МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование в Internet

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Современные технологии разработки веб-приложений»

Тема: Веб-приложение для поиска и получения животных из приютов

Исполнитель

студент 4 курса группы 5 Песецкий Никита Андреевич

(Ф.И.О.)

Руководитель работы ассистент Некрасова А.П.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2024

Оглавление

[Введение 4](#_Toc184931192)

[1 Постановка задачи 5](#_Toc184931193)

[1.1 Аналитический обзор аналогов 6](#_Toc184931194)

[1.1.1 Printrepublic.by 6](#_Toc184931195)

[1.1.2 Vsemayki.ru 7](#_Toc184931196)

[1.2 Разработка функциональных требование и вариантов использования 8](#_Toc184931197)

[1.3 Выводы по разделу 10](#_Toc184931198)

[2 Проектирование web-приложения 11](#_Toc184931199)

[2.1 Описание средств разработки 11](#_Toc184931200)

[2.1.1 Visual Studio Code 11](#_Toc184931201)

[2.1.2 Фреймворк NestJS 11](#_Toc184931202)

[2.1.3 Библиотека React 11](#_Toc184931203)

[2.1.4 PostgreSQL 12](#_Toc184931204)

[2.1 Обобщенная структура управлением приложения 12](#_Toc184931205)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 12](#_Toc184931206)

[2.4 Выводы по разделу 14](#_Toc184931207)

[3 Разработка web-приложения 15](#_Toc184931208)

[3.1 Общая структура 15](#_Toc184931209)

[3.2 Разработка серверной части web-приложения 15](#_Toc184931210)

[3.2.1 Регистрация 18](#_Toc184931211)

[3.2.2 Авторизация 19](#_Toc184931212)

[3.2.3 Поиск вакансий 21](#_Toc184931213)

[3.2.4 Добавление вакансий 22](#_Toc184931214)

[3.2.5 Чат между работодателем и соискателем 23](#_Toc184931215)

[3.3 Разработка клиентской части web-приложения 25](#_Toc184931216)

[3.4 Выводы по разделу 28](#_Toc184931217)

[4 Тестирование web-приложения 29](#_Toc184931218)

[4.1 Тестирование авторизации и регистрации 29](#_Toc184931219)

[4.2 Тестирование добавление вакансии 31](#_Toc184931220)

[4.3 Тестирование поиска вакансий 32](#_Toc184931221)

[4.4 Тестирование заполнения профиля 34](#_Toc184931222)

[4.4 Выводы по разделу 35](#_Toc184931223)

[5 Руководство пользователя 36](#_Toc184931224)

[5.1 Руководство пользователя 36](#_Toc184931225)

[5.1.1 Соискатель 36](#_Toc184931226)

[5.1.2 Работодатель 42](#_Toc184931227)

[5.1.3 Менеджер 45](#_Toc184931228)

[5.2 Установка приложения 47](#_Toc184931229)

[5.3 Выводы по разделу 47](#_Toc184931230)

[Заключение 48](#_Toc184931231)

[Список используемых источников 49](#_Toc184931232)

[Приложение А 50](#_Toc184931233)

[Приложение Б 51](#_Toc184931234)

# Введение

В современном мире цифровизация охватывает все больше аспектов нашей жизни, и сфера благополучия животных не является исключением. Люди, заинтересованные в приобретении питомцев, стремятся не только найти подходящего животного, но и взаимодействовать с приютами напрямую. Это может включать как просмотр доступных животных, так и оформление заявок на получение животного, а также общение с сотрудниками приютов для получения дополнительной информации. Однако многие сталкиваются с проблемой поиска надежных приютов, отслеживания статусов своих заявок и эффективного общения на существующих платформах. В условиях растущей осведомленности о правах животных и увеличении числа питомцев в приютах возникает необходимость в создании удобных инструментов для взаимодействия между потенциальными клиентом и приютами.

Актуальность данной темы обусловлена возрастающей ролью онлайн-платформ для поиска и получения животных. Такие платформы позволяют пользователям легко находить приюты, просматривать доступных животных, оформлять заявки на получение и делиться опытом. Для приютов это удобная площадка для публикации информации о животных, получения обратной связи и управления процессом выдачи животных. В то же время администраторы могут поддерживать порядок на платформе, управляя категориями и контролируя контент.

Существующие решения в данной сфере зачастую узко специализированы и не предлагают полноценного функционала для взаимодействия пользователей, приютов и администраторов. Отсутствие платформы, объединяющей все необходимые функции, ограничивает возможности как потенциальных клиентов, так и приютов. Таким образом, существует потребность в разработке web-приложения, которое обеспечивало бы полный набор функций для взаимодействия всех участников платформы и упрощало бы процессы поиска, оформления и отслеживания получения, а также коммуникации между пользователями и приютами.

Целью данного проекта является создание web-приложения, которое позволит пользователям находить животных в приютах, оформлять заявки на получение животного, отслеживать статус своих заявок. Дополнительно, проект предусматривает инструменты для управления контентом и ролями пользователей, которые будут доступны администраторам приютов.

Таким образом, созданный web-сервис не только упростит процесс поиска и получения животных из приютов, но и создаст полноценную платформу для поддержки благополучия животных, улучшения коммуникации между приютами и потенциальными клиентами, а также повышения осведомленности о проблемах бездомных животных.

# 1 Постановка задачи

Анализ существующих решений на рынке является важным этапом разработки любого веб-приложения. Он позволяет выявить потребности пользователей, определить их ожидания и выделить направления для внедрения уникальных функций.

Веб-приложение для поиска и получения животных из приютов, созданное на основе современных технологий, направлено на упрощение взаимодействия между потенциальными клиентами и приютами, предоставляющими животных для клиентов. Это приложение должно стать удобным и универсальным инструментом, который обеспечит эффективное взаимодействие между пользователями и приютами, а также упростит процесс поиска, оформления и отслеживания клиентов.

Должны быть выполнены следующие требования:

* регистрация, авторизация как пользователя или управляющего приютом;
* просмотр и навигация по списку животных
* поиск и фильтрация по критериям;
* просмотр подробной информации о каждом животном;
* добавление новых животных в приют управляющим;
* редактирование информации о животных в приюте управляющим;
* обработка запросов на получение животных управляющим;
* добавление заболеваний в список заболеваний администратором;
* Блокирование пользователей администратором.

Проект включает проектирование архитектуры приложения, разработку интерфейсов и серверной части, создание базы данных, тестирование и развертывание готового продукта.

## 1.1 Аналитический обзор аналогов

На текущий момент на рынке представлено несколько платформ, направленных на содействие получателям животных, однако большинство из них имеют ограниченный функционал или ориентированы на конкретные регионы. Основные существующие решения включают такие платформы, как Petfinder, Adoptapet и другие региональные сайты. Эти сервисы предоставляют пользователям возможность просматривать доступных животных, фильтровать их по различным критериям и связываться с приютами для оформления получения. Тем не менее, у них есть ряд недостатков, которые ограничивают их эффективность и удобство использования.

### 1.1.1 Petfinder

Petfinder [1] — Одна из крупнейших платформ для поиска и получения животных в Северной Америке. Предлагает широкий выбор животных, возможность фильтрации по типу, породе, возрасту и другим параметрам. Однако, взаимодействие с приютами ограничено формами связи через сайт, что затрудняет непосредственное общение и обмен информацией.

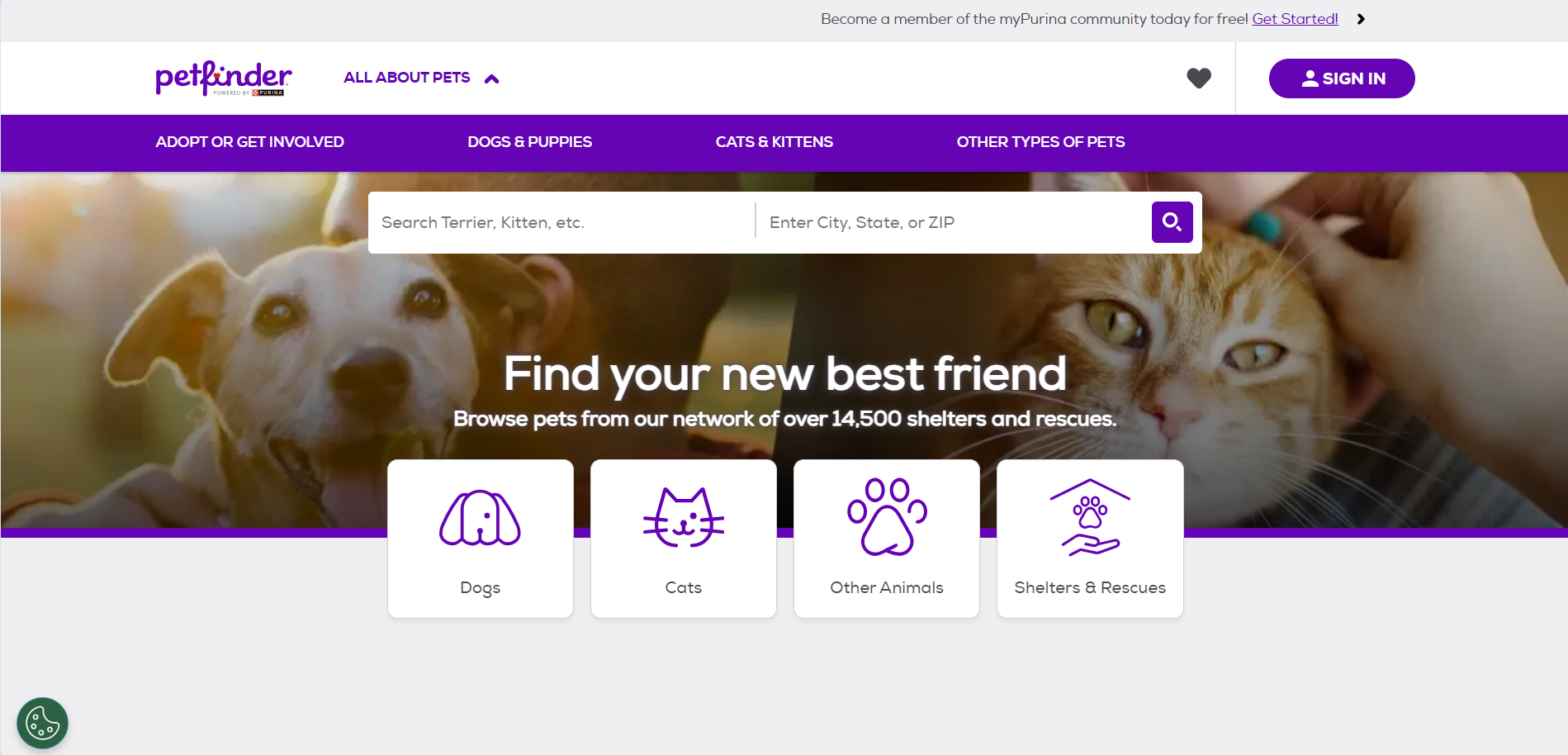


Рисунок 1.1 – Интернет-ресурс Petfinder

Взаимодействие с приютами на Petfinder осуществляется через предоставление контактной информации приютов, что позволяет пользователям напрямую связываться с ними для уточнения деталей получения, организации встреч и получения дополнительной информации о животных. Платформа сотрудничает с тысячами приютов, спасательных организаций и волонтерских групп, обеспечивая широкий выбор животных для получения животного. Пользователи могут создавать личные кабинеты, где сохраняются их заявки на получение животного, история взаимодействий с приютами и избранные животные. Возможность добавления животных в избранное облегчает доступ к ним в будущем, а уведомления о новых поступлениях и статусах заявок помогают пользователям быть в курсе всех обновлений.

Однако, несмотря на множество преимуществ, у Petfinder существуют и некоторые недостатки. Одним из основных недостатков является ограниченная возможность коммуникации, так как взаимодействие с приютами осуществляется через электронную почту или телефон, что может замедлять процесс обмена информацией и усложнять общение. Процесс подачи заявок также может быть неинтуитивным и требовать заполнения большого количества форм, что иногда отпугивает пользователей. Кроме того, хотя Petfinder охватывает большую часть Северной Америки, пользователи из других регионов могут столкнуться с ограниченным доступом к информации о местных приютах, что снижает универсальность платформы.

С технической точки зрения Petfinder использует современные веб-технологии для обеспечения высокой производительности и надежности платформы. Веб-сайт оптимизирован для работы как на настольных компьютерах, так и на мобильных устройствах, что делает его доступным для широкого круга пользователей. Платформа также соблюдает стандарты безопасности для защиты личной информации пользователей и данных приютов. Наличие API позволяет приютам и сторонним разработчикам интегрировать данные о животных в свои собственные приложения и веб-сайты, расширяя функциональные возможности Petfinder.

### 1.1.2 Adoptapet

Adoptapet [2] — одна из ведущих онлайн-платформ для поиска и усыновления животных, функционирующей в различных регионах США и других странах. С момента своего основания Adoptapet зарекомендовал себя как надежный ресурс для тех, кто стремится найти нового питомца или помочь животным из приютов обрести дом. Платформа предлагает обширную базу данных животных, доступных для усыновления, что позволяет пользователям легко находить подходящих питомцев по различным критериям, таким как вид, порода, возраст, размер, пол, цвет и местоположение.

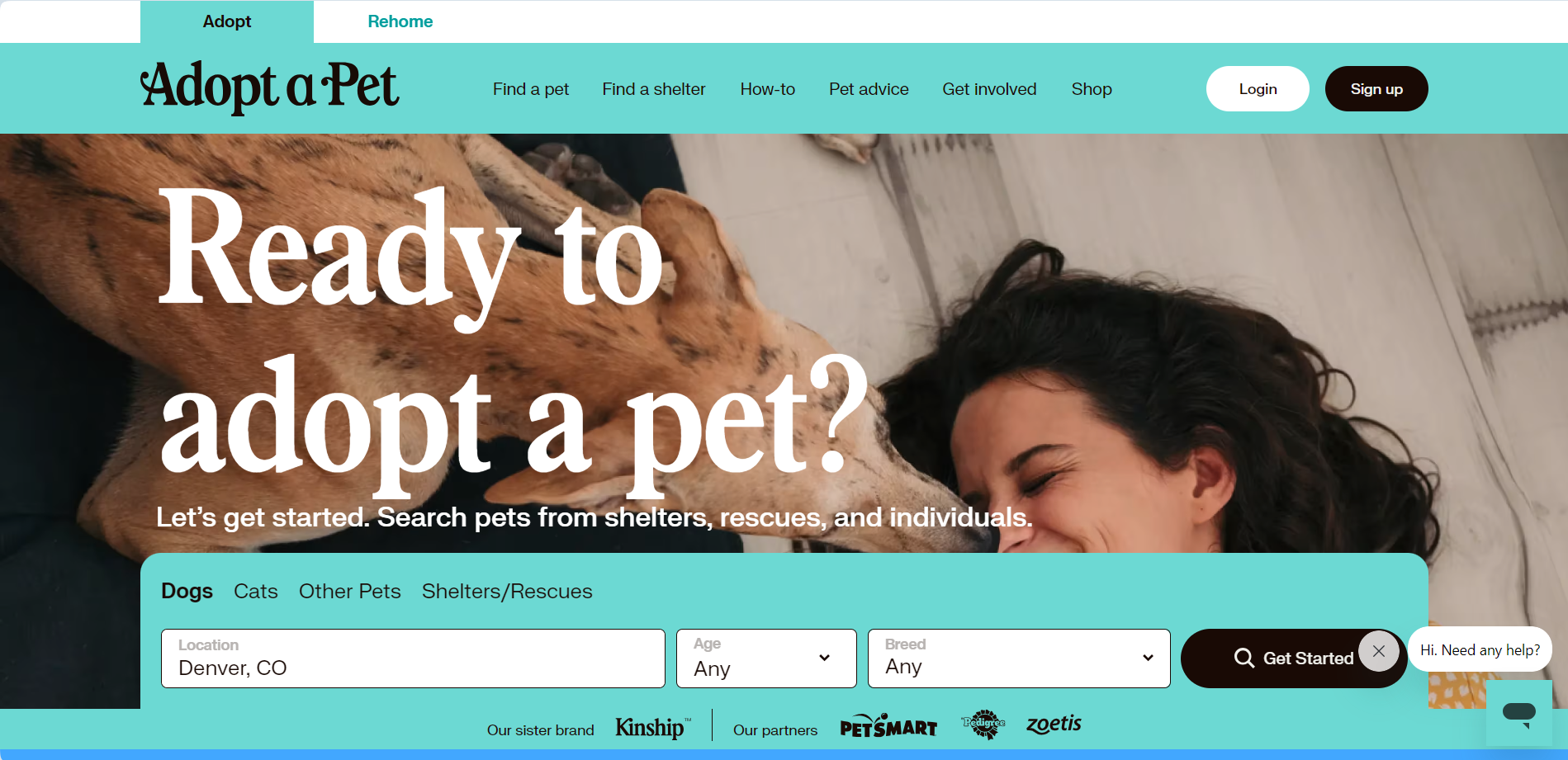


Рисунок 1.2 – Интернет-ресурс Adoptapet.com

Платформа предоставляет обширную базу данных животных, доступных для усыновления, что позволяет пользователям легко искать подходящих питомцев по различным критериям, таким как вид животного (собаки, кошки, птицы и другие), порода, возраст, размер, пол, цвет и местоположение. Каждый профиль животного содержит детальную информацию, включая фотографии, описание характера, потребностей, медицинскую историю и рекомендации по уходу, что помогает потенциальным усыновителям сделать осознанный выбор.

Adoptapet предлагает удобный личный кабинет для пользователей, где они могут сохранять избранных животных, отслеживать статус своих заявок на усыновление и получать уведомления о новых поступлениях и изменениях в статусе существующих заявок. Это помогает пользователям быть в курсе всех обновлений и эффективно управлять процессом усыновления. Кроме того, платформа предоставляет различные ресурсы и статьи, посвященные подготовке к усыновлению, уходу за животными, обучению и решению поведенческих проблем, что способствует повышению осведомленности и ответственности среди потенциальных владельцев питомцев.

С технической точки зрения Adoptapet использует современные веб-технологии для обеспечения высокой производительности и надежности платформы. Веб-сайт оптимизирован для работы как на настольных компьютерах, так и на мобильных устройствах, что делает его доступным для широкого круга пользователей. Платформа также соблюдает стандарты безопасности для защиты личной информации пользователей и данных приютов. Наличие API позволяет приютам и сторонним разработчикам интегрировать данные о животных в свои собственные приложения и веб-сайты, расширяя функциональные возможности Adoptapet.

## 1.2 Разработка функциональных требований и вариантов использования

Функционально веб-приложение для усыновления животных должно поддерживать четыре роли: гость, пользователь, управляющий приютом и администратор. Гости могут зарегистрироваться или авторизоваться, выбирая роль пользователя или управляющего приютом.

Пользователи получают доступ к просмотру списка животных и могут легко ориентироваться в каталоге. Для удобства предусмотрены функции поиска и фильтрации по различным критериям, таким как возраст, размер или порода. Также пользователи могут изучать детальную информацию о каждом животном, включая его фотографии и историю, и оставлять заявки на усыновление.

Управляющие приютом имеют возможность добавлять новых животных в базу данных, указывая ключевую информацию, такую как возраст, здоровье и особенности. Они также могут редактировать данные о животных и обрабатывать заявки пользователей на их усыновление.

Администраторы отвечают за управление платформой на более высоком уровне. Они могут добавлять в систему новые заболевания для учета здоровья животных, а также блокировать пользователей или приюты в случае необходимости.

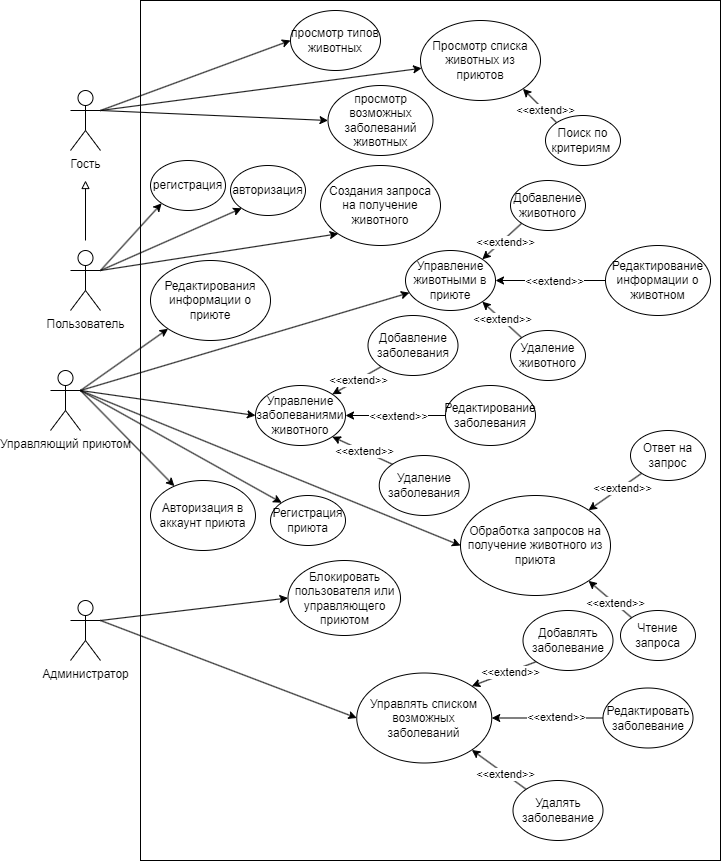


Рисунок 1.4 – Диаграмма вариантов использования

Диаграмма UML вариантов использования позволяет понять, что доступно каждой роли в данном веб-приложении.

## 1.3 Выводы по разделу

В данном разделе был проведен обзор аналогичных веб-приложений, таких как Adoptapet и Petfinder, что позволило выявить ключевые функции для нашего приложения. Основной функционал включает регистрацию и авторизацию для различных ролей, таких как пользователь, управляющий приютом и администратор.

Также обсуждены требования к удобному интерфейсу и функционалу, обеспечивающему эффективную работу всех ролей, что проиллюстрировано разработанной UML-диаграммой.

# 2 Проектирование web-приложения

# 2.1 Описание средств разработки

При разработке приложения были использованы:

* интегрированная среда разработки Visual Studio Code;
* фреймворк ExpressJS;
* библиотека React;
* база данных MSSQL;

### 2.1.1 Visual Studio Code

Visual Studio Code — это современная и производительная интегрированная среда разработки, которая поддерживает широкий спектр языков программирования, таких как JavaScript, TypeScript, Python, C++ и многие другие. Она также совместима с популярными библиотеками и фреймворками, включая React и ExpressJS. VS Code оснащена мощными инструментами для написания, отладки и тестирования кода, а её функциональность можно значительно расширить благодаря множеству доступных плагинов и дополнений. Среда обеспечивает удобную работу с различными платформами и базами данных, сохраняя лёгкость и гибкость в использовании.

### 2.1.2 Фреймворк ExpressJS

**ExpressJS** — это минималистичный и гибкий фреймворк для разработки веб-приложений на платформе Node.js. Он предоставляет мощные инструменты для создания серверной части приложений, включая маршрутизацию, обработку запросов и ответов, управление middleware и интеграцию с базами данных. Благодаря лаконичности и модульности, ExpressJS позволяет разработчикам быстро создавать API и веб-сервисы, сохраняя при этом контроль над архитектурой приложения. Он поддерживает динамическое создание страниц, обработку форм, а также работу с файлами и сессиями. ExpressJS является отличным выбором для построения как небольших приложений, так и масштабируемых систем благодаря своей производительности и совместимости с большим количеством библиотек Node.js.

### 2.1.3 Библиотека React

React — это популярная JavaScript-библиотека, предназначенная для создания пользовательских интерфейсов, с особым акцентом на одностраничные приложения (SPA). Разработанная компанией Facebook, React позволяет создавать интерфейсы, используя компоненты, которые можно многократно переиспользовать, что делает разработку более эффективной и масштабируемой. Основной особенностью React является использование виртуального DOM, который обеспечивает высокую производительность приложения, минимизируя количество прямых изменений в реальном DOM. Это ускоряет процесс рендеринга и улучшает взаимодействие пользователя с приложением, обеспечивая плавный и отзывчивый интерфейс.

### 2.1.4 MSSQL

**Microsoft SQL Server (MSSQL)** — это реляционная система управления базами данных (СУБД), разработанная корпорацией Microsoft. Она предоставляет мощный инструментарий для хранения, обработки и анализа данных, поддерживая язык запросов SQL (Structured Query Language). MSSQL широко используется для создания надежных и масштабируемых решений благодаря встроенным функциям, таким как управление транзакциями, репликация данных, поддержка хранимых процедур и триггеров. Система также предлагает расширенные возможности для аналитики, включая интеграцию с инструментами бизнес-анализа (BI) и поддержку обработки больших данных через Hadoop. MSSQL отличается высокой производительностью, безопасностью и интеграцией с экосистемой Microsoft, что делает её популярным выбором как для локальных серверов, так и для облачных решений, таких как Microsoft Azure.

## 2.1 Обобщенная структура управлением приложения

Для эффективного управления веб-приложением, предназначенным для заказа усыновления животных, необходимо создать интуитивно понятный и функциональный интерфейс, который упростит взаимодействие пользователей с платформой. Это включает разработку оптимизированных запросов для добавления, редактирования и удаления карточек животных, профилей пользователей и данных о доступных животных. Ключевой функционал должен включать возможность поиска животных с использованием различных фильтров. Такой подход обеспечит удобство использования платформы и повысит эффективность взаимодействия всех сторон.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

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

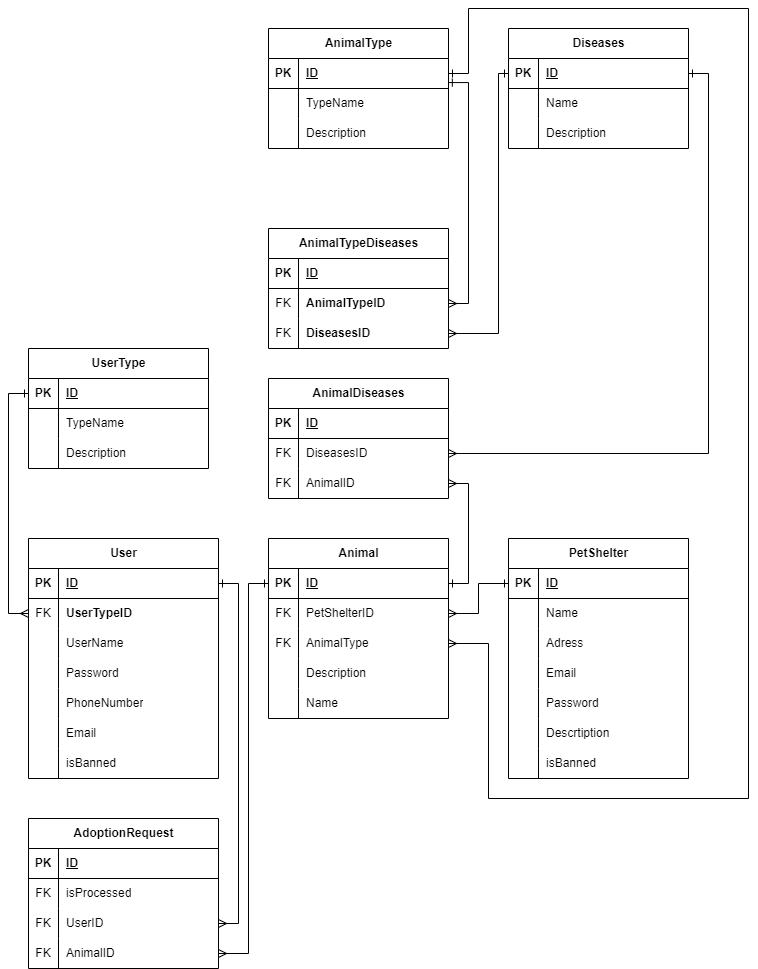


Рисунок 2.1 – Логическая схема базы данных

Таблица User используется для хранения информации о пользователях, таких как имя пользователя, пароль, адрес электронной почты и номер телефона. Также предусмотрены поля для указания типа пользователя (например, гость, управляющий приютом или администратор) и статуса блокировки. Первичный ключ обеспечивает уникальность записей, а связь с таблицей UserType позволяет управлять ролями пользователей.

Таблица UserType содержит информацию о доступных типах пользователей, включая их названия и описания. Эта таблица связана с таблицей User и обеспечивает возможность гибкого управления типами учетных записей.

Таблица Animal используется для хранения информации о животных, которые доступны для усыновления. Она содержит такие данные, как имя, описание, тип животного и связь с приютом. Также предусмотрены внешние ключи для связи с таблицами AnimalType и PetShelter, что позволяет классифицировать животных и указывать их местонахождение.

Таблица AnimalType хранит данные о типах животных, включая название и описание. Она связана с таблицами Animal и AnimalTypeDiseases, что позволяет учитывать специфику каждого типа животного.

Таблица Diseases предназначена для хранения информации о заболеваниях животных. Она содержит уникальный идентификатор, название и описание болезни. Эта таблица связана с Animal через AnimalDiseases и с AnimalType через AnimalTypeDiseases, что позволяет учитывать заболевания конкретных животных и типов.

Таблица AnimalDiseases связывает животных с их заболеваниями. Она содержит внешний ключ для каждого животного и соответствующего заболевания, что позволяет учитывать состояние здоровья каждого животного.

Таблица AnimalTypeDiseases связывает типы животных с характерными для них заболеваниями. Это позволяет эффективно управлять данными о предрасположенности определённых типов животных к конкретным болезням.

Таблица PetShelter хранит информацию о приютах, включая название, адрес, электронную почту и описание. Также предусмотрены поля для пароля и статуса блокировки. Эта таблица связана с Animal, что позволяет отслеживать, какие животные находятся в конкретном приюте.

Таблица AdoptionRequest используется для хранения заявок на усыновление животных. Она содержит информацию о пользователе, животном и статусе обработки заявки. Внешние ключи связывают её с таблицами User и Animal, что позволяет организовать процесс усыновления.

Эти таблицы формируют целостную структуру данных для управления пользователями, животными, приютами и заявками, обеспечивая функциональность системы усыновления.

## 2.4 Выводы по разделу

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой. В данном разделе были рассмотрены основные таблицы базы данных MSSQL. Определены их ключевые поля и связи между таблицами, что позволяет эффективно управлять данными и поддерживать функциональность приложения.

# 3 Разработка web-приложения

## 3.1 Общая структура

Общая структура данного проекта состоит из двух основных частей: клиентской и серверной. Серверная часть отвечает за обработку логики приложения, взаимодействие с базой данных, а также за управление запросами от клиента и их соответствующую обработку Клиентская часть, в свою очередь, отвечает за пользовательский интерфейс, взаимодействие с сервером и отображение данных.

## 3.2 Разработка серверной части web-приложения

Серверная часть приложения разработана с использованием фреймворка NestJS для платформы Node.js. Для реализации различных функций были использованы такие пакеты, как websockets для создания соединений в реальном времени, nodemailer для отправки электронных писем, passport-google-oauth20 для интеграции с аутентификацией через Google, argon2 для безопасного хеширования паролей, а также prisma для работы с базой данных PostgreSQL. Prisma упрощает взаимодействие с базой данных, предоставляя мощный ORM-инструмент для выполнения запросов и управления данными

Структура разработки приложения на NestJS основывается на трех основных компонентах: Controller, Service и Module. На рисунке 3.1 будет представлена схема структуры данного проекта.

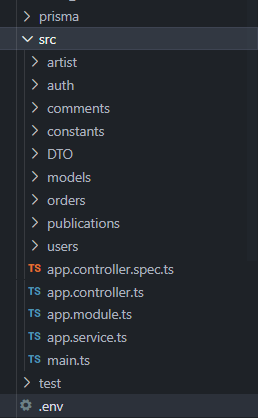


Рисунок 3.1 – Структура проекта

Controller отвечает за обработку входящих HTTP-запросов. В нем описываются маршруты (endpoints), которые слушают определенные пути, и обрабатывают действия, такие как создание, получение, обновление или удаление данных. Контроллеры также инициируют вызовы бизнес-логики, которая реализована в Service. Это позволяет отделить логику маршрутизации от более сложной бизнес-логики.

Service — это слой, где реализована основная бизнес-логика приложения. В сервисах описываются методы для работы с данными, которые могут включать взаимодействие с базой данных, внешними API или другими сервисами. Вся сложная обработка данных и вычисления выполняются здесь, обеспечивая чистоту и поддержку контроллеров.

Module служит для организации приложения и связки контроллеров, сервисов и других компонентов. Каждый модуль инкапсулирует в себе все необходимые части для выполнения конкретной функции, включая контроллеры и сервисы, а также определения зависимостей между компонентами. Модуль позволяет эффективно управлять зависимостями и упрощает масштабируемость приложения.

Главный модуль приложения — AppModule. Он является основным и подключает все остальные модули, отвечающие за различные части функционала приложения. В AppModule объединяются контроллеры и сервисы, которые реализуют бизнес-логику и взаимодействие с внешними сервисами. Модули для работы с базой данных, авторизацией и другими функциями подключаются в AppModule, что упрощает управление зависимостями. Код AppModule будет представлен на рисунке 3.1, иллюстрирующем структуру приложения.

@Module({

@Module({

imports: [AuthModule, UsersModule, PublicationsModule],

controllers: [AppController],

providers: [AppService],

})

export class AppModule {}

Листинг 3.1 – Компонет AppModule

В NestJS каждый модуль состоит из нескольких частей. В разделе imports указываются модули, которые необходимы для работы текущего модуля, будь то внутренние или сторонние. Это позволяет подключать зависимости и взаимодействовать между модулями. В exports указываются компоненты, такие как сервисы, которые должны быть доступны для других модулей. Раздел controllers отвечает за обработку входящих запросов от клиента и взаимодействие с сервисами для выполнения бизнес-логики. В providers размещаются сервисы, репозитории и другие компоненты, которые реализуют бизнес-логику, взаимодействуют с базой данных или внешними сервисами, и предоставляются другим частям приложения через инъекцию зависимостей.

Точка входа в приложение NestJS — это файл main.ts. В этом файле происходит инициализация приложения, настройка всех необходимых параметров и запуск сервера. Вначале создается экземпляр приложения, где указывается главный модуль, отвечающий за структуру всего проекта. Далее настраиваются глобальные миддлвары, такие как валидация запросов, обработка cookies, сжатие данных и настройка CORS для разрешения междоменных запросов. Также в файле задаются префиксы для API-эндпоинтов и путь для статики. После этого сервер запускается на указанном порту, который берется из конфигурации. Этот файл играет ключевую роль в запуске приложения и установке всех его параметров. В листинге 3.2 будет представлен данный файл.

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.use(cookieParser());

app.enableCors({

origin: 'http://localhost:5173', // Замените на ваш фронтенд URL

methods: ['GET', 'POST', 'PUT', 'DELETE'],

allowedHeaders: ['Content-Type', 'Authorization'],

credentials: true, // Добавьте эту строку для включения credentials

});

await app.listen(process.env.PORT ?? 3000);

}

bootstrap();

Листинг 3.2 – Код bootstrap

В данном файле указан префикс api, который добавляется ко всем API-запросам, что упрощает организацию маршрутов. Используется middleware compression для сжатия данных, улучшая производительность и уменьшая нагрузку на сеть. Конфигурация приложения загружается через ConfigService, который читает переменные из .env файла, что позволяет гибко настраивать параметры, такие как порт и URL фронтенда.

В рамках данного проекта будут рассматриваться следующие основные функции:

* регистрация;
* авторизация;
* поиск публикаций;
* добавление публикаций;
* отправление заказа художнику.

### 3.2.1 Регистрация

В любом приложении одним из самых важных аспектов является правильная и безопасная регистрация и авторизация пользователей. Эти процессы должны быть реализованы таким образом, чтобы обеспечить надежную защиту данных пользователей, удобство их использования и интеграцию с другими частями системы. В этом разделе рассмотрим процесс регистрации пользователей.

Для регистрации в данном приложении используется AuthController, который обрабатывает endpoint register. В запросе на регистрацию с фронтенда передаются такие данные, как email, password и роль пользователя. Роль пользователя задается на клиентской стороне.

Код для логики регистрации будет представлен в листинге 3.3.

@Post('register')

async register(

@Body() body: { username: string; email: string; password: string },

@Res() response: Response,

) {

const user = await this.usersService.createUser(

body.username,

body.email,

body.password,

);

if (user) {

const { access\_token, refresh\_token } = await this.authService.login(

body.email,

body.password,

);

response.cookie('refresh\_token', refresh\_token, {

httpOnly: true,

secure: true,

sameSite: 'strict',

});

return response.send({ user, access\_token });

}

return response

.send({ error: 'Неудачная попытка регистрации' })

.status(404);

}

@HttpCode(HttpStatus.OK) @Post('login') async login(

@Body() body: { email: string; password: string },

@Res() response: Response,

) {

const { user, access\_token, refresh\_token } = await this.authService.login(

body.email,

body.password,

);

response.cookie('refresh\_token', refresh\_token, {

httpOnly: true,

secure: true,

sameSite: 'strict',

});

return response.send({ user, access\_token });

}

Листинг 3.3 – Код регистрации

Основной задачей на сервере является проверка данных, хеширование пароля с помощью алгоритма Argon2 для безопасности, а затем сохранение данных в базу данных. Данный процесс может быть реализован следующим образом:

Сначала проверяется, существует ли пользователь с указанным email. Если такой пользователь уже зарегистрирован, возвращается ошибка. Далее, после того как проверка email проходит успешно, пароль пользователя хешируется с помощью алгоритма Argon2, что гарантирует его безопасность. Затем, в зависимости от роли пользователя, которая задается на фронтенде, создается и сохраняется объект пользователя в базе данных.

После того как пользователь успешно зарегистрирован, генерируются два токена. Access токен предназначен для аутентификации запросов к API. Этот токен имеет ограниченный срок действия (обычно один час), и он отправляется в response клиенту, где хранится в localStorage. Он прикрепляется к каждому запросу для подтверждения подлинности пользователя. Refresh токен, в свою очередь, хранится в cookie и действует в течение 30 дней. Его задача — обеспечивать возможность обновления access токена после его истечения, без необходимости повторной авторизации пользователя. Это повышает безопасность, поскольку refresh токен защищен от атак XSS за счет использования флага HttpOnly, который не позволяет JavaScript доступать его содержимое.

Таким образом, два токена обеспечивают баланс между удобством и безопасностью, гарантируя, что сессия пользователя будет безопасно продолжаться в течение долгого времени, при этом минимизируя риски утечек данных.

### 3.2.2 Авторизация

Авторизация — это важный процесс, который позволяет пользователям получить доступ к защищенным ресурсам системы после того, как они прошли проверку.

Код для логики авторизации будет представлен в листинге 3.4.

@HttpCode(HttpStatus.OK) @Post('login') async login(

@Body() body: { email: string; password: string },

@Res() response: Response,

) {

const { user, access\_token, refresh\_token } = await this.authService.login(

body.email,

body.password,

);

response.cookie('refresh\_token', refresh\_token, {

httpOnly: true,

secure: true,

sameSite: 'strict',

});

return response.send({ user, access\_token });

}

Листинг 3.4 – Код авторизации

В данном приложении процесс авторизации состоит из нескольких этапов. Во-первых, пользователь передает два поля: email и password. На сервере происходит проверка существования пользователя с указанным email, затем пароль проверяется на соответствие хешу, сохраненному в базе данных. Если пользователь с таким email существует и пароль правильный, генерируются два токена: access токен и refresh токен. Access токен используется для выполнения запросов к серверу и имеет ограниченный срок действия (обычно один час), в то время как refresh токен имеет более длительный срок жизни (30 дней) и используется для получения нового access токена, когда старый истекает.

Если пользователь с таким email не найден или пароль неверный, система генерирует ошибку. В случае успешной авторизации, оба токена отправляются клиенту. Access токен помещается в localStorage клиента для использования в последующих запросах, а refresh токен сохраняется в cookie для безопасного обновления токенов.

### 3.2.3 Поиск публикаций

Поиск публикаций является одной из основополагающих функций для пользователей, ищущих работу. Хорошо настроенный механизм поиска позволяет пользователям быстро находить подходящие вакансии, что существенно улучшает опыт использования приложения. Важно, чтобы процесс поиска был быстрым, точным и удобным, с возможностью фильтрации по различным параметрам, таким как должность, местоположение, категория и другие.

Основной запрос для поиска вакансий будет обрабатывать параметры, передаваемые пользователем, и фильтровать вакансии в базе данных по этим параметрам. Запрос может включать фильтрацию по ключевым словам в названии вакансии, категории, местоположению и другим параметрам, которые могут быть полезны для соискателей.

В листинге 3.5 будет представлен основной запрос для поиска публикаций.

static async searchPublications(

page: number,

filter: any

): Promise<AxiosResponse<any>> {

return $api

.get<any>(

`publications/search?page=${page}

${filter.artist\_id ? `&artist\_id=${filter.artist\_id}` : ""}

${filter.title ? `&title=${filter.title}` : ""}

`

)

.then((response) => response);

}

Листинг 3.5 – Код для поиска вакансии

От клиента в запросе будут приходить основные фильтры, такие как workType, salaryFrom, salaryTo, categories, experience, employmentType, title, dateFrom, dateTo, sort, education, showSalary, regionId, и cityId. Если эти параметры присутствуют в запросе, они будут учитываться при поиске вакансий.

Затем формируем объект filters, который включает в себя все эти параметры. Далее, этот объект добавляется к запросу, чтобы получить вакансии, соответствующие переданным фильтрам.

Если поиск осуществляется по параметру title, то поиск будет осуществляться не только по названию вакансии, но и по названию компании, а также по ключевым навыкам, что позволяет более точно найти подходящие вакансии. Для других фильтров, например, workType, salaryFrom, salaryTo и других, будут отбираться только вакансии, которые соответствуют этим критериям. Также можно указать regionId и cityId для более точного поиска вакансий в нужном географическом регионе.

Кроме того, фильтры salaryFrom и salaryTo позволят отфильтровать вакансии по диапазону заработной платы, а dateFrom и dateTo — по дате создания вакансий.

### 3.2.4 Добавление публикаций

Добавление вакансий в приложении доступно только работодателям, так как именно они отвечают за создание предложений о трудоустройстве. Для обеспечения безопасности и правильной работы системы используется механизм авторизации с проверкой роли пользователя. Для того чтобы гарантировать, что только пользователь с ролью company может добавлять вакансии, в NestJS применяется Guard.

Guard — это механизм, который проверяет, имеет ли пользователь права для выполнения определенного действия. В случае с добавлением вакансий, Guard проверяет, что запрос поступил от пользователя с ролью company. Если роль пользователя не соответствует нужной, Guard блокирует выполнение запроса и возвращает ошибку.

Пример работы Guard заключается в том, что когда пользователь пытается создать вакансию, его запрос проходит через Guard, который проверяет его роль. Если роль пользователя — company, запрос разрешается, и создается вакансия. Если роль отличается, запрос отклоняется.

Реализация Guard будет показана в листинге 3.6.

async canActivate(context: ExecutionContext) {

const user = await this.userService.findUserByEmail(email);const company = await this.companyService.getCompanyByUserId(user.id);

if (!company) {

return false;

} if (user.role.name !== Role.COMPANY) {

return false;

} request.company = company;

request.user = user; return true;}

Листинг 3.6 – Guard для компании

Для добавления вакансии предусмотрены обязательные поля, такие как заголовок вакансии, описание, заработная плата (от и до), а также необязательные поля, включая город. Эти обязательные поля нужны для того, чтобы вакансия была полноценной и содержала все важные требования, что позволит работодателям корректно представить свои предложения, а соискателям — понимать, что требуется от кандидата.

Логика добавления вакансии включает обязательные поля, такие как заголовок вакансии, описание вакансии, заработная плата (от и до), город (необязательно), а также другие дополнительные поля, которые могут быть добавлены по необходимости.

Реализация данного процесса будет представлена в Листинге 3.7.

const { title, description, salaryFrom, salaryTo, regionId, specializationId, statusVacancyId, cityId, workTypeId, employmentTypeId, levelEducationId, experienceLevelId, languages, conditions, skills, responsibilities, requirements } = data;

const { id: companyId } = company;

const vacancy = await prisma.vacancy.create({

data: {

...updateData,

company: {

connect: { id: companyId },

},

},

});

if (!vacancy) {

throw new APIError('Вакансия не создана', 500);

}

Листинг 3.6 – Создание вакансии

Если какой-либо параметр введен некорректно, генерируется ошибку и отправляет клиенту соответствующее сообщение об ошибке с кодом состояния.

### 3.2.5 Чат между работодателем и соискателем

Чат между работодателем и соискателем является важной частью взаимодействия в процессе поиска работы. Он позволяет уточнить условия, задать дополнительные вопросы или организовать дальнейшее общение, например, для телефонных звонков или общения в социальных сетях.

В данном приложении чат создается только после того, как соискатель откликнется на вакансию. После этого, работодатель и соискатель могут начать общение, но при этом соискатель не может отправлять сообщения, пока работодатель не примет его заявку. После того как работодатель одобрит заявку соискателя, чат становится активным и обе стороны могут обмениваться сообщениями.

Взаимодействие через чат происходит с использованием WebSocket, что позволяет обеспечивать реальное время для отправки и получения сообщений. Основным событием для отправки сообщений является sendMessage, который обрабатывается на сервере и пересылается всем участникам чата.

Логика работы этого события будет представлена в листинге 3.7.

@SubscribeMessage('sendMessage')

async handleSendMessage(

@MessageBody() data,

@ConnectedSocket() client: Socket,

): Promise<void> {

const userId = client['userId'];

if (userId) {

const user = await this.userService.getUserById(userId);

if (user) {

const message = await this.chatService.addMessage({

chatId: data.chatId,

senderId: senderId,

content: data.content,

isCompany: isCompany,

});

const allSessionsForReceiver =

this.connectedUsers[receiver?.id] || [];

allSessionsForReceiver.forEach((session: Socket) => {

session.emit('message', {

notice: `The ${isCompany ? 'company' : 'user'} ${user.email} has sent you a message`,

message: message,

});

});

}

}

}

Листинг 3.7 – Событие sendMessage

В чате между работодателем и соискателем каждое сообщение должно содержать определенные обязательные данные: получателя, уникальный идентификатор чата (chatId) и сам контент сообщения. Если какое-либо из этих обязательных полей отсутствует, сообщение не будет создано.

После того как сообщение успешно создается, оно отправляется всем подключенным устройствам участников чата, а также самому отправителю. Это гарантирует, что если пользователь подключен с нескольких устройств ,он получит уведомление на всех своих устройствах.

## 3.3 Разработка клиентской части web-приложения

Клиентская часть веб-приложения была разработана с учетом принципов кроссплатформенности и используется в браузере. Было принято решение о реализации SPA (Single Page Application), что позволяет приложению работать без перезагрузки страницы, динамически обновляя данные на основе запросов к серверу. Это позволяет значительно улучшить пользовательский опыт, так как приложение не требует перезагрузки при каждом взаимодействии.

Для общения с сервером используется асинхронная загрузка данных, что предотвращает задержки в интерфейсе при получении информации. Для выполнения запросов клиентская часть использует такие технологии, как AJAX и библиотеку Axios, которая облегчает работу с HTTP-запросами и позволяет гибко отправлять данные с определённым методом на заранее определённый путь. Это позволяет динамически обновлять страницы, не прерывая работу приложения, и улучшает общую отзывчивость интерфейса.

На рисунке 3.2 представлена структура проекта.

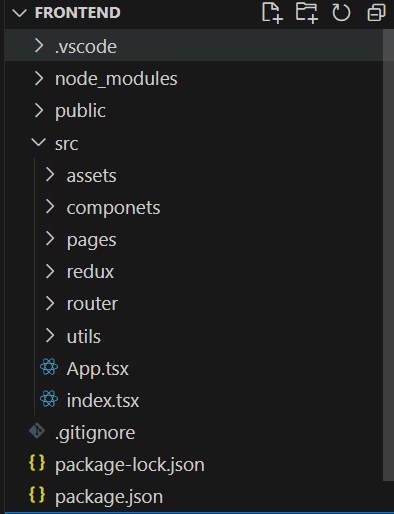


Рисунок 3.2 – Структура проекта

Структура клиентской части веб-приложения организована с учетом удобства разработки и масштабируемости. В папке components хранятся все компоненты, которые являются строительными блоками интерфейса, как простые, так и сложные. Папка pages предназначена для страниц приложения, каждая из которых представляет собой набор компонентов, отображаемых в зависимости от маршрута. В router сосредоточена логика маршрутизации, которая управляет переходами между страницами и определяет, какой компонент должен быть отображен для каждого маршрута. Папка util содержит вспомогательные функции, такие как обработка дат, валидация данных и обработка ошибок, которые могут быть использованы в различных частях приложения. В redux хранится логика управления состоянием через Redux, включая редюсеры, экшены и store, а также асинхронные запросы с использованием middleware. Папка assets включает статические файлы, такие как изображения, стили и шрифты, необходимые для отображения интерфейса. Такая структура проекта позволяет легко поддерживать и расширять приложение, разделяя ответственность между различными частями системы.

Для хранения глобального состояния в проекте используется React-Redux, что позволяет централизованно управлять данными и синхронизировать их между компонентами. Это упрощает передачу данных и делает код более читаемым и масштабируемым.

Для асинхронных запросов применяется RTK Query, который упрощает работу с API, автоматически генерируя экшены и редюсеры. Он также реализует кэширование данных, управление ошибками и синхронизацию состояния, что ускоряет разработку и уменьшает количество шаблонного кода.

В листинге 3.8 будет представлен слайс, который управляет состоянием.

const initialState = {

user: null,

loading: true,

socket: null,

};

const AuthSlice: any = createSlice({

name: "Auth",

initialState,

reducers: {

setUser(state: AuthStateType, action: PayloadAction<any>) {

state.user = action.payload;

}, },

});

Листинг 3.8 –Слайс для хранения состояния

Запрос в RTK Query будет представлен в листинге 3.9. В этом запросе указывается URL сервера, а также метод, который нужно использовать (например, POST или GET), и передаваемые данные (в теле запроса — body), если они есть. Это позволяет создавать хук для вызова эндпоинта, который удобно использовать в компонентах. Этот хук автоматически управляет состоянием запроса, включая флаги для загрузки (loading), данные (data) и ошибки (error), что значительно упрощает обработку асинхронных запросов и состояния в приложении.

export const authApi = createApi({

reducerPath: "auth-api",

baseQuery: baseQueryWithToken,

endpoints: (builder) => ({

register: builder.mutation({

query: (user) => {

return {

url: "/auth/register",

method: "POST",

body: user,

};

},

}),

Листинг 3.9 –Запрос в RTK Query

Основная маршрутизация в приложении настраивается в объекте RouterConfig, где передается компонент App, являющийся точкой входа в приложение. Это настраивается в файле index.tsx, где RouterConfig используется для создания маршрутов. Благодаря этому маршрутизатору, который управляет состоянием SPA (Single Page Application), страницы приложения обновляются без перезагрузки страницы, что обеспечивает плавный и быстрый переход между компонентами.

Пример кода маршрутизации в RouterConfig будет представлен в листинге 3.10.

if (store.isLoading) {

return <div>Loading</div>;

} else {

return (

<Router>

<div className="app-wrapper">

{store.isAuth && <SideBar />}

<div className="content">

<Routes>

<Route path="/login" element={<Login />} />

<Route path="/registration" element={<RegistrationPage />} />

{store.isAuth ? (

<>

<Route path="/gallery" element={<MainPage />} />

<Route path="/profile" element={<ProfilePage />} />

<Route

path="/publication/:id"

element={<PublicationPage />}

/>

<Route path="\*" element={<HelloPage />} />

</>

) : (

""

)}

<Route path="\*" element={<HelloPage />} />

</Routes>

</div>

</div>

</Router>

);

}

Листинг 3.10 –Настрйока маршрутизации

В данном коде также указывается страница ошибки (errorElement), которая отображается в случае возникновения ошибки маршрутизации. Это позволяет пользователю видеть понятное сообщение или компонент, информирующий о проблемах с загрузкой маршрута. Компонент Error будет показан, если произойдет ошибка в процессе загрузки страницы.

Кроме того, используется компонент Suspense, который помогает обработать асинхронную загрузку приложения или его компонентов. Если компонент еще не загружен, то Suspense отобразит компонент Loading — индикатор загрузки. Это обеспечивает плавную работу приложения и улучшает пользовательский опыт, так как вместо пустой страницы или задержки пользователи увидят индикатор загрузки, пока контент не будет готов.

Внутри element компонента App будет загружаться все остальное приложение. Это позволяет динамически подгружать нужные страницы и компоненты в зависимости от маршрута, при этом поддерживая SPA-архитектуру, в которой приложение не перезагружается полностью, а только меняется его содержимое.

Таким образом, в случае с ошибкой будет показываться страница ошибки, а если приложение или его компоненты все еще загружаются, пользователю будет отображаться индикатор загрузки, улучшая восприятие работы приложения.

## 3.4 Выводы по разделу

В данном разделе рассмотрены структура и технологии как серверной, так и клиентской частей приложения. Серверная часть основана на NestJS, который реализует регистрацию, авторизацию, добавление вакансий и чат между работодателями и соискателями, с использованием хеширования паролей и двух токенов (access и refresh) для безопасности. Клиентская часть построена на React с React-Redux для управления состоянием и RTK Query для асинхронных запросов, а маршрутизация и асинхронная загрузка страниц реализованы с использованием react-router-dom и компонента Suspense.

# Тестирование web-приложения

## 4.1 Тестирование авторизации и регистрации

При тестировании данного приложения, были применены сценарии, которые могли бы привести к ошибке. В этой главе рассмотрим некоторые такие сценарии и посмотрим на их обработку.

В момент авторизации, возможна такая ситуация, в которой пользователь ничего не ввёл. Обработка данного сценария приведена на рисунке 4.1.

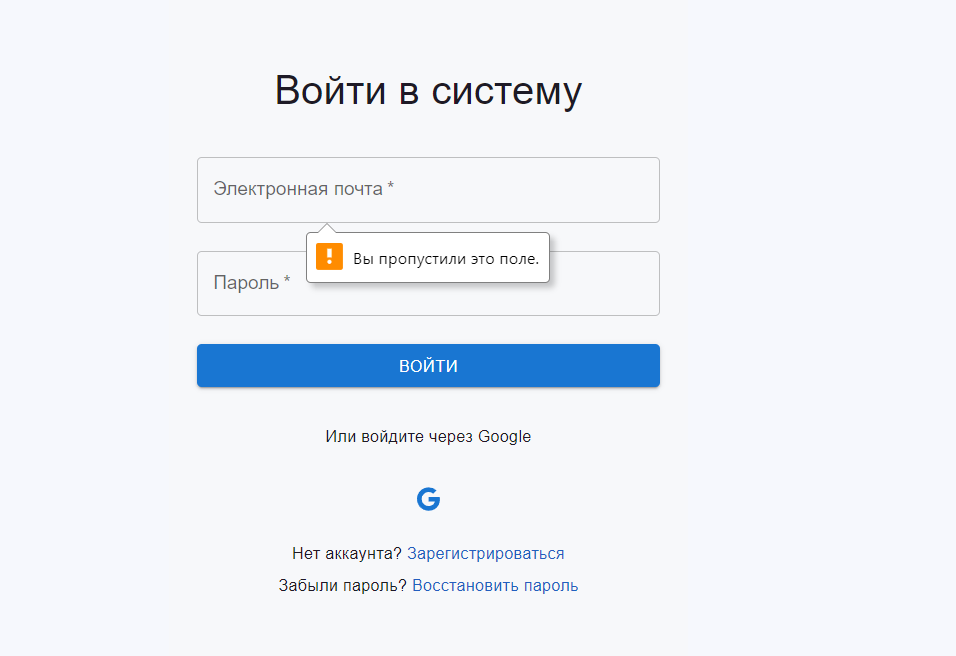


Рисунок 4.1 — Обработка пустых полей при авторизации

При вводе некорректных данных, возникает ошибка, представленная на рисунке 4.2.

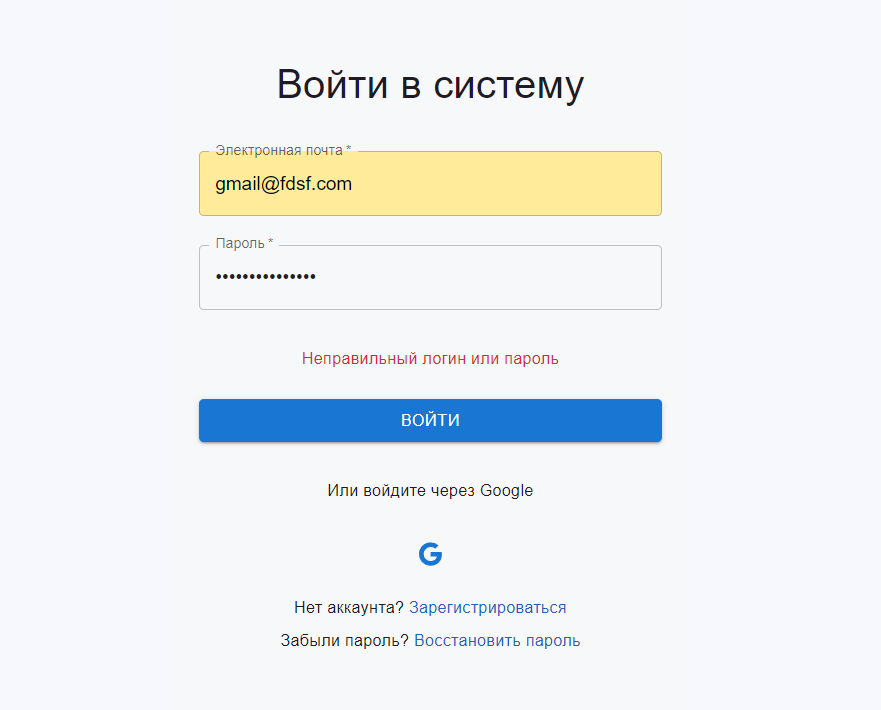


Рисунок 4.2 — Обработка некорректных данных

При регистрации может быть попытка создать аккаунт с логином, который уже занят, поэтому было принято решение проверить это.

Предусловием будет то, что уже создан аккаунт с логином gmail@fdsf.com, но я буду пытать создать еще один аккаунт с таким логином. Результат данного теста представлен на рисунке 4.3.

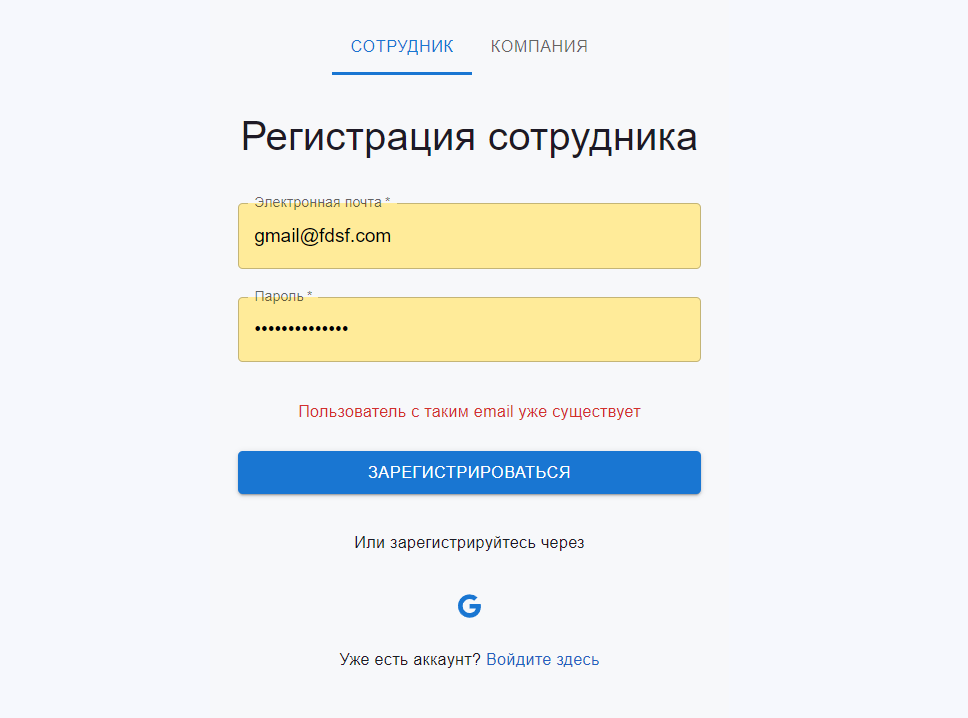


Рисунок 4.3 — Обработка ввода уже зарегистрированного email

Дальше нужно попробовать ввести некорректные данные. Например, ввести пароль, состоящий из 3 символов. Результат данного тестирования представлен на рисунке 4.4.

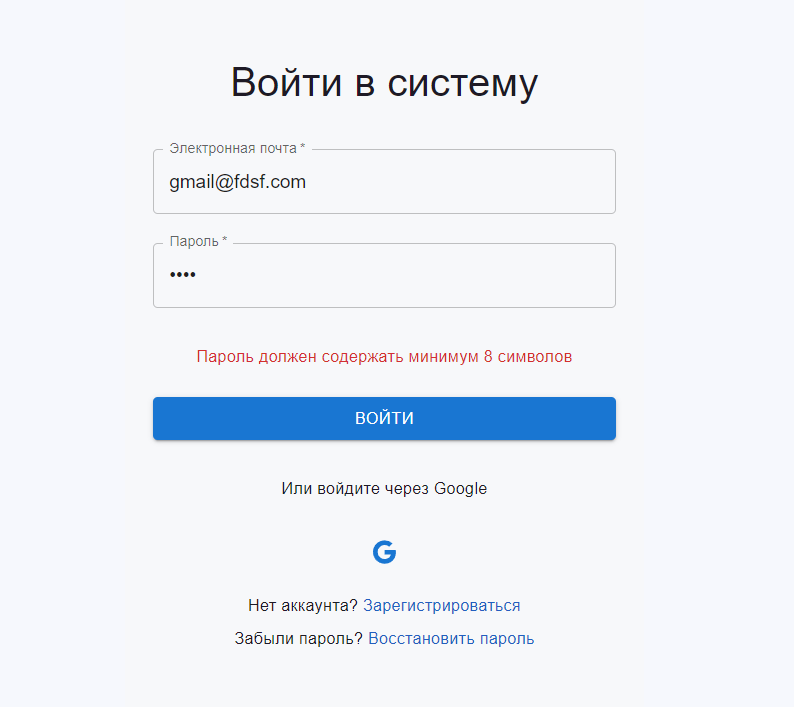


Рисунок 4.4 — Результат c паролем, состоящим из 3 символов

Результаты тестирования авторизации и регистрации говорят о том, что она работает корректно.

## 4.2 Тестирование добавление вакансии

Тестирование добавления вакансии является важным этапом для проверки корректной работы функционала. Рассмотрим несколько сценариев. Первый сценарий — это попытка добавить вакансию с пустыми всеми полями. В этом случае система должна отклонить запрос и отобразить соответствующее сообщение об ошибке.

На рисунке 4.7 представлен результат тестирования.

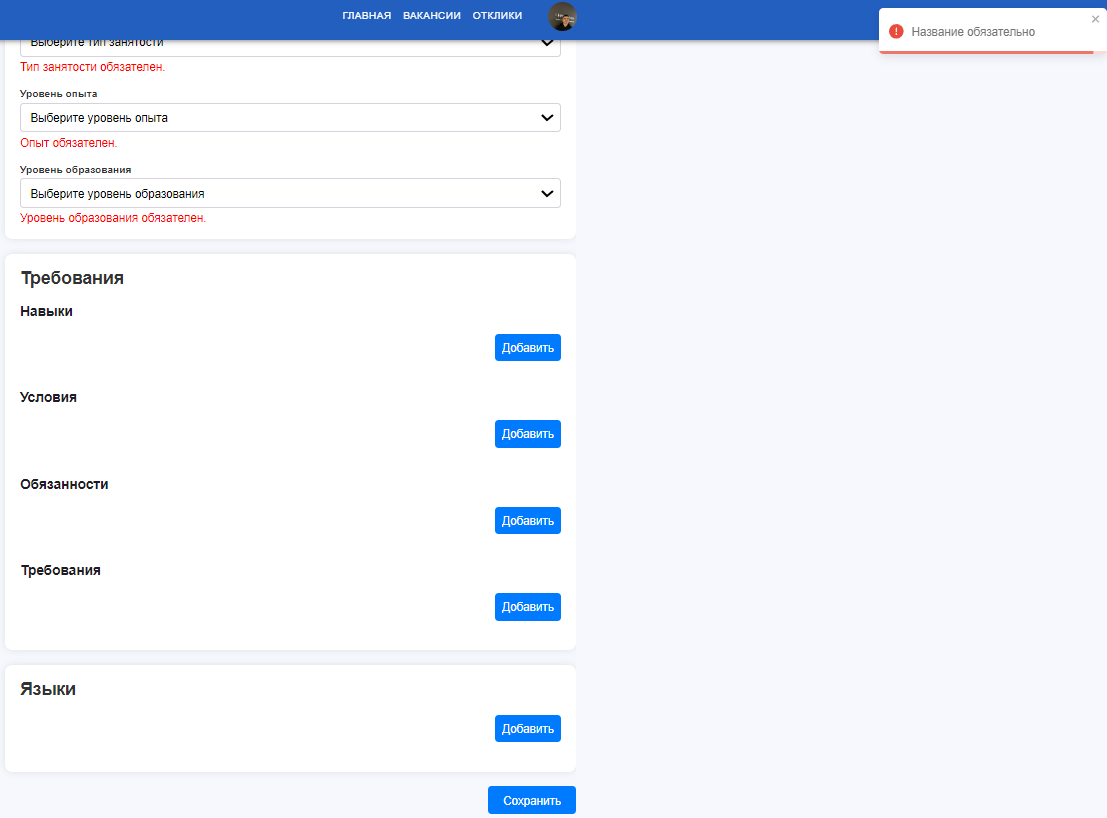


Рисунок 4.7 – Добавление вакансии с пустыми полями

При создании вакансии необходимо заполнить обязательные поля. Если они отсутствуют, создание вакансии будет невозможно. Проведенное тестирование подтвердило, что система корректно обрабатывает такой сценарий.

Другой сценарий — заполнение всех полей, включая ввод текста с максимально допустимым количеством символов.

На рисунке 4.8 представлен результат этого тестирования.

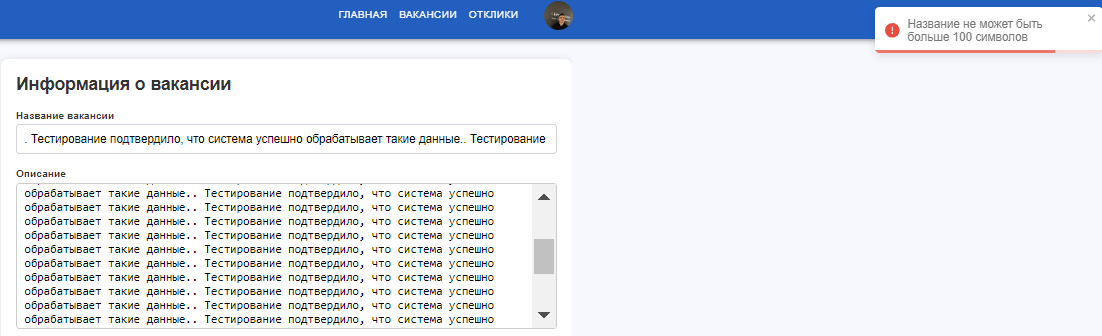


Рисунок 4.8 – Поля с большим количеством текста

Как видно из результатов тестирования, для каждого поля предусмотрены проверки на ограниченное количество символов, что исключает возможность создания вакансии с превышением допустимого объема данных.

Следующий этап тестирования заключался в попытке добавить вакансию с неправильным типом данных, например, указав для зарплаты буквы вместо чисел. На рисунке 4.9 представлен результат данного тестирования.

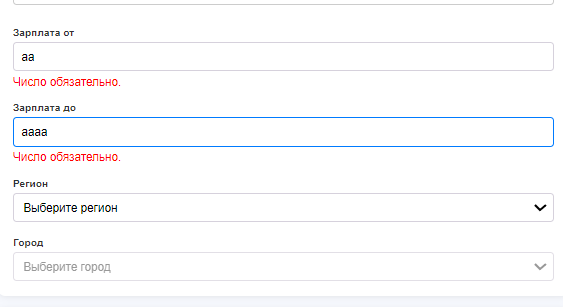


Рисунок 4.9 – Поля c другим типом данных

При выполнении всех требований и корректном заполнении обязательных полей вакансия успешно создается

## 4.3 Тестирование поиска вакансий

В функционале поиска предусмотрены определенные поля для ввода текста и выпадающие меню для выбора параметров. Первый сценарий тестирования предполагает ввод пустого title с несколькими пробелами. Ожидается, что в результате будут показаны все доступные вакансии.

На рисунке 4.10 представлен результат данного тестирования.

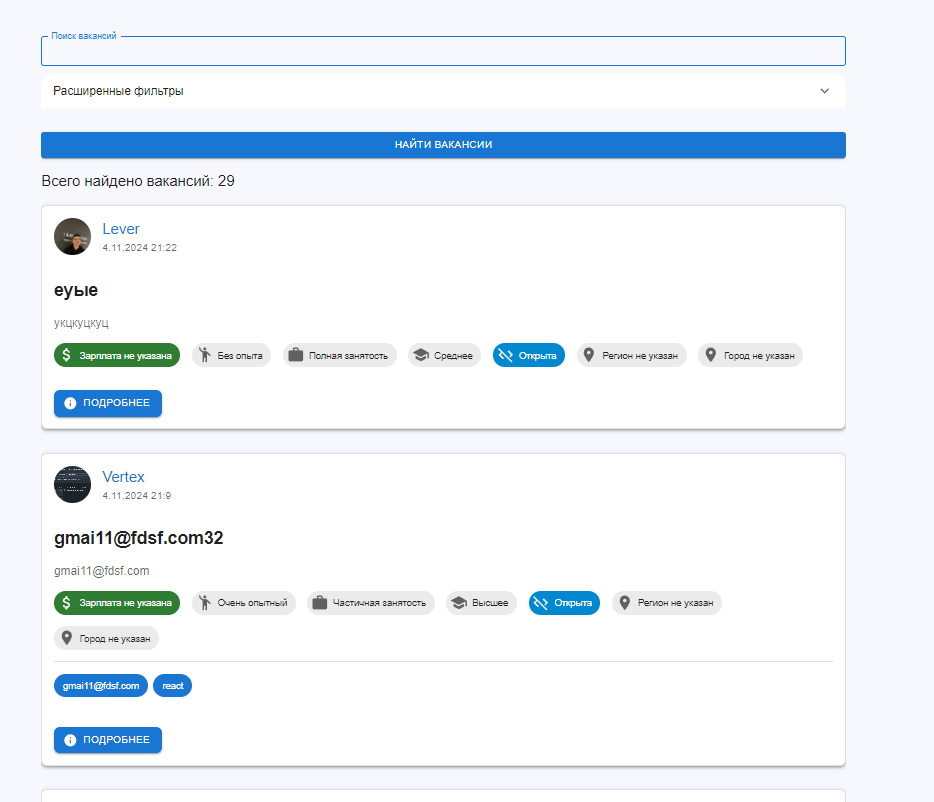


Рисунок 4.10 – Поиск с пробелами в input

Этот тест был успешно выполнен. Следующим этапом было тестирование поиска вакансии с именем ron, имея в виду часть слова "Frontend". На рисунке 4.11 представлен результат данного тестирования.

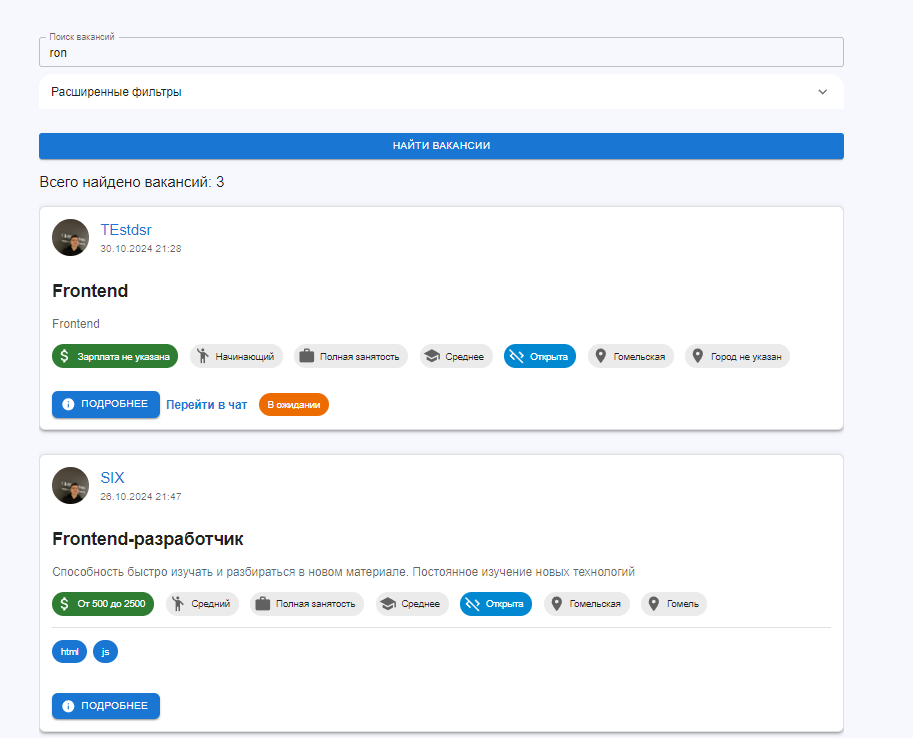


Рисунок 4.11 – Поиск определнным именем

Следующим этапом тестирования была проверка фильтрации по дате. На рисунке 4.12 представлен результат данного тестирования.

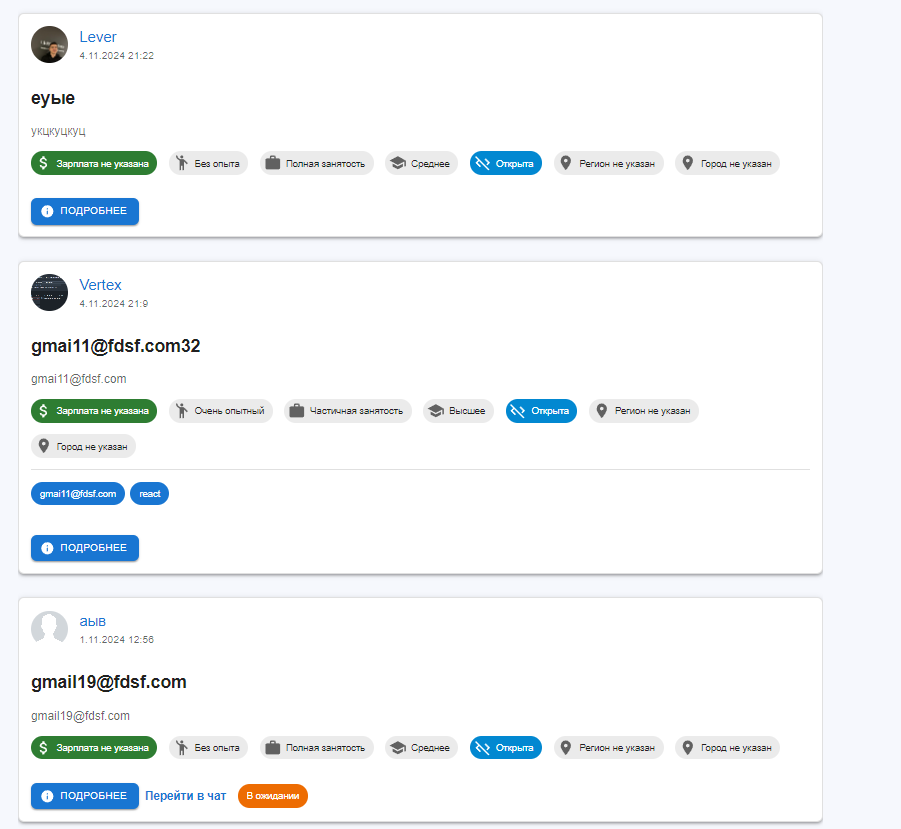


Рисунок 4.12 – Сортировка по дате

Тесты по поиску вакансий были проведены успешно, подтверждая корректную работу фильтров и отображение результатов.

## 4.4 Тестирование заполнения профиля

Для тестирования заполнения профиля был проведен сценарий с пустыми данными. На рисунке 4.13 представлен результат данного тестирования.

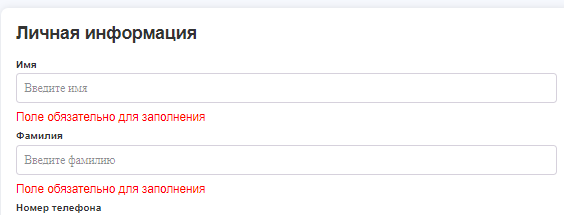


Рисунок 4.13 – Проверка введения пустых полей

Далее была проведена проверка на возможность загрузки файла неверного формата вместо изображения, так как система принимает только графические файлы. На рисунке 4.14 представлен результат данного тестирования.

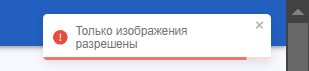


Рисунок 4.14 — Результат c выбором другого формата файла

Тест прошел успешно, так как при попытке отправить файл неверного формата, не являющегося изображением, система корректно отклонила загрузку и не позволила продолжить процесс.

## 4.4 Выводы по разделу

Таким образом, были проведены тестирования нового функционала, включая добавление вакансий, поиск вакансий, а также заполнение профиля пользователя. Все тесты прошли успешно, функционал работает корректно. Система правильно обрабатывает обязательные поля, фильтрацию данных и загрузку файлов. В случае ввода некорректных данных система корректно генерирует ошибки и предотвращает сохранение неверных данных. Проведенные тесты подтвердили правильность работы всех функций, что свидетельствует о высокой надежности и безопасности приложения.

# 5 Руководство пользователя

## 5.1 Руководство пользователя

При открытии веб-приложения, первой страницей является страница авторизации. Если у пользователя еще нет аккаунта, он может перейти на страницу регистрации, где необходимо выбрать роль: соискатель или работодатель. Страница регистрации, на которой происходит выбор роли, представлена на рисунке 5.1.

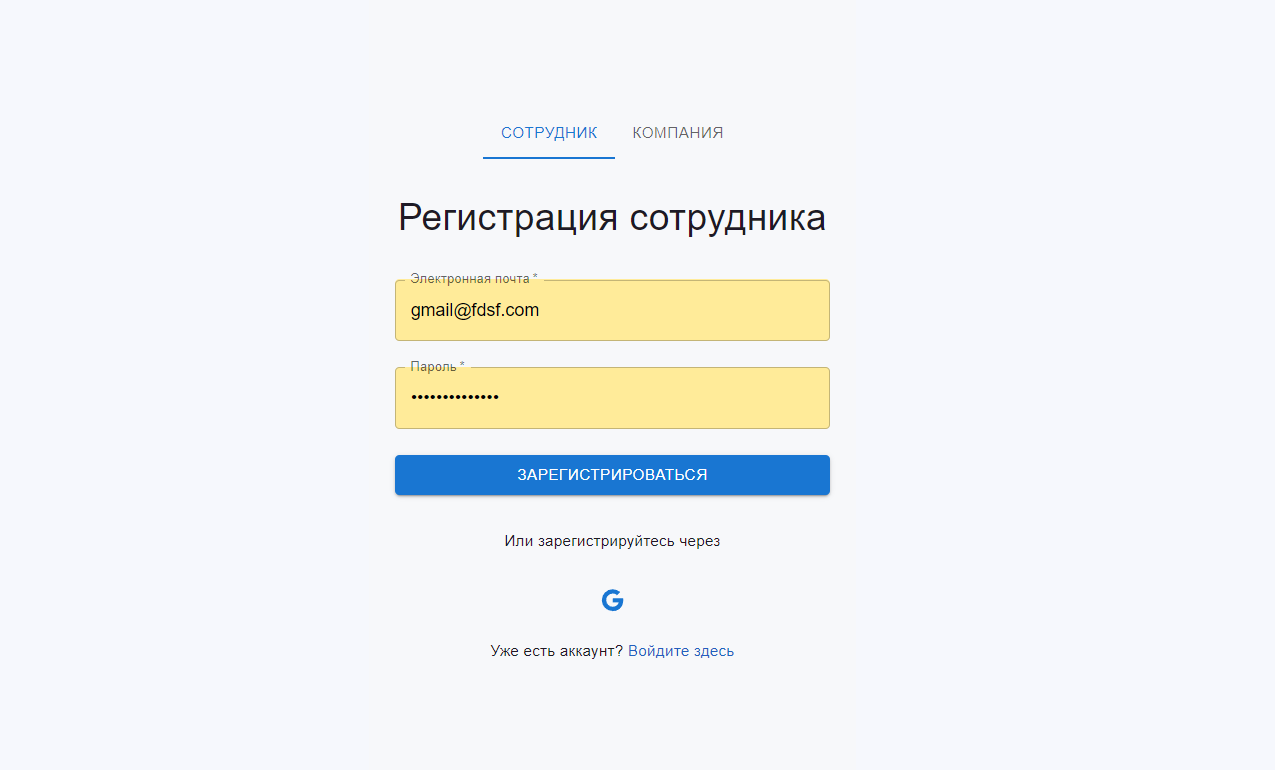


Рисунок 5.1 — Страница регистрации

После выбора роли, в зависимости от выбранной роли, будут отображаться разные страницы и функциональные возможности. Например, для соискателя будет доступен профиль с возможностью отклика на вакансии, для работодателя — создание и управление вакансиями.

### 5.1.1 Соискатель

После регистрации, соискателю будет доступна домашняя страница, на которой будут отображаться вакансии компаний, на которые он подписан. Эта страница служит основным интерфейсом для поиска и отклика на вакансии, с возможностью взаимодействия с работодателями. На данной странице также могут быть фильтры для поиска, чтобы соискатель мог уточнить свои предпочтения по вакансии. На рисунке 5.2 представлена эта страница домашнего экрана соискателя.

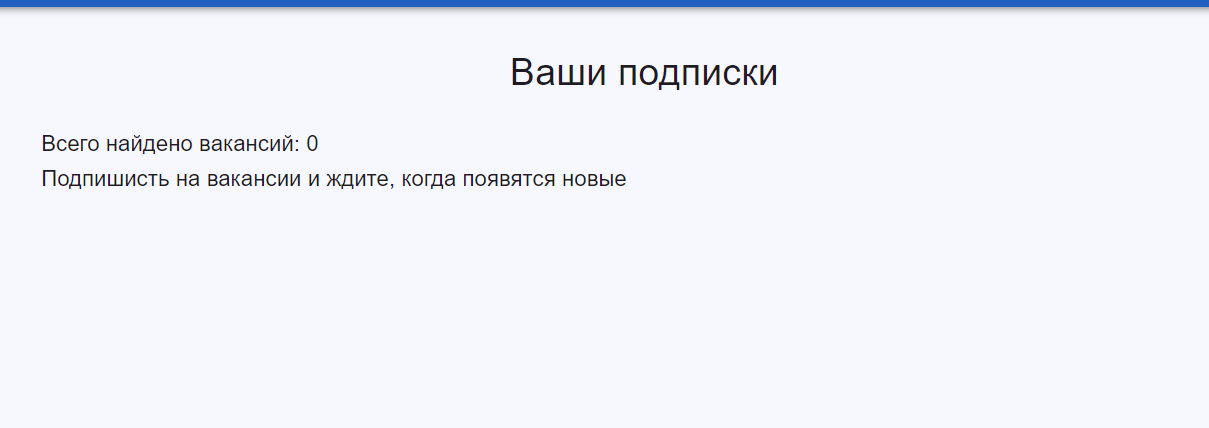


Рисунок 5.2 — Домашняя страница

На странице компании отображается список всех компаний, а справа отображаются только те компании, на которые пользователь подписан. Это позволяет быстро ориентироваться и находить подходящие компании. На рисунке 5.3 представлена данная страница, где показан список компаний с возможностью выбора подписанных.

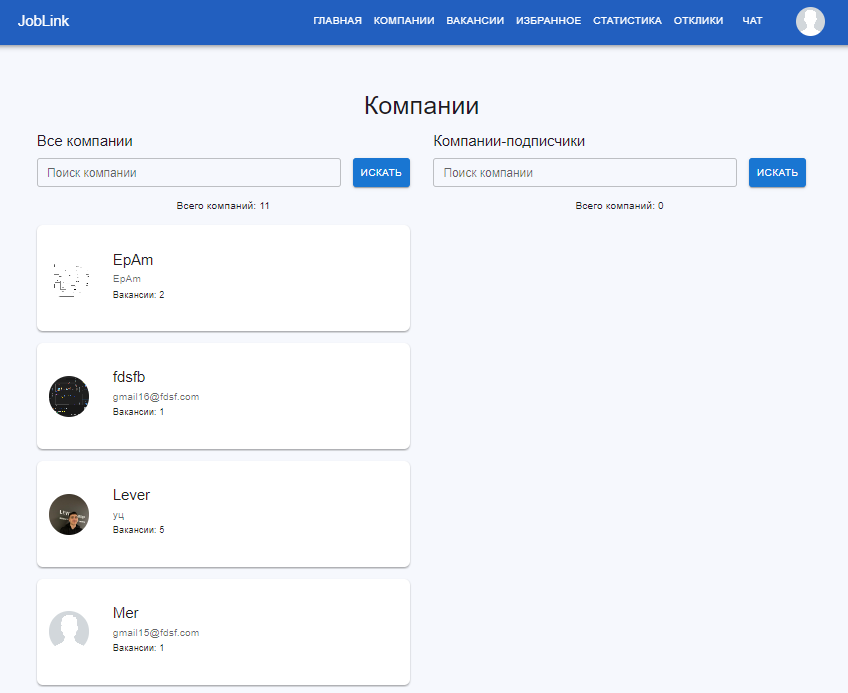


Рисунок 5.3 — Страница компании

Следующая страница — это страница вакансий, на которой соискатель может по определённым параметрам, таким как должность, зарплата, опыт работы и другие фильтры, найти подходящую вакансию. Это дает возможность пользователю быстро сузить поиск и найти предложения, которые соответствуют его запросам и ожиданиям. На рисунке 5.4 представлена эта страница с фильтрами, которые позволяют точнее подобрать вакансии по нужным критериям.

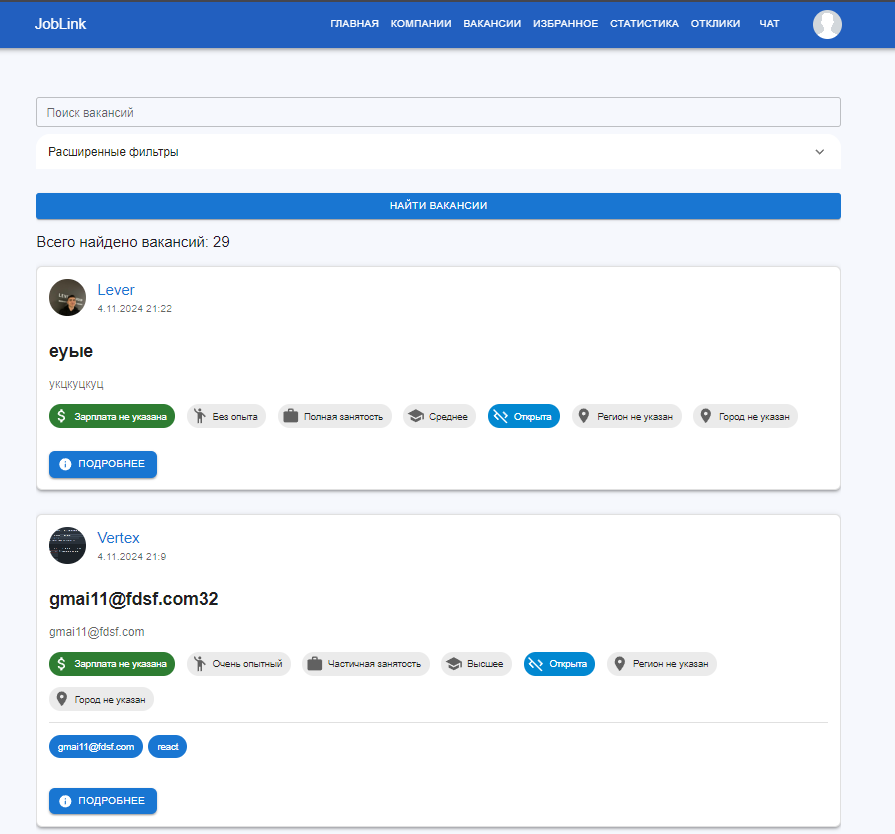


Рисунок 5.4 — Страница для поиска вакансий

Также соискатель может перейти на страницу вакансии, где он сможет подробно ознакомиться с всей информацией о вакансии, включая требования, обязанности и условия работы. На этой странице он может откликнуться на вакансию, добавить её в избранное или удалить из избранных. Если соискатель откликнется на вакансию, он также получит возможность перейти в чат с работодателем для дальнейшего общения. На рисунке 5.5 представлена данная страница с возможностью отклика, добавления в избранное и перехода в чат.

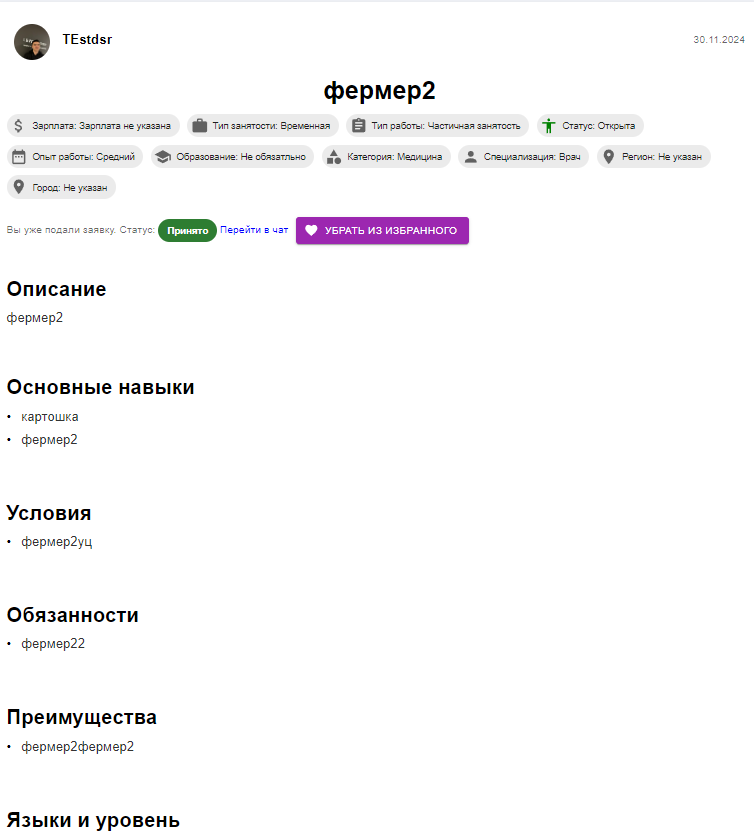


Рисунок 5.5 — Страница вакансии

Следующая страница — это страница Избранные, где соискатель может просмотреть все вакансии, которые он добавил в избранное. Это позволяет ему легко вернуться к интересующим его вакансиям позже, подать заявку или рассмотреть другие детали. На рисунке 5.6 представлена страница Избранные, на которой отображаются все сохранённые вакансии для дальнейшего взаимодействия.

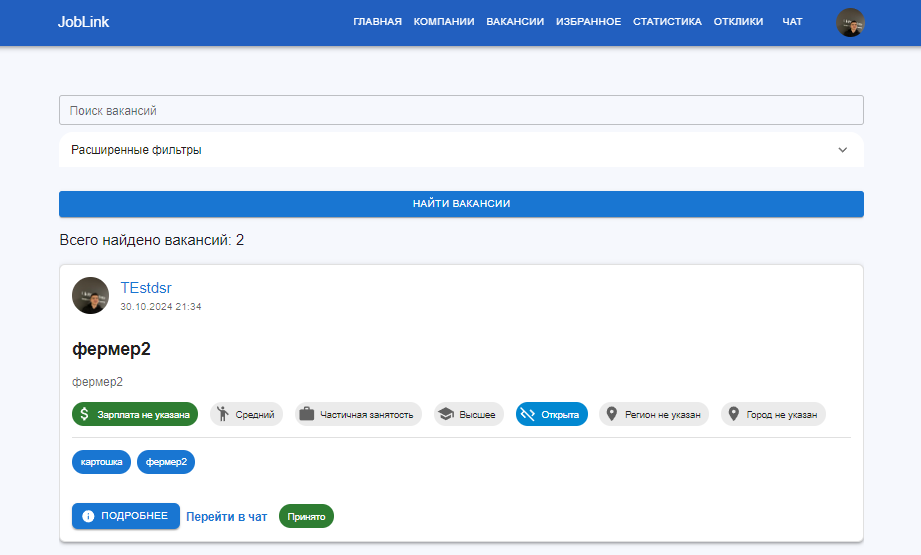


Рисунок 5.6 — Страница избранные

Следующая страница — это страница Статистика, где пользователь может просмотреть общую статистику, такую как количество откликов на вакансии, просмотры и другие данные, которые помогают оценить активность на платформе. На рисунке 5.7 представлена страница Статистика, на которой отображается подробная информация о взаимодействии пользователя с вакансиями.

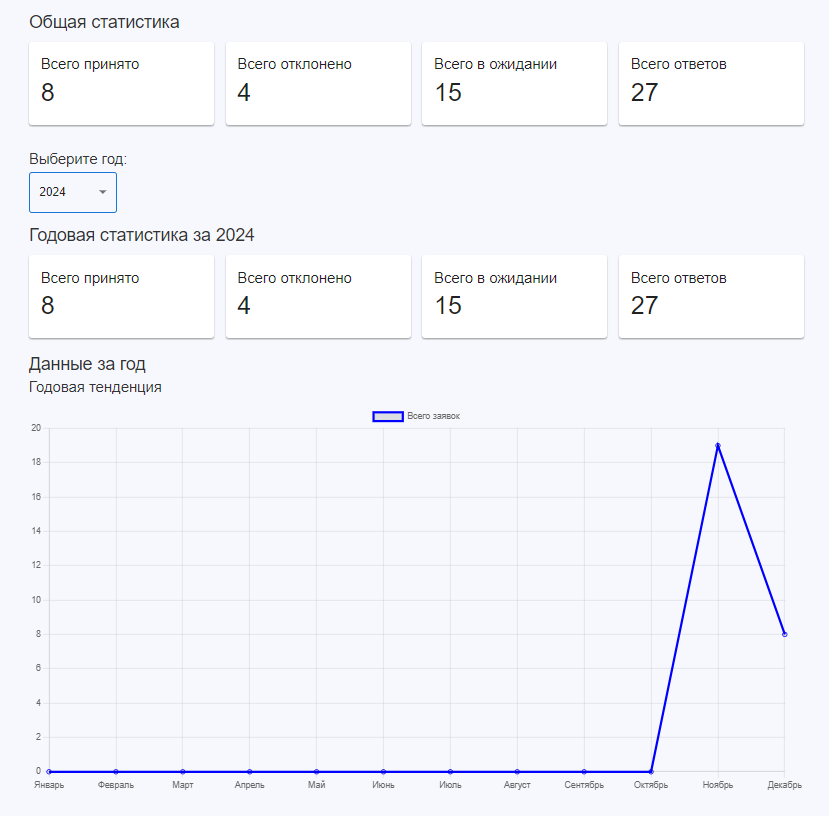


Рисунок 5.7 — Страница статистика

Есть также страница Отклики на вакансии, где соискатель может просмотреть все свои отклики и найти конкретную вакансию, на которую он откликался. На рисунке 5.8 представлена страница Отклики на вакансии, где отображается список всех откликов соискателя, а также возможность взаимодействовать с каждым откликом, например, перейти к чату с работодателем или просмотреть подробности вакансии.

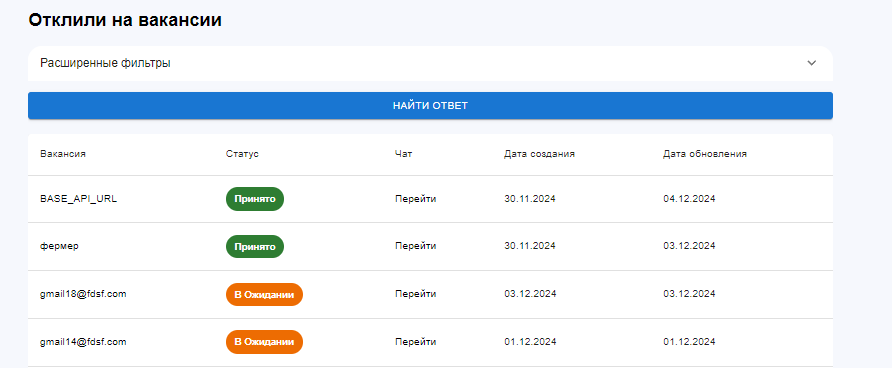


Рисунок 5.8 — Страница отклики на вакансию

Дальнейшая страница — это чаты, где соискатель может перейти к чату с конкретным работодателем. При переходе откроется сам чат, где соискатель и работодатель могут обмениваться сообщениями. На рисунке 5.9 представлена страница с чатами, которая отображает все чаты соискателя с работодателями, что позволяет легко отслеживать и взаимодействовать с откликами на вакансии.

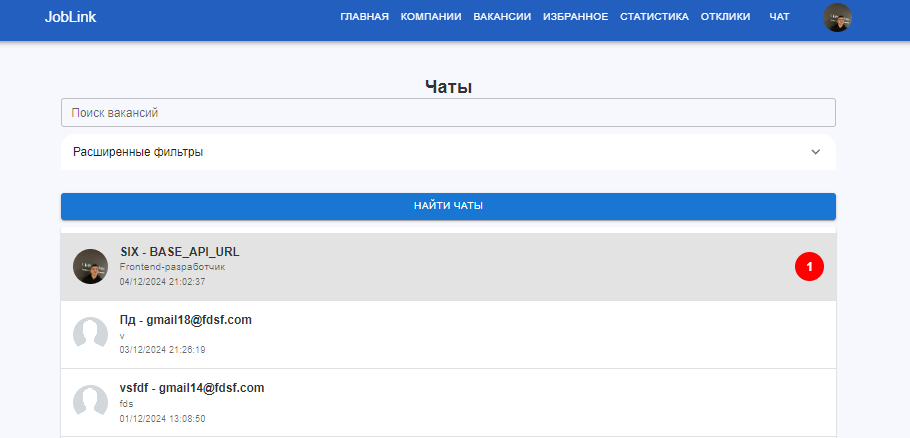


Рисунок 5.9 — Страница чаты

Самая важная страница для соискателя — это профиль. На этой странице соискатель может просматривать свой профиль, а также перейти к редактированию информации. На рисунке 5.10 представлена страница профиля соискателя, где отображается вся информация о пользователе, а также доступ к редактированию.

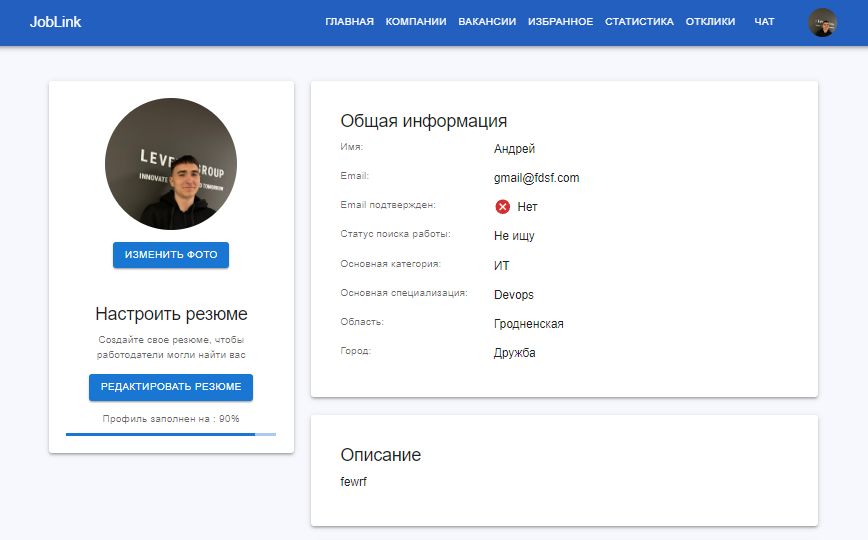


Рисунок 5.10— Страница профиль соискателя

Таким образом, выглядит руководство пользователя для соискателя. На протяжении всего процесса соискатель может зарегистрироваться, заполнить свой профиль, просматривать вакансии, откликаться на них, добавлять в избранное и взаимодействовать с работодателями через чат. Важно отметить, что каждый шаг включает в себя доступ к необходимой информации, которая помогает соискателю эффективно искать работу, поддерживать связь с работодателями и управлять своей карьерой через интерфейс приложения.

### 5.1.2 Работодатель

Для работодателя после регистрации появляется страница редактирования профиля компании, на которой необходимо указать название компании, так как без этого создание профиля невозможно. На рисунке 5.11 представлена страница редактирования профиля компании. На данной странице работодатель может указать важную информацию о компании, такую как название, описание, которые необходимы для правильного отображения информации о компании на платформе.

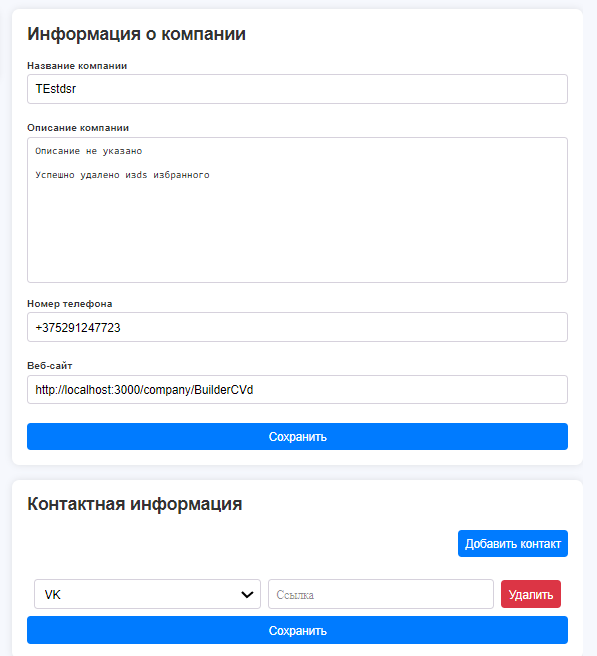


Рисунок 5.10— Страница профиля соискателя

После этого работодатель может перейти на страницу профиля компании, где будет отображаться название компании, описание и актуальные вакансии, которые размещены на платформе. На рисунке 5.11 представлена страница профиля компании, где можно увидеть полную информацию о компании, а также актуальные вакансии, доступные для соискателей.

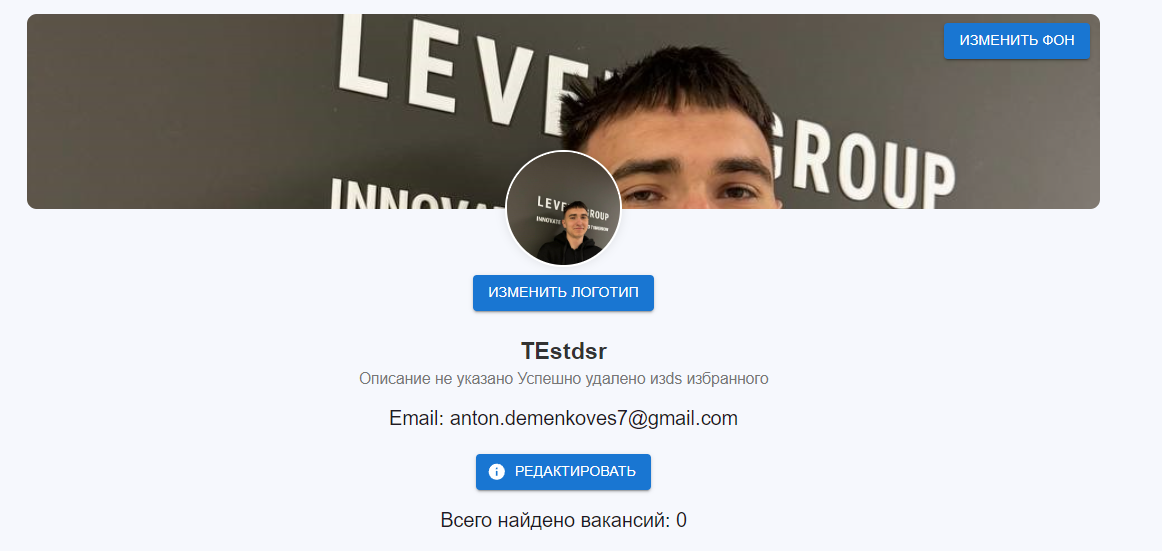


Рисунок 5.11— Страница профиля компании

Есть также страница домашняя, которая доступна работодателю и описывает общую статистику компании. На рисунке 5.12 представлена эта страница, где можно увидеть сводную информацию о деятельности компании на платформе.

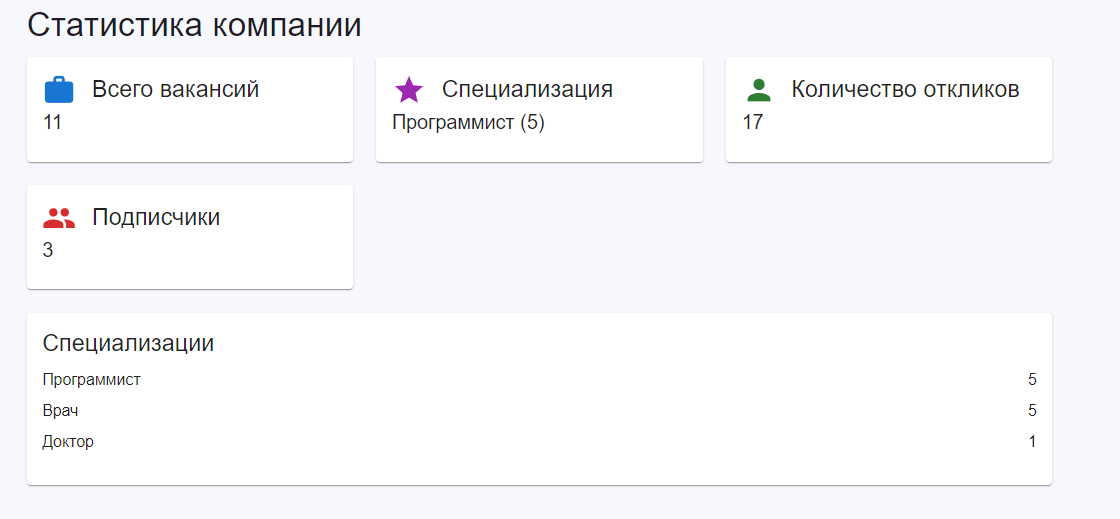


Рисунок 5.12— Страница статистики компании

Следующая страница — это страница ваканси", где отображаются все текущие вакансии компании. На этой странице работодатель может просматривать, редактировать и управлять своими вакансиями, а также перейти на страницу добавления новой вакансии. На рисунке 5.14 представлена страница вакансий для компании.

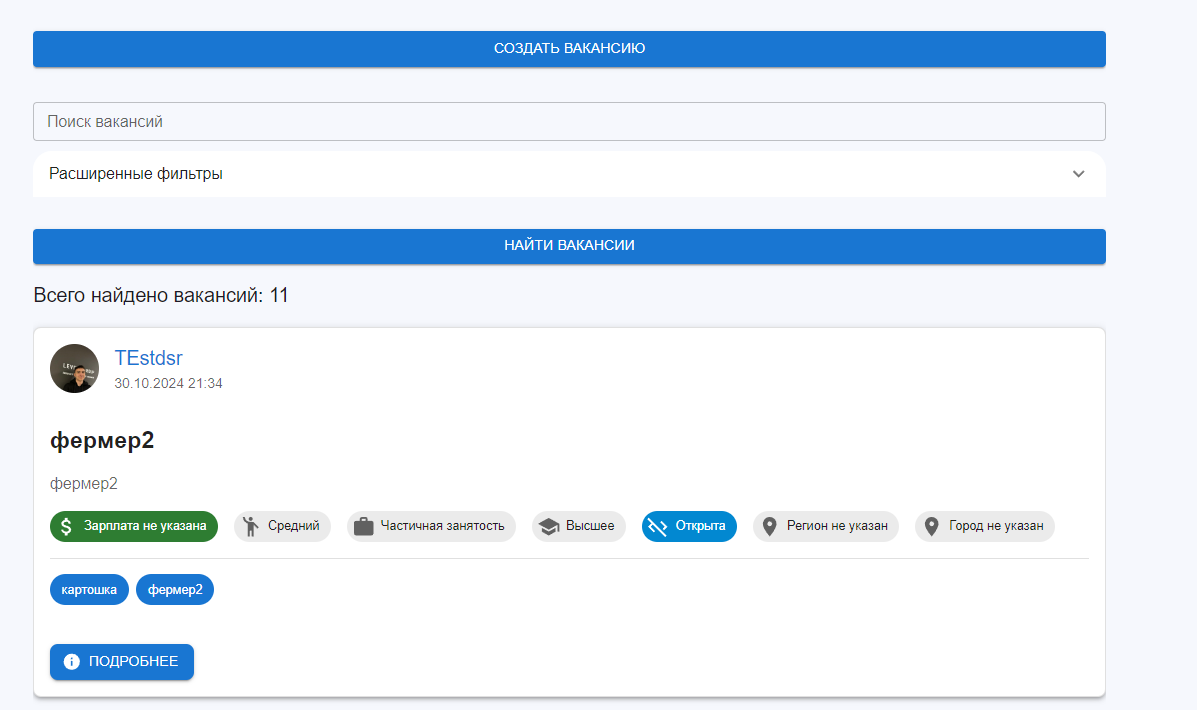


Рисунок 5.13— Страница вакансии компании

После этого существует страница отклики, на которой работодатель может просматривать отклики на свои вакансии. Здесь можно перейти к конкретной вакансии, просмотреть чаты с соискателями, а также взаимодействовать с откликами. На рисунке 5.14 представлена страница откликов.

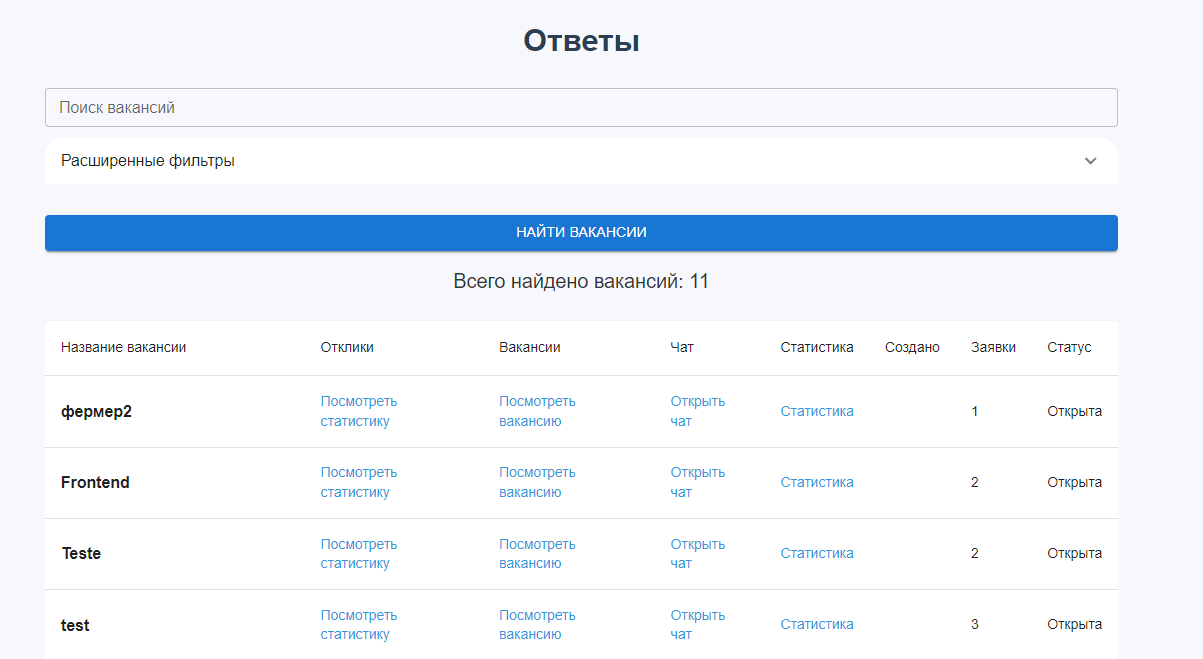


Рисунок 5.14— Страница отклики

На странице ответы на вакансию работодатель может просматривать полную информацию о соискателях, которые откликнулись на вакансию. Работодатель может оценить кандидатов, изучить их профили, а также отправить ответ по каждой вакансии. На рисунке 5.15 представлена данная страница.

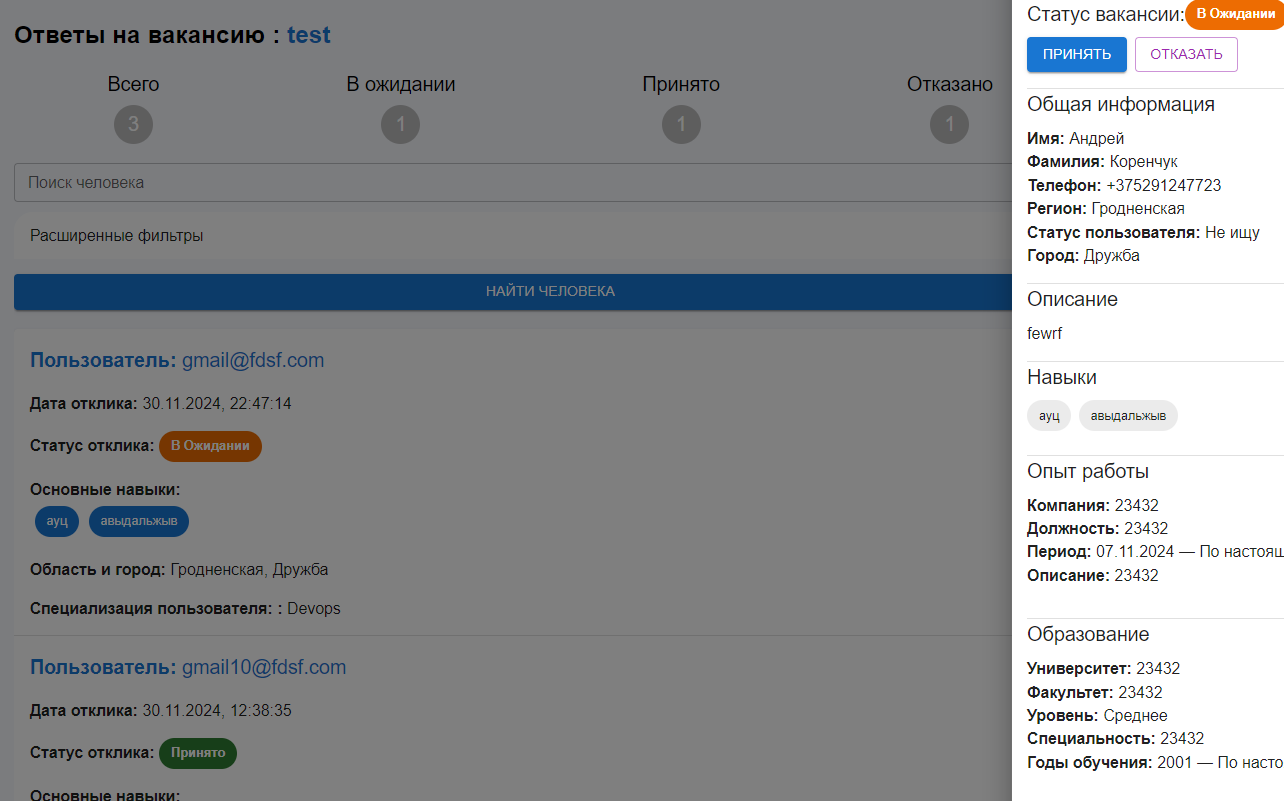


Рисунок 5.15— Страница ответы на вакансию

Таким образом, выглядит руководство пользователя для работодателя.

### 5.1.3 Менеджер

После авторизации менеджер попадает на домашнюю страницу, которая отображает общую статистику сайта. На данной странице представлены данные о всех сущностях, включая количество зарегистрированных пользователей, вакансий, откликов, компаний и другие ключевые показатели. На рисунке 5.16 представлена домашняя страница для менеджера.

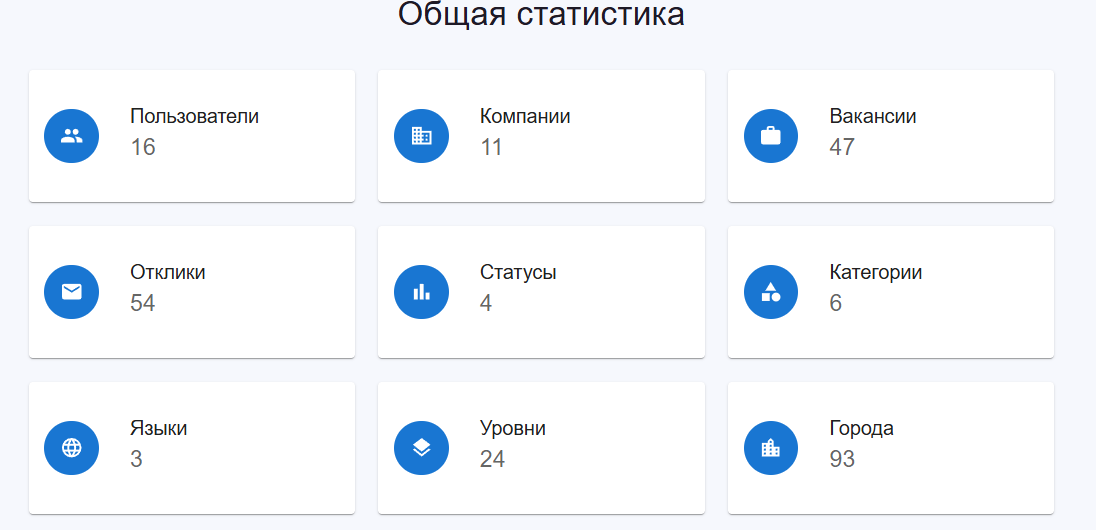


Рисунок 5.16— Домашняя страница для менеджера

Следующая страница — это страница категорий, где менеджер может добавлять новые категории и спецификации к существующим категориям. На рисунке 5.17 представлена данная страница, где менеджер имеет возможность как добавлять новые категории, так и редактировать спецификации, делая платформу более удобной для пользователей.

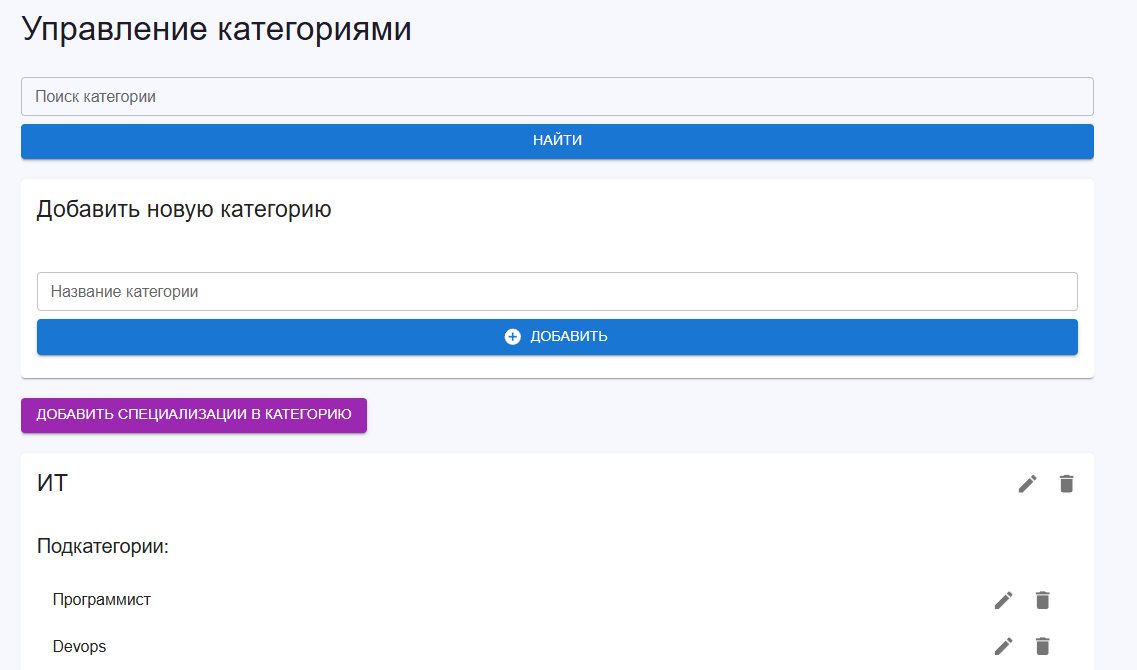


Рисунок 5.17— Страница категории

Также для менеджера есть страница с языками, на которой он может добавлять новые языки, а также страница городов и областей, где он имеет возможность добавлять новые города и области или редактировать существующие. На рисунке 5.18 представлена страница города.

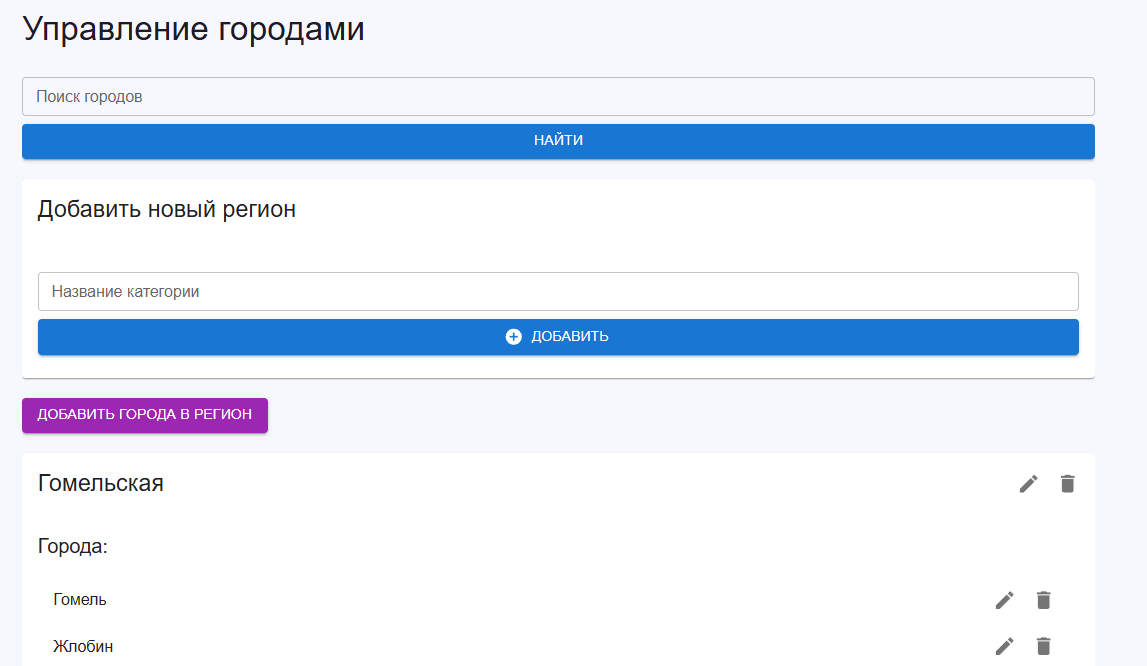


Рисунок 5.18— Страница города

Таким образом, выглядит руководство пользователя для менеджера.

## 5.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, которая соединяет базу данных и React приложение. Для этого необходимо запустить скрипт, который настроит соединение с базой данных и запустит сервер.

2. Запустить React приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером. Для этого необходимо запустить команду для сборки и запуска React приложения.

После выполнения этих шагов приложение будет полностью готово к работе и пользователь сможет начать использовать его функционал.

## 5.3 Выводы по разделу

В данном разделе были рассмотрены руководства пользователя для всех ролей: соискателя, работодателя и менеджера. Для каждой роли были подробно описаны доступные функции и действия, которые могут быть выполнены на платформе, начиная с регистрации и профиля, заканчивая управлением вакансиями, откликами и взаимодействием через чаты. На примерах с изображениями показано, как правильно использовать приложение и какие действия доступны пользователям в зависимости от их роли. Таким образом, был наглядно продемонстрирован процесс работы с платформой, что делает использование приложения интуитивно понятным и удобным для всех категорий пользователей.

# Заключение

В процессе разработки веб-приложения для создания и управления заказами кастомизированной одежды у художников была достигнута цель предоставления удобного инструмента для взаимодействия между клиентами, художниками и администраторами. Для реализации проекта был использован стек технологий, включающий Nest.js, React и PostgreSQL, что позволило обеспечить стабильную серверную часть, гибкий и интуитивно понятный интерфейс, а также надежное хранение данных.

В рамках разработки были реализованы следующие ключевые функции: регистрация и авторизация пользователей с учетом различных ролей (клиент, художник, администратор); создание и управление заказами клиентами; возможность художников принимать или отклонять заказы; публикация портфолио художниками; управление категориями, заказами и пользователями администраторами. Также была добавлена возможность общения между клиентами и художниками через встроенный чат.

Процесс разработки включал проектирование базы данных, создание серверной и клиентской частей, а также интеграцию всех необходимых функций для обеспечения удобной работы приложения. Все функциональные возможности прошли этап тестирования, и проект успешно подтвердил свою работоспособность, соответствуя всем техническим требованиям.

Тестирование показало стабильную работу приложения, включая взаимодействие с заказами, профилями пользователей и чатом. Ошибки и недочеты в процессе тестирования не выявлены, а все заявленные сценарии были успешно реализованы.

Разработанное приложение готово к эксплуатации и предоставляет простой и эффективный инструмент для управления заказами кастомизированной одежды, взаимодействия клиентов и художников, а также администрирования системы.

# Список используемых источников

1. Petfinder.com [Электронный ресурс] / Режим доступа: https://www.petfinder.com/ – Дата доступа: 08.12.2024.
2. Adoptapet.com [Электронный ресурс] / Режим доступа: https://www.adoptapet.com/ – Дата доступа: 08.12.2024.
3. Finder [Электронный ресурс] / Режим доступа: https://finder.work– Дата доступа: 08.12.2024.
4. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 08.12.2024.
5. Руководство по React [Электронный ресурс] – Режим доступа: https://metanit.com/web/react/– Дата доступа: 08.12.2024.
6. React hooks in Axios interceptors [Электронный ресурс] – Режим доступа: https://dev.to/arianhamdi/react-hooks-in-axios-interceptors-3e1h – Дата доступа: 08.12.2024.
7. How to Set Up a Websocket Server with Node.js and Express [Электронный ресурс] – Режим доступа: https://cheatcode.co/tutorials/how-to-set-up-a-websocket-server-with-node-js-and-express – Дата доступа: 08.12.2024.

# Приложение А

co

Листинг 1 – Функция регистраиции

# Приложение Б