, принимая параметр id из маршрута (req.params). Функция представлена в листинге 3.18.

```
  static async getById(req, res) {
    try {
      const { id } = req.params;

      // Проверка: ID должен быть числом и присутствовать
      if (!id || isNaN(Number(id))) {
        return res.status(400).json({
          error: "Некорректный ID. Пожалуйста, укажите правильный ID.",
        });
      }

      const animal = await AnimalService.getById(Number(id));

      if (!animal) {
```

```
            return res
              .status(404)
              .json({ error: "Животное с таким ID не найдено" });
          }

          return res.status(200).json(animal);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при получении животного", message:
e.message });
        }
  }
```

Листинг 3.18 – Функция просмотра страницы животного

В начале функции выполняется проверка валидности полученного ID. Если id отсутствует или не является числом, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если ID корректен, функция вызывает сервис AnimalService.getById, передавая ему числовое значение ID для поиска животного в базе данных. Если животное с указанным ID не найдено, возвращается сообщение об ошибке с кодом состояния 404 (Not Found).

В случае успешного нахождения животного, функция возвращает объект с информацией о животном и кодом состояния 200 (OK). В случае возникновения ошибки во время выполнения запроса или взаимодействия с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.3 Реализация функции поиска животных

Функция search в AnimalController предназначена для выполнения поиска животных по различным критериям. Она обрабатывает HTTP GET-запросы, принимая параметры поиска из строки запроса (req.query). Функция представлена в листинге 3.19.

```
static async search(req, res, next) {
        try {
          let animals;
          animals = await AnimalService.getAll({
            ...req.query,
          });
```

```
            return res.status(200).json(animals);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при получении животных", message: e.message
});
          }
  }
```

Листинг 3.19 – Функция поиска животных

В теле функции вызывается сервис AnimalService.getAll, которому передаётся параметр page из запроса для выполнения соответствующего поиска. Сервис обрабатывает запрос, применяя заданные параметры, и возвращает список животных, соответствующих странице.

Если операция выполняется успешно, функция возвращает список найденных животных с кодом состояния 200 (OK). В случае возникновения ошибки при обработке запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке..

3.7.4 Реализация функции фильтрации по критериям

Функция search в AnimalController реализует процесс фильтрации списка животных по заданным критериям. Она обрабатывает HTTP GET-запросы, принимая параметры фильтрации из строки запроса (req.query). Функция представлена в листинге 3.20.

```
static async search(req, res, next) {
          try {
            let animals;
            animals = await AnimalService.getAll({
              ...req.query,
            });
            return res.status(200).json(animals);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при получении животных", message: e.message
});
          }
  }
```

Листинг 3.20 – Функция фильтрации по критериям

Функция вызывает сервис AnimalService.getAll, передавая ему все параметры из запроса для применения фильтров и поиска соответствующих животных. Сервис обрабатывает запрос, фильтруя данные в соответствии с переданными критериями, и возвращает список животных, соответствующих этим критериям.

При успешном выполнении операции функция возвращает отфильтрованный список животных с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.5 Реализация функции просмотра каталога типов животных

Функция getAll в AnimalTypeController отвечает за получение списка всех типов животных, доступных в системе. Она обрабатывает HTTP GET-запросы и не принимает дополнительных параметров. Функция представлена в листинге 3.21.

```
static async getAll(req, res) {
            try {
              const animalTypes = await AnimalTypeService.getAll();
              return res.status(200).json(animalTypes);
            } catch (error) {
              return res.status(500).json({
                error: "Ошибка при получении типов животных",
                message: error.message,
              });
            }
  }
```

Листинг 3.21 – Функция просмотра каталога типов животных

В теле функции вызывается сервис AnimalTypeService.getAll, который извлекает все записи из таблицы типов животных в базе данных. Полученные данные содержат информацию о каждом типе животного, такую как название и описание.

Если операция выполняется успешно, функция возвращает список типов животных с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает

сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.6 Реализация функции просмотра каталога заболеваний

Функция getAll в DiseaseController предназначена для получения списка всех заболевани. Она обрабатывает HTTP GET-запросы, принимая параметры фильтрации из строки запроса (req.query). Функция представлена в листинге 3.22.

```
static async getAll(req, res) {
        try {
          const diseases = await DiseaseService.getAll(req.query);
          return res.status(200).json(diseases);
        } catch (error) {
          return res.status(500).json({
            error: "Ошибка при получении списка заболеваний",
            message: error.message, });}}
```

Листинг 3.22 – Функция просмотра каталога заболеваний

Функция вызывает сервис DiseaseService.getAll, для получения списка заболеваний. Сервис обрабатывает запрос и возвращает список заболеваний.

При успешном выполнении операции функция возвращает список заболеваний с кодом состояния 200 (OK). В случае возникновения ошибки при обработке запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.7 Реализация функции поиска заболевания

Функция getAll в DiseaseController реализует процесс поиска заболеваний по заданным критериям. Она обрабатывает HTTP GET-запросы, принимая параметры поиска из строки запроса (req.query). Функция представлена в листинге 3.23.

```
static async getAll(req, res) {
        try {
          const diseases = await DiseaseService.getAll(req.query);
          return res.status(200).json(diseases);
        } catch (error) {
          return res.status(500).json({
            error: "Ошибка при получении списка заболеваний",
            message: error.message,
```

```
            });
        }
}
```

Листинг 3.23 – Функция поиска заболеваний

Функция вызывает сервис DiseaseService.getAll, передавая ему параметры из запроса для выполнения соответствующего поиска и фильтрации заболеваний. Сервис обрабатывает запрос, применяя заданные параметры, и возвращает список заболеваний, соответствующих критериям поиска.

Если операция выполняется успешно, функция возвращает список найденных заболеваний с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.7.8 Реализация функции просмотра критериев

Функция getAll в CriteriaController отвечает за получение списка всех критериев оценки животных, используемых в системе. Она обрабатывает HTTP GET-запросы и не принимает дополнительных параметров. Функция представлена в листинге 3.24.

```
static async getAll(req, res) {
        try {
          const criteria = await CriteriaService.getAll();
          return res.status(200).json(criteria);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при получении критериев", message: e.message
});
        }
 }
```

Листинг 3.24 – Функция просмотра критериев

В теле функции вызывается сервис CriteriaService.getAll, который извлекает все записи из таблицы критериев в базе данных. Полученные данные содержат информацию о каждом критерии, такую как название и описание.

Если операция выполняется успешно, функция возвращает список критериев с кодом состояния 200 (OK). В случае возникновения ошибки при

выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.8 Реализация функций для клиента

3.8.1 Реализация функции создания запроса на получение животного

Функция create в AdoptionRequestController отвечает за создание нового запроса на усыновление животного пользователем. Она обрабатывает HTTP POST-запросы, принимая идентификаторы животного (animalId), пользователя (userId) и описание заявки из тела запроса (req.body). Функция представлена в листинге 3.25.

```
static async create(req, res) {
        try {
          const { animalId, userId, description } = req.body;
          if (!animalId || !userId || !description) {
            return res
              .status(400)
              .json({ error: "Все поля обязательны для создания заявки" });
          }
          const request = await AdoptionRequestService.create({
            animalId,
            userId,
            description,
          });
          return res.status(201).json(request);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при создании заявки", message: e.message });
        }
  }
```

Листинг 3.25 – Функция создания запрсоа на получения животного

В начале функции выполняется проверка наличия всех обязательных полей (animalId, userId, description). Если какое-либо из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис AdoptionRequestService.create, передавая ему данные для создания новой заявки.

При успешном создании заявки возвращается объект созданной заявки с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.8.2 Реализация функции просмотр запросов на получение животного из приюта

Функция getAll в AdoptionRequestController реализует процесс получения списка заявок на усыновление животных. Она обрабатывает HTTP GET-запросы и возвращает данные о заявках в зависимости от роли пользователя, делающего запрос. Функция представлена в листинге 3.26.

```
static async getAll(req, res) {
        try {
          let requests;
          if (req.user.role == "shelterManager") {
            requests = await AdoptionRequestService.getAll({
              ...req.query,
              petShelterId: req.user.id,
            });
          } else if (req.user.role == "user") {
            requests = await AdoptionRequestService.getAll({
              ...req.query,
              UserId: req.user.id,
            });
          } else {
            return res
            .status(500)
            .json({ error: "Ошибка при получении заявок” });
          }
          return res.status(200).json(requests);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при получении заявок", message: e.message });
        }
  }
```

Листинг 3.26 – Функция просмотра запросов на получения животного из приюта

В теле функции выполняется проверка роли пользователя (req.user.role):

Если роль пользователя — "shelterManager" (управляющий приютом), функция вызывает сервис AdoptionRequestService.getAll, передавая ему параметры запроса и идентификатор приюта (petShelterId: req.user.id), что позволяет получить только те заявки, которые относятся к конкретному приюту.

Если роль пользователя — "user" (клиент), функция вызывает тот же сервис, но с параметром UserId: req.user.id, что позволяет получить только те заявки, которые созданы конкретным пользователем.

Если операция выполняется успешно, функция возвращает список заявок с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9 Реализация функций для управляющего приютом

3.9.1 Реализация функции просмотра животных в приюте

Функция search в AnimalController отвечает за получение списка животных, находящихся в конкретном приюте. Она обрабатывает HTTP GET-запросы, принимая параметры фильтрации из строки запроса (req.query). Функция представлена в листинге 3.27.

```
static async search(req, res, next) {
            try {
              let animals;
              animals = await AnimalService.getAll({
                ...req.query,
              });
              return res.status(200).json(animals);
            } catch (e) {
              return res
                .status(500)
                .json({ error: "Ошибка при получении животных", message: e.message
});
            }
  }
```

Листинг 3.27 – Функция просмотра животных в приюте

В теле функции вызывается сервис AnimalService.getAll, которому передаются параметры из запроса для выполнения соответствующего поиска и

фильтрации животных. Сервис обрабатывает запрос, применяя заданные параметры, и возвращает список животных, соответствующих критериям.

Если операция выполняется успешно, функция возвращает список животных с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.2 Реализация функции добавления животного в приют

Функция create в AnimalController отвечает за процесс добавления нового животного в приют. Она обрабатывает HTTP POST-запросы, принимая данные животного, такие как идентификатор приюта (petShelterId), тип животного (animalTypeId), имя, описание, возраст и пол из тела запроса (req.body). Функция представлена в листинге 3.28.

```
static async create(req, res, next) {
        try {
          const { petShelterId, animalTypeId, name, description, age, gender } =
            req.body;

          if (
            !petShelterId ||
            !animalTypeId ||
            !name ||
            !description ||
            !age ||
            !gender
          ) {
            return res
              .status(400)
              .json({ error: "Все поля обязательны для заполнения" });
          }

          if (isNaN(parseInt(age))) {
            return res.status(400).json({ error: "Age должно быть числом" });
          }

          const animal = await AnimalService.create({
            petShelterId,
            animalTypeId,
            name,
```

```
            description,
            age: parseInt(age),
            gender,
          });
          return res.status(201).json(animal);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при создании животного", message: e.message
});
        }
  }
```

Листинг 3.28 – Функция добавлекния животного в приют

В начале функции выполняется проверка наличия всех обязательных полей. Если какое-либо из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request). Дополнительно выполняется проверка, является ли поле age числом. Если возраст не является числовым значением, также возвращается ошибка с кодом состояния 400.

Если все проверки проходят успешно, функция вызывает сервис AnimalService.create, передавая ему данные для создания нового животного. При успешном добавлении животного возвращается объект созданного животного с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.3 Реализация функции удаления животного из приюта

Функция delete в AnimalController отвечает за удаление животного из приюта по его идентификатору (ID). Она обрабатывает HTTP DELETE-запросы, принимая параметр id из маршрута (req.params). Функция представлена в листинге 3.29.

```
      static async delete(req, res, next) {
        try {
          const { id } = req.params;

          if (!id) {
            return res
              .status(400)
              .json({ error: "ID животного обязателен для удаления" });
```

```
            }

            await AnimalsController.checkOwnership(id, req.user.id);

            await AnimalService.delete(parseInt(id));
            return res.status(200).json({ message: "Животное успешно удалено" });
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при удалении животного", message: e.message
});
          }
  }
```

Листинг 3.29 – Функция удаления животного из приюта

В теле функции вызывается метод AnimalModel.delete, передавая ему идентификатор животного для удаления из базы данных. Если операция выполняется успешно, функция возвращает подтверждение удаления. В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.4 Реализация функции редактирование животного в приюте

Функция update в AnimalController отвечает за обновление информации о животном в приюте. Она обрабатывает HTTP PUT запрос, принимая данные для обновления из тела запроса (req.body). Функция представлена в листинге 3.30.

```
static async update(req, res, next) {
          try {
            const { ID, petShelterId, animalTypeId, name, description } = req.body;

            // Проверка на обязательные поля
            if (!ID) {
              return res
                .status(400)
                .json({ error: "Все поля обязательны для обновления" });
            }

            const updatedAnimal = await AnimalService.update(ID, {
              petShelterId,
```

```
            animalTypeId,
            name,
            description,
            age,
            gender,
          });
          return res.status(200).json(updatedAnimal);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при обновлении животного", message:
e.message });
        }
  }
```

Листинг 3.30 – Функция редактирования животного в приюте

В начале функции извлекаются необходимые поля из тела запроса: идентификатор животного (ID), идентификатор приюта (petShelterId), тип животного (animalTypeId), имя, описание, возраст и пол.

Функция выполняет проверку наличия обязательного поля ID. Если оно отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если идентификатор присутствует, функция вызывает сервис AnimalService.update, передавая ему идентификатор животного и данные для обновления. При успешном обновлении информации о животном функция возвращает объект обновленного животного с кодом состояния 200 (OK).

В случае возникновения ошибки при обновлении животного или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.5 Реализация функции добавления заболевания животному

Функция create в AnimalDiseasesController отвечает за добавление заболевания конкретному животному. Она обрабатывает HTTP POST-запросы, принимая идентификаторы животного (animalId) и заболевания (diseasesId) из тела запроса (req.body). Функция представлена в листинге 3.31.

```
static async create(req, res) {
        const { animalId, diseasesId } = req.body;
        try {
          if (!animalId || !diseasesId) {
```

```
            return res
              .status(400)
              .json({ error: "AnimalId и DiseasesId обязательны" });
          }
          const record = await AnimalDiseasesService.create(animalId, diseasesId);
          return res.status(201).json(record);
        } catch (error) {
          return res
            .status(500)
            .json({ error: "Ошибка при создании записи", message: error.message
});
        }
  }
```

Листинг 3.31 – Функция добавления заболевания животному

В начале функции выполняется проверка наличия обязательных полей animalId и diseasesId. Если одно из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис AnimalDiseasesService.create, передавая ему идентификаторы животного и заболевания для создания новой записи в связующей таблице. При успешном добавлении связи возвращается объект созданной записи с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.6 Реализация функции удаления заболевания у животного

Функция delete в AnimalDiseasesController отвечает за удаление связи между животным и заболеванием по идентификатору записи (ID). Она обрабатывает HTTP DELETE-запросы, принимая параметр id из маршрута (req.params).. Функция представлена в листинге 3.32.

```
      static async delete(req, res) {
        const { id } = req.params;
        try {
          if (isNaN(parseInt(id))) {
            return res.status(400).json({ error: "id должно быть числом" });
          }
          const deletedRecord = await AnimalDiseasesService.delete(parseInt(id));
          return res.status(200).json(deletedRecord);
```

```
        } catch (error) {
          return res
            .status(500)
            .json({ error: "Ошибка при удалении записи", message: error.message
});
        }
  }
```

Листинг 3.32 – Функция удаления заболевания у животного

В теле функции выполняется проверка, является ли полученный id числом. Если id не является числовым значением, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если проверка проходит успешно, функция вызывает сервис AnimalDiseasesService.delete, передавая ему числовой идентификатор записи для удаления из базы данных. При успешном удалении функции возвращается объект удаленной записи с кодом состояния 200 (OK). В случае возникновения ошибки при удалении записи или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.7 Реализация функции удаление критерия у животного

Функция delete в AnimalCriteriaController отвечает за удаление связи между животным и критерием оценки по идентификатору записи (ID). Она обрабатывает HTTP DELETE-запросы, принимая параметр id из маршрута (req.params). Функция представлена в листинге 3.33.

```
static async delete(req, res) {
        try {
          const { id } = req.params;

          if (!id || isNaN(Number(id))) {
            return res.status(400).json({ error: "Некорректный ID" });
          }

          await AnimalCriteriaService.delete(Number(id));
          return res.status(200).json({ message: "Связь успешно удалена" });
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при удалении связи", message: e.message });
```

```
        }
  }
```

Листинг 3.33 – Функция удаления критерия у животного

В начале функции выполняется проверка наличия и валидности полученного id. Если id отсутствует или не является числом, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если проверка проходит успешно, функция вызывает сервис AnimalCriteriaService.delete, передавая ему числовой идентификатор записи для удаления из базы данных. При успешном удалении связи функция возвращает сообщение о успешном удалении с кодом состояния 200 (OK). В случае возникновения ошибки при удалении связи или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.8 Реализация функции добавление критерия животному

Функция create в AnimalCriteriaController отвечает за создание связи между животным и критерием. Она обрабатывает HTTP POST-запросы, принимая идентификаторы животного (animalId) и критерия (criteriaId) из тела запроса (req.body). Функция представлена в листинге 3.34.

```
static async create(req, res) {
        try {
          const { animalId, criteriaId } = req.body;

          if (!animalId || !criteriaId) {
            return res
              .status(400)
              .json({ error: "Animal ID и Criteria ID обязательны" });
          }

          const link = await AnimalCriteriaService.create(animalId, criteriaId);
          return res.status(201).json(link);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при создании связи", message: e.message });
        }
  }
```

Листинг 3.34 – Функция добавления критерия животному

В начале функции выполняется проверка наличия обязательных полей animalId и criteriaId. Если одно из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис AnimalCriteriaService.create, передавая ему идентификаторы животного и критерия для создания новой записи в связующей таблице. При успешном создании связи возвращается объект созданной записи с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.9 Реализация функции просмотра запросов на получение животного из приюта

Функция getAll в AdoptionRequestController отвечает за получение списка всех заявок на получение животных, поступивших в приют. Она обрабатывает HTTP GET-запросы и возвращает данные о заявках в зависимости от роли пользователя, делающего запрос. Функция представлена в листинге 3.35.

```
static async getAll(req, res) {
        try {
          let requests;
          if (req.user.role == "shelterManager") {
            requests = await AdoptionRequestService.getAll({
              ...req.query,
              petShelterId: req.user.id,
            });
          } else if (req.user.role == "user") {
            requests = await AdoptionRequestService.getAll({
              ...req.query,
              UserId: req.user.id,
            });
          } else {
            return res
            .status(500)
            .json({ error: "Ошибка при получении заявок" });
          }
          return res.status(200).json(requests);
        } catch (e) {
          return res
```

```
            .status(500)
            .json({ error: "Ошибка при получении заявок", message: e.message });
        }
}
```

Листинг 3.35 – Функция просмотра запросов на получение животного из приюта

В теле функции выполняется проверка роли пользователя (req.user.role):

Если роль пользователя — "shelterManager" (управляющий приютом), функция вызывает сервис AdoptionRequestService.getAll, передавая ему параметры запроса и идентификатор приюта (petShelterId: req.user.id), что позволяет получить только те заявки, которые относятся к конкретному приюту.

Если роль пользователя — "user" (клиент), функция вызывает тот же сервис, но с параметром UserId: req.user.id, что позволяет получить только те заявки, которые созданы конкретным пользователем.

Если операция выполняется успешно, функция возвращает список заявок с кодом состояния 200 (OK). В случае возникновения ошибки при выполнении запроса или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.9.10 Реализация функции создания ответа на запрос пользователя о получении животного

Функция update в AdoptionRequestController отвечает за обновление статуса заявки на усыновление животного, включая добавление ответа от управляющего приютом. Она обрабатывает HTTP PUT запрос, принимая параметры id из маршрута (req.params) и данные для обновления из тела запроса (req.body). Функция представлена в листинге 3.36.

```
static async update(req, res) {
        try {
          const { id } = req.params;
          const { animalId, userId, isProcessed, answer } = req.body;

          const updatedRequest = await
AdoptionRequestService.update(parseInt(id), {
            AnimalId: animalId ? parseInt(animalId) : undefined,
            UserId: userId ? parseInt(userId) : undefined,
            isProcessed: isProcessed ? parseInt(isProcessed) : undefined,
            answer: answer,
```

```
            });
            return res.status(200).json(updatedRequest);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при обновлении заявки", message: e.message });
          }
}
```

Листинг 3.36 – Функция создания ответа на запрос пользователя о получении животного

В теле функции извлекаются следующие поля из запроса: идентификатор заявки (id), идентификаторы животного (animalId) и пользователя (userId), статус обработки заявки (isProcessed) и ответ (answer). Функция вызывает сервис AdoptionRequestService.update, передавая ему идентификатор заявки и объект с обновленными данными.

Если операция обновления выполняется успешно, функция возвращает обновленный объект заявки с кодом состояния 200 (OK). В случае возникновения ошибки при обновлении заявки или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10 Реализация функций для администратора

3.10.1 Реализация функции блокировки клиента

Функция ban в UserController отвечает за изменение статуса блокировки клиента в системе. Она обрабатывает HTTP PUT запрос, принимая идентификатор пользователя (id) и статус блокировки (ban) из тела запроса (req.body). Функция представлена в листинге 3.37.

```
        static async ban(req, res, next) {
          try {
            const { id, ban } = req.body;

            if (!id) {
              return res
                .status(400)
                .json({ error: "ID обязателен для обновления пользователя" });
            }
```

```
            const updatedPetShelter = await UserService.update(id, {
              isBanned: ban,
            });
            return res.status(200).json(updatedPetShelter);
          } catch (e) {
            return res.status(500).json({
              error: "Ошибка при изменении пользователя",
              message: e.message,
            });
          }
  }
```

Листинг 3.37 – Функция блокировки клиента

В начале функции выполняется проверка наличия обязательного поля id. Если id отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если идентификатор присутствует, функция вызывает сервис UserService.update, передавая ему идентификатор пользователя и объект с обновленным полем isBanned. Значение isBanned указывает на статус блокировки пользователя (1 — заблокирован, 0 — не заблокирован).

При успешном обновлении статуса блокировки функция возвращает объект обновленного пользователя с кодом состояния 200 (OK). В случае возникновения ошибки при обновлении пользователя или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.2 Реализация функции блокировка управляющего приютом

Функция ban в PetShelterController отвечает за изменение статуса блокировки управляющего приютом в системе. Она обрабатывает HTTP PUT-запрос, принимая идентификатор приюта (id) и статус блокировки (ban) из тела запроса (req.body).. Функция представлена в листинге 3.38.

```
static async ban(req, res, next) {
          try {
            const { id, ban } = req.body;

            if (!id) {
              return res
                .status(400)
                .json({ error: "ID обязателен для обновления приюта" });
```

```
            }
            const updatedPetShelter = await PetShelterService.update(id, {
              isBanned: ban,
            });
            return res.status(200).json(updatedPetShelter);
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при обновлении приюта", message: e.message
});
          }
  }
```

Листинг 3.38 – Функция блокировки управляющего приютом

В начале функции выполняется проверка наличия обязательного поля id. Если id отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если идентификатор присутствует, функция вызывает сервис PetShelterService.update, передавая ему идентификатор приюта и объект с обновленным полем isBanned. Значение isBanned указывает на статус блокировки приюта (1 — заблокирован, 0 — не заблокирован).

При успешном обновлении статуса блокировки функция возвращает объект обновленного приюта с кодом состояния 200 (OK). В случае возникновения ошибки при обновлении приюта или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.3 Реализация функции добавления типа животного

Функция create в AnimalTypeController отвечает за создание нового типа животного в системе. Она обрабатывает HTTP POST-запросы, принимая название типа животного (TypeName) и описание (Description) из тела запроса (req.body). Функция представлена в листинге 3.39.

```
static async create(req, res) {
        const { TypeName, Description } = req.body;
        try {
          if (!TypeName || !Description) {
            return res
              .status(400)
              .json({ error: "Не все обязательные поля заполнены" });
```

```
            }
            const animalType = await AnimalTypeService.create({
              TypeName,
              Description,
            });
            return res.status(201).json(animalType);
          } catch (error) {
            return res.status(500).json({
              error: "Ошибка при создании типа животного",
              message: error.message,
            });
          }
  }
```

Листинг 3.39 – Функция добавлекния типа животного

В начале функции выполняется проверка наличия обязательных полей TypeName и Description. Если одно из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис AnimalTypeService.create, передавая ему данные для создания нового типа животного. При успешном создании типа животного возвращается объект созданного типа с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.4 Реализация функции удаление типа животного

Функция delete в AnimalTypeController отвечает за удаление типа животного из системы по его идентификатору (ID). Она обрабатывает HTTP DELETE-запросы, принимая параметр id из маршрута (req.params).. Функция представлена в листинге 3.40.

```
static async delete(req, res) {
          const { id } = req.params;
          try {
            const deletedAnimalType = await AnimalTypeService.delete(id);
            return res.status(200).json(deletedAnimalType);
          } catch (error) {
            return res.status(500).json({
              error: "Ошибка при удалении типа животного",
```

```
            message: error.message,
          });
        }
}
```

Листинг 3.40 – Функция удаленяи типа животного

В теле функции вызывается сервис AnimalTypeService.delete, передавая ему идентификатор типа животного для удаления из базы данных. Если операция выполняется успешно, функция возвращает объект удаленного типа животного с кодом состояния 200 (OK).

В случае возникновения ошибки при удалении типа животного или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.5 Реализация функции добавления заболевания

Функция create в DiseaseController отвечает за создание нового заболевания в системе. Она обрабатывает HTTP POST-запросы, принимая название заболевания (name) и описание (description) из тела запроса (req.body). Функция представлена в листинге 3.41.

```
        static async create(req, res) {
          const { name, description } = req.body;
          try {
            if (!name || !description) {
              return res
                .status(400)
                .json({ error: "Поля name и description обязательны" });
            }
            const disease = await DiseaseService.create({ name, description });
            return res.status(201).json(disease);
          } catch (error) {
            return res.status(500).json({
              error: "Ошибка при создании заболевания",
              message: error.message,
            });
          }
}
```

Листинг 3.41 – Функция добавления заболевания

В начале функции выполняется проверка наличия обязательных полей name и description. Если одно из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис DiseaseService.create, передавая ему данные для создания нового заболевания. При успешном создании заболевания возвращается объект созданного заболевания с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.6 Реализация функции удаления заболевания

Функция delete в DiseaseController отвечает за удаление заболевания из системы по его идентификатору ID. Она обрабатывает HTTP DELETE-запросы, принимая параметр id из маршрута req.params. Функция представлена в листинге 3.42.

```
static async delete(req, res) {
        const { id } = req.params;
        try {
          const numId = parseInt(id);
          if (isNaN(numId)) {
            return res.status(404).json({ error: "ID должно быть числом" });
          }
          const deletedDisease = await DiseaseService.delete(numId);
          return res.status(200).json(deletedDisease);
        } catch (error) {
          return res.status(500).json({
            error: "Ошибка при удалении заболевания",
            message: error.message,
          });
        }
  }
```

Листинг 3.42 – Функция удаления заболевания

В теле функции происходит конвертация полученного id в число (numId). Если конвертация не удалась и id не является числом, возвращается сообщение об ошибке с кодом состояния 404 (Not Found) и текстом "ID должно быть числом".

Если идентификатор корректен, функция вызывает сервис DiseaseService.delete, передавая ему числовой идентификатор для удаления из

базы данных. При успешном удалении заболевания функция возвращает объект удаленного заболевания с кодом состояния 200 (OK).

В случае возникновения ошибки при удалении заболевания или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.7 Реализация функции создания критерия

Функция create в CriteriaController отвечает за создание нового критерия животного в системе. Она обрабатывает HTTP POST-запросы, принимая название критерия name и описание description из тела запроса req.body. Функция представлена в листинге 3.43.

```
static async create(req, res) {
        try {
          const { name, description } = req.body;

          // Проверка на обязательные поля
          if (!name || !description) {
            return res
              .status(400)
              .json({ error: "Имя и описание обязательны для создания" });
          }

          const criterion = await CriteriaService.create({
            Name: name,
            Description: description,
          });
          return res.status(201).json(criterion);
        } catch (e) {
          return res
            .status(500)
            .json({ error: "Ошибка при создании критерия", message: e.message });
        }
  }
```

Листинг 3.43 – Функция создания критерия

В начале функции выполняется проверка наличия обязательных полей name и description. Если одно из этих полей отсутствует, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request).

Если все необходимые поля присутствуют, функция вызывает сервис CriteriaService.create, передавая ему данные для создания нового критерия. При успешном создании критерия возвращается объект созданного критерия с кодом состояния 201 (Created). В случае возникновения ошибки на сервере или при обработке запроса, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.10.8 Реализация функции удаления критерия

Функция delete в CriteriaController отвечает за удаление критерия животного по его идентификатору (ID). Она обрабатывает HTTP DELETE-запросы, принимая параметр id из req.params. Функция представлена в листинге 3.44.

```
        static async delete(req, res) {
          try {
            const { id } = req.params;

            // Проверка: ID должен быть числом и присутствовать
            if (!id || isNaN(Number(id))) {
              return res.status(400).json({
                error: "Некорректный ID. Пожалуйста, укажите правильный ID.",
              });
            }

            await CriteriaService.delete(Number(id));
            return res.status(200).json({ message: "Критерий успешно удалён" });
          } catch (e) {
            return res
              .status(500)
              .json({ error: "Ошибка при удалении критерия", message: e.message
});
          }
  }
```

Листинг 3.44 – Функция удаление критерия

В начале функции выполняется проверка наличия и валидности полученного id. Если id отсутствует или не является числом, возвращается сообщение об ошибке с кодом состояния 400 (Bad Request) и текстом "Некорректный ID. Пожалуйста, укажите правильный ID."

Если проверка проходит успешно, функция вызывает сервис CriteriaService.delete, передавая ему числовой идентификатор для удаления из базы данных. При успешном удалении критерия функция возвращает сообщение о успешном удалении с кодом состояния 200 (OK).

В случае возникновения ошибки при удалении критерия или взаимодействии с базой данных, функция возвращает сообщение об ошибке с кодом состояния 500 (Internal Server Error) и детальной информацией об ошибке.

3.11 Структура клиентской части

3.11.1 Реализация структуры проекта

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории src. Директории представлены в таблице 3.6

Таблица 3.6 – Директории проекта в папке src и их описание

Директория	Описание
components	Содержит компоненты приложения, такие как кнопки, элементы навигации, элементы управления, а также отдельные структурные элементы. Также включает элементы для отображения различных сущностей, таких как статьи или комментарии.
const	Содержит интерфейсы, используемые для типизации данных.
image	Содержит статические изображения используемые в интерфейсе.
pages	Содержит страницы приложения, соответствующие различным маршрутам. Эти страницы обрабатывают основные пользовательские действия.
store	Реализует управление состоянием приложения. Включает директории для пользователем или управляющим приютом.
utils	Содержит вспомогательные модули и утилиты.
api	Содержит сервисы для отправки запросов.

3.11.2 Реализация компонент

Компоненты в проекте играют центральную роль, предоставляя гибкие и переиспользуемые элементы пользовательского интерфейса. Они включают как небольшие элементы (например, кнопки), так и более сложные структуры (например, формы или меню). В таблица 3.7 представлено описание компонент.

Таблица 3.7 – Описание компонент

Компонента	Описание

header	Реализует верхний колонтитул приложения, отображая название и основные навигационные элементы.
footer	Реализует нижний колонтитул приложения с краткой информацией, ссылками на социальные сети и контактными данными.
GetAllDiseasesForm	Компонент для отображения всех заболеваний, доступных в системе.
GetAllCriteria	Отображает все доступные критерии для отбора животных.
GetAllAnimalTypesForm	Выводит список всех доступных типов животных в системе.
animalCarousel	Компонент, реализующий карусель для отображения изображений животных.
GetAllAnimalsForm	Отображает список всех животных, зарегистрированных в системе, с их базовой информацией, возможностью фильтрации и сортировки.
CreateAdoptionRequestForm	Компонент для создания новой заявки на усыновление животного. Предоставляет форму для заполнения необходимой информации и отправки данных.
GetAllAdoptionRequestsForm	Выводит список всех заявок на усыновление.
UpdateAdoptionRequestForm	Обеспечивает возможность редактирования существующей заявки на усыновление.
CriteriaList	Отображает список критериев для отбора животных.
Dropdown	Выпадающий список для выбора одного из доступных элементов.
Paginator	Реализует пагинацию для навигации между страницами данных. Поддерживает перелистывание страниц вперёд и назад, а также быстрый переход на конкретную страницу.

3.12 Выводы по разделу

1. Сервер разработан на платформе Node.js с использованием фреймворка Express.js, в совокупности это обеспечивает гибкость и производительность приложению.

2. В качестве СУБД использована MSSQL для хранения данных о пользователях, домах, квартирах, бронированиях и токенах.

3. Структура приложения основана на модульном подходе с применением современных библиотек для клиентской и серверной частей.

4. Для передачи данных используется JSON для текстовой информации.

5. Реализованы все функции для всех ролей: гостя, клиента, управляющего приютом, администратора. Количество функций веб-приложения составило 30.

4. Тестирование веб-приложения

4.1 Функциональное тестирование

Для проведения корректной работы функций разработанного веб-приложения было произведено ручное тестирование, описание и итоги представлены в таблице 4.1.

Таблица 4.1 – Описание функциональных тестов

Номер теста	Функция веб-приложения	Описание теста	Ожидаемый результат	Статус теста
1	Регистрация клиента	Действие: Ввести корректные данные (username, email, password, phone number) и отправить форму.	Данные пользователя заносятся в базу данных.	Пройден
2	Авторизация клиента	Действие: Ввести корректные данные для входа (username, password) и нажать "Войти".	Пользователь успешно авторизуется и попадает на главную страницу.	Пройден
3	Просмотр каталога животных	Действие: Перейти на страницу с каталогом животных.	Животные отображаются на странице.	Пройден
4	Просмотр каталога болезней	Действие: Открыть страницу с болезнями.	Болезни отображаются на странице.	Пройден
5	Просмотр каталога типов животных	Действие: Открыть страницу с типами животных.	Типы животных отображаются на странице.	Пройден
6	Фильтрация по критериям	Действие: Перейти на страницу с каталогом животных. Указать критерии поиска. Нажать кнопку поиска.	Животные с указанными критериями отображаются на странице.	Пройден

7	Поиск заболеваний	Действие: Перейти на страницу болезней. Ввести в поисковую строку название.	Болезни, название которых содержит это слово отображаются на странице.	Пройден
8	Поиск животных	Действие: Перейти на страницу каталога животных. Выбрать тип животных. Нажать поиск.	Животные с указанным типом отображаются на странице.	Пройден
9	Просмотр критериев	Действие: Перейти на страницу каталога животных. Нажать на кнопку критерии.	Критерии видны в выпадающем списке	Пройден
10	Просмотр критериев	Действие: Перейти на страницу каталога животных. Нажать на кнопку критерии.	Критерии видны в выпадающем списке	Пройден
11	Просмотр страницы животного	Действие: Перейти на страницу каталога животных. Нажать на животное в списке.	Страница с подробной информацией о животном.	Пройден
12	Создание запроса на получение животного	Действие: Войти в аккаунт клиента, перейти в каталог животных, нажать на животное, заполнить поле для текста, нажать на кнопку создать запрос.	В профиле появится запрос.	Пройден

13	Добавление животного в приют	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел животные, заполнить поля корректными данными (animal type id, Name, Age, gender, description). Нажать кнопку создать.	На экране появится оповещение об успешном создании животного.	Пройден
14	Просмотр списка животных приюта	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел животные, нажать поиск.	Список животных текущего приюта.	Пройден
15	Обновление информации о животном	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел животные, заполнить поля корректными данными, нажать обновить.	Обновление данных о животном.	Пройден
16	Удаление животного из приюта	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел животные, заполнить поля корректными данными, нажать удалить.	Обновление данных о животном.	Пройден

17	Добавление заболевания животному	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел заболевания животных, заполнить поля корректными данными, нажать создать.	Появление записи в бд	Пройден
18	Удаление заболевания животного	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел заболевания животных, заполнить поля корректными данными, нажать удалить.	Появление записи в бд	Пройден
19	Добавление критерия животному		Появление записи в бд	Пройден
20	Удаление критерия у животного		Удаление записи из бд	Пройден
21	Просмотр запросов на получение животного из приюта	Перейти в профиль приюта, в раздел запросы	Список запросов на странице	Пройден
22	Создание ответа на запрос пользователя о получении животного	Действие: Войти в аккаунт управляющего приютом. Перейти в профиль, открыть раздел запросов, заполнить поля корректными	Изменение запроса	Пройден

		данными, нажать обновить заявку.		
23	Блокировка клиента	Войти в аккаунт администратора. Войти в профиль, раздел admin, заполнить поля валидными данными, нажать подтвердить.	Оповещение об успешном выполнении.	Пройден
24	Блокировка управляющего приютом	Войти в аккаунт администратора. Войти в профиль, раздел admin, заполнить поля валидными данными, нажать подтвердить.	Оповещение об успешном выполнении.	Пройден
25	Добавление типа животного	Войти в аккаунт администратора. Войти в профиль, раздел Animal Types, заполнить поля валидными данными, нажать создать.	В каталоге типов животных появится новый тип.	Пройден
26	Удаление типа животного	Войти в аккаунт администратора. Войти в профиль, раздел Animal Types, заполнить поля валидными данными, нажать удалить.	В каталоге типов животных пропадёт тип.	Пройден
27	Создание критерия	Войти в аккаунт администратора. Войти в профиль, раздел Criteria, заполнить поля валидными	В списке критериев появится новый критерий.	Пройден

		данными, нажать создать.		
28	Удаление критерия	Войти в аккаунт администратора. Войти в профиль, раздел Criteria, заполнить поля валидными данными, нажать удалить.	В списке критериев пропадёт критерий.	Пройден
29	Добавление заболевания в список заболеваний	Войти в аккаунт администратора. Войти в профиль, раздел Disseases, заполнить поля валидными данными, нажать создать.	В каталоге болезней появится болезнь.	Пройден
30	Удаление заболевания из списка заболеваний	Войти в аккаунт администратора. Войти в профиль, раздел Disseases, заполнить поля валидными данными, нажать удалить.	В каталоге болезней пропадёт болезнью	Пройден

4.2 Выводы по разделу

1. Разработано 30 тестов, охватывающих все функции веб-приложения.
2. Проверена корректность обработки данных при выполнении пользовательских и администраторских операций взаимодействия с базой данных.
3. Все тесты завершены успешно, что подтверждает готовность приложения к эксплуатации.
4. Покрытие кода тестами составило 100%.

5 Руководство пользователя

При запуске веб-приложения неавторизованный пользователь будет на главной странице приложения. Главная страница является информационной точкой входа, где пользователь может получить краткую информацию о проекте.

Пользователь может ознакомиться с описанием проекта, его значимостью и преимуществами использования данного приложения. Для наглядного представления главной страницы веб-приложения, на рисунке 5.1 отображается внешний вид и расположение информационных элементов на странице.

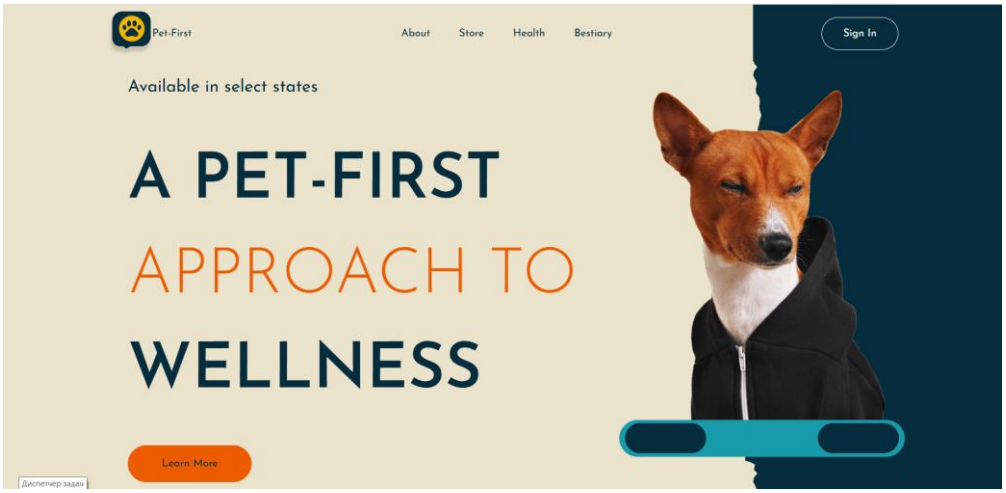

Рисунок 5.1 – Главная страница

5.1 Регистрация

Если гость хочет получить более широкий функционал, он может нажать на кнопку «Sign In» в навигационном меню. Форма авторизации представлена на рисунке 5.2. Для перехода на регистрацию надо нажать кнопку «Sign In». Форма регистрации представлена на рисунке 5.3.

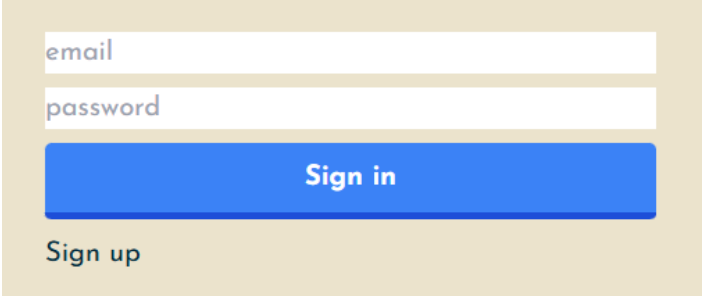

Рисунок 5.2 – Форма авторизации

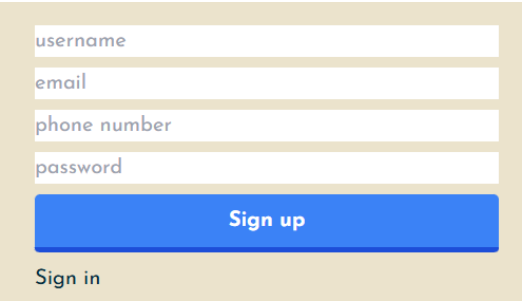

Рисунок 5.3 – Форма регистрации

Здесь необходимо заполнить форму, указав свои данные. После заполнения формы, гость нажимает кнопку «Sign up», которая инициирует процесс проверки введённых данных и после этого, если все успешно, пользователь авторизует в систему под созданным аккаунтов.

5.2 Авторизация

Выполнить вход на сайт можно заполнив данные в форме изображённой на рисунке 5.3, необходимо корректно заполнить данные email и пароль.

5.3 Просмотр каталога животных

Для просмотра каталога животных необходимо выбрать соответствующий раздел в меню или на главной странице. Страница с просмотром каталога животных представлена на рисунке 5.4.

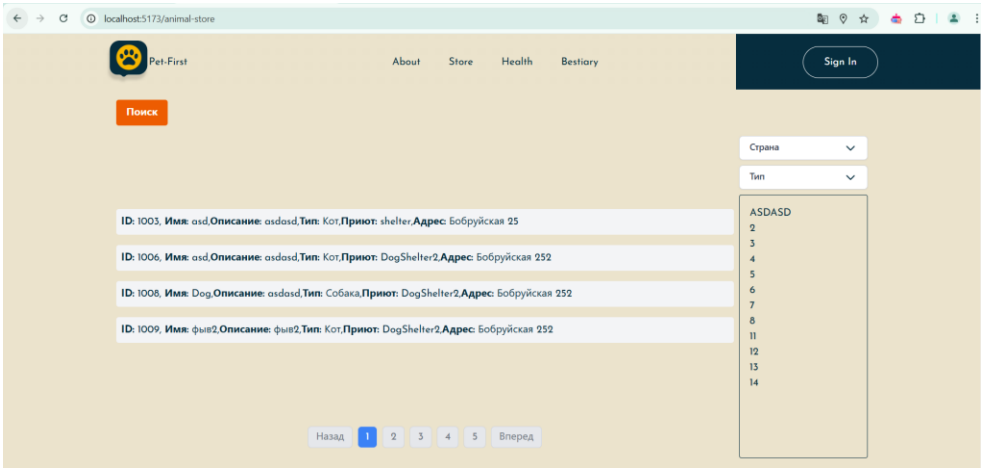

Рисунок 5.4 – Каталог животных

Заключение

В ходе выполнения проекта было реализовано веб-приложение, которое соответствует всем заявленным целям и требованиям.

1. В системе предусмотрено три ключевые роли – пользователь, администратор и гость.
2. Реализовано 30 функций, включая регистрацию, авторизацию, просмотр каталога животных, болезней и типов животных, поиск и фильтрация, дял управляющего приютом управление животными, их критериями и заболеваниями, обработка запросов клиентов, для клиентов создание запросов на получение животного и просмотр истории запросов, для администратора управление списком заболеваний, списком типов животных и блокировкой аккаунтов.
3. В базе данных для хранения было создано 9 таблиц: пользователи, управляющие приютом, животные, типы животных, заболевания, заболевания животных, критерии, критерии животных.
4. Архитектура приложения построена на принципах клиент-серверного взаимодействия. Серверная часть разработана с использованием платформа Node.js, а клиентская – на React.js.
5. Разработаны и проведены 30 успешных функциональных тестов, а покрытие тестами составило 100%

В соответствии с полученным результатом работы можно сделать вывод, что цель достигнута, а требования технического задания полностью выполнены.

Список используемых источников

1. RFC 2616 HTTP/1.1 [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc2616 – Дата доступа: 04.01.2025.
2. Node.js [Электронный ресурс]. – Режим доступа: https://nodejs.org/en/ – Дата доступа: 04.01.2025.
3. «Приют Бирюлево» [Электронный ресурс]. – Режим доступа: https://izpriuta.ru/– Дата доступа: 04.01.2025.
4. «Преданное сердце» [Электронный ресурс]. – Режим доступа: https://predannoeserdce.ru/ – Дата доступа: 04.01.2025.
5. «Искра» [Электронный ресурс]. – Режим доступа: https://iskracats.org/– Дата доступа: 04.01.2025.
6. MSSQL [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/sql/ – Дата доступа: 04.01.2025.
7. Prisma – ORM (Object-Relational Mapping) [Электронный ресурс]. – Режим доступа: https://www.prisma.io/docs/ – Дата доступа: 04.01.2025.
8. nginx [Электронный ресурс]. – Режим доступа: https://nginx.org/ru/ – Дата доступа: 04.01.2025.
9. RFC 9293 TCP [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/info/rfc9293 – Дата доступа: 04.01.2025.
10. Docker Compose [Электронный ресурс]. – Режим доступа: https://docs.docker.com/compose/ – Дата доступа: 04.01.2025.
11. Express.js [Электронный ресурс]. – Режим доступа: https://expressjs.com/ru/ – Дата доступа: 04.01.2025.
12. bcrypt [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/bcrypt – Дата доступа: 04.01.2025.
13. cookie-parser [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/cookie-parser – Дата доступа: 04.01.2025..
14. jsonwebtoken [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/jsonwebtoken – Дата доступа: 04.01.2025.
15. @prisma/client [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@prisma/client – Дата доступа: 04.01.2025.
16. cors [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/cors – Дата доступа: 04.01.2025.
17. nodemon [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/nodemon – Дата доступа: 04.01.2025.
18. swagger-jsdoc [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/swagger-jsdoc – Дата доступа: 04.01.2025.
19. swagger-ui-express [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/swagger-ui-express – Дата доступа: 04.01.2025.

20. Axios [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/axios – Дата доступа: 04.01.2025.
21. Mobx [Электронный ресурс]. – Режим доступа: https://mobx.js.org/ – Дата доступа: 04.01.2025.
22. react-toastify [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/react-toastify – Дата доступа: 04.01.2025.
23. React [Электронный ресурс]. – Режим доступа: https://react.dev/ – Дата доступа: 04.01.2025.
24. Tailwindcss [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/tailwindcss – Дата доступа: 04.01.2025.