МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

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 курсовому проекту на тему:**

Web-приложение «Поиск работы»

Выполнил студент Сенченя Владислав Игоревич

(Ф.И.О.)

Руководитель проекта ст.препод. Дубовик М.В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты ст.препод. Дубовик М.В.

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

Нормоконтролер ст.препод. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[**Введение** 3](#_Toc135264119)

[**1** **Постановка задачи** 4](#_Toc135264120)

[**1.1** **Алгоритмы решения** 4](#_Toc135264121)

[**1.2** **Обзор прототипов** 5](#_Toc135264122)

[**2** **Проектирование web-приложения** 6](#_Toc135264123)

[**2.1** **Обобщённая архитектура** 6](#_Toc135264124)

[**2.2** **Серверная часть приложения** 7](#_Toc135264125)

[**2.3** **Клиентская часть приложения** 8](#_Toc135264126)

[**2.4** **Безопасность. Аутентификация и авторизация** 8](#_Toc135264127)

[**2.5** **Общее описание работы** 8](#_Toc135264128)

[**2.6** **Диаграмма вариантов использования** 9](#_Toc135264129)

[**2.7** **Проектирование базы данных** 11](#_Toc135264130)

[**3** **Разработка web-приложения** 13](#_Toc135264131)

[**3.1** **Серверная часть** 13](#_Toc135264132)

[**3.2** **JWT. Аутентификация и авторизация** 13](#_Toc135264133)

[**3.3** **Обмен данными в реальном времени** 14](#_Toc135264134)

[**3.4** **разработки документации** 15](#_Toc135264135)

[**3.5** **Клиентская часть** 16](#_Toc135264136)

[**3.6** **Реализация базы данных** 17](#_Toc135264137)

[**3.7** **Конечные точки** 17](#_Toc135264138)

[**4** **Тестирование web-приложения** 20](#_Toc135264139)

[**4.1** **Форма регистрации** 20](#_Toc135264140)

[**4.2** **Чат** 21](#_Toc135264141)

[**5** **Руководство пользователя** 23](#_Toc135264142)

[**5.1** **Список вакансий** 23](#_Toc135264143)

[**5.2** **Отклик на вакансию** 24](#_Toc135264144)

[**5.3**  **Добавление вакансии в избранное** 25](#_Toc135264145)

[**5.4** **Создание компании** 26](#_Toc135264146)

[**Заключение** 28](#_Toc135264147)

[**Список используемых источников** 29](#_Toc135264148)

[**Приложение А** 30](#_Toc135264149)

[**Приложение Б** 31](#_Toc135264150)

# **Введение**

Современный рынок труда становится всё более цифровым. Всё большее количество вакансий публикуется онлайн, а процесс отбора кандидатов во многих компаниях полностью автоматизирован. В этой ситуации разработка веб-приложений, которые способствуют эффективному взаимодействию работодателей и соискателей становится, становится всё более актуальной.

Новизной проекта является создание функционала, который позволит взаимодействовать не только между работодателем и соискателем, но и между любыми другими пользователями.

Целью курсового проекта является разработка веб-приложения «Поиск работы», которое позволит соискателям находить вакансии, а работодателям находить рабочих. Одной из основных задач заключается в создании интерфейса, который будет удобен и понятен для пользователя и позволит отобразить всю необходимую информацию.

Основные задачи курсового проекта включают:

* Разработка базы данных, которая будет хранить пользователей, вакансии, компании, отклики;
* Создание пользовательского интерфейса, который обеспечит удобный и интуитивно понятный доступ к данным.
* Разработка программного интерфейса для загрузки и получения актуальных данных;
* Внедрение функционала, позволяющего пользователю взаимодействовать с программным интерфейсом через пользовательских интерфейс.

В ходе выполнения курсового проекта будут применяться современные технологии, методы разработки веб-приложений и инструменты программирования. При этом ориентируясь на лучшие практики в области пользовательского интерфейса, чтобы обеспечить комфортное взаимодействие пользователя с интерфейсом, а также применяемых технологиях и инструментах.

1. **Постановка задачи**

Главной задачей курсового проекта является разработка веб-приложения, позволяющее пользователю создавать компания, просматриваться и откликаться на вакансии, связываться с работодателем благодаря чату. Пользователь, создавший компанию, в свою очередь способен создавать и удалять вакансии, приглашать других пользователей в свою кампанию, изменить статус откликов, удалять пользователей из собственной компании. В данной курсовой работе требовалось реализовать следующие задачи:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли работодателя и кандидата
* обеспечивать возможность создания компании;
* обеспечивать возможность добавления вакансии в избранное;
* обеспечивать возможность поиска вакансий по названию;
* обеспечивать возможность создания новых вакансий;
* обеспечивать возможность отклика на вакансии;
* обеспечивать возможность взаимодействия работодателя и нанимателя при помощи чата.

**1.1 Алгоритмы решения**

При разработке курсового проекта использовались следующие технологии: Node.js, фреймворк NestJS, ORM prisma, Socket.IO для back-end части. Angular, Bootstrap, Socket.IO-client, RxJS для front-end части. В качестве базы данных использовалась PostgreSQL.

Node.js [1] – это среда выполнения JavaScript на стороне сервера, построенная на движке V8. Он позволяет использовать JavaScript на сервере вне браузера. В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

NestJS [2] – фреймворк для создания эффективных масштабируемых серверных приложений, разработки серверных приложений на Node.js. В нем используется прогрессивный JavaScript, он разработан на TypeScript и полностью его поддерживает.

Prisma [3] – современный ORM для Node.js, позволяющий работать с базами данных без использования SQL.

Socket.IO [4] – библиотека JavaScript, основанная на веб-сокетах. Преимущественно необходима при написании приложений, в которых обмен данных происходит в реальном времени.

Bootstrap [5] – это открытый и бесплатный HTML -, CSS - и JS-фреймворк, который используют веб-разработчики для быстрой верстки адаптивных дизайнов сайтов и веб-приложений. Включает в себя CSS - и HTML-шаблоны оформления для веб-форм, меток, типографики, кнопок, блоков навигации и других компонентов веб-интерфейса.

Angular [6] – фреймворк для написания одностраничных приложений, написанных на TypeScript, JavaScript.

Socket.IO-client – это библиотека JavaScript клиента для Socket.IO.

Swagger [7] – это набор инструментов и спецификаций для создания, описания и визуализации веб-сервисов API. Он предоставляет унифицированный подход к проектированию, разработке и документированию API, что упрощает их использование и интеграцию.

RxJS [8] – это библиотека для реактивного программирования, которая позволит удобно организовать работу с событиями и асинхронным кодом, писать сложную логику декларативно.

**1.2 Обзор прототипов**

Для разработки веб-приложения необходимо провести сравнительных анализ существующих веб-приложений, выделив их плюсы и минусы.

Сайт rabota.by:

* Плюсы:
  1. Интеграция возможности входа с различными сервисами;
  2. Вход по номеру телефона;
  3. Персонализированные рекомендации по вакансиям;
  4. Интуитивно понятный интерфейс.
* Минусы:
  1. Нет возможности просматривать последние чаты;
  2. Нет возможности написать работодателю напрямую.

Сайт ares.by:

* Плюсы:

1. Множество фильтров для поиска вакансии.

* Минусы:
  1. Неудобная система навигации;
  2. Некоторые из ссылок бесконечно загружаются или перенаправляют на главную страницу без какого-либо уведомления.

Сайт vakanto.by:

* Плюсы:
  1. Имеет интеграцию с другими сайтами по поиску работы
* Минусы:
  1. Использует незащищённое соединение
  2. Имеет сложный в навигации интерфейс

После анализа нескольких сайтов было принято решение выбрать сайт rabota.by для дальнейшей разработки прототипа «Поиск работы». Этот сайт имеет функционал, превосходящий другие аналоги.

# **2 Проектирование web-приложения**

Проектирование архитектуры проекта – одна из самых сложных и важных задач в процессе разработки приложения, так как в зависимости от проектирования архитектуры определяется как уровень зависимости между компонентами в приложении, так и расширяемость отдельных его модулей.

Архитектура проекта – общая структура и представления приложения как системы, состоящих из некоторой совокупности взаимодействующих между собой подсистем. Разработка архитектуры – одно из первых, с чем сталкивается разработчик при

Основные задачи проектирования архитектуры проекта:

* Выделением в системе отдельных компонентов, слоёв и модулей, сгруппировав логически связанную функциональность;
* Определение способов взаимодействие между выделенными компонентами, слоями, модулями.

Для разработки проекта было принято решение использовать стандартную трёхуровневую архитектуру, так как такая архитектура позволяет добиться низкого уровня связанности, что в свою очередь обеспечивает приложению хорошую масштабируемость, надёжность и простоту поддержки.

**2.1 Обобщённая архитектура**

Приложение было разработано на клиент-серверной архитектуре, которая представляет собой распределённую модель, в которой клиентское приложение и серверное приложение взаимодействуют между собой для обмена данными и выполнения задач.

В такой архитектуре клиент является пользовательским интерфейсом или приложение, запущенное на устройстве пользователя. Клиент отправляет запросы на сервер, получая от него ответ и отображает информацию пользователю.

Сервер – некоторое устройство, которое принимает запросы от клиентов, обрабатывает их и возвращает соответствующие ответы. Сервер обычно отвечает за обработку бизнес-логики, доступ к базе данных, выполнение вычислений и другие операции, необходимые для обработки запросов клиентов.

Клиент и сервер взаимодействуют посредством HTTP протокола. Клиент отправляет HTTP-запрос, сервер его обрабатывает и возвращает HTTP-ответ. Архитектура предоставлена на рисунке 2.1.

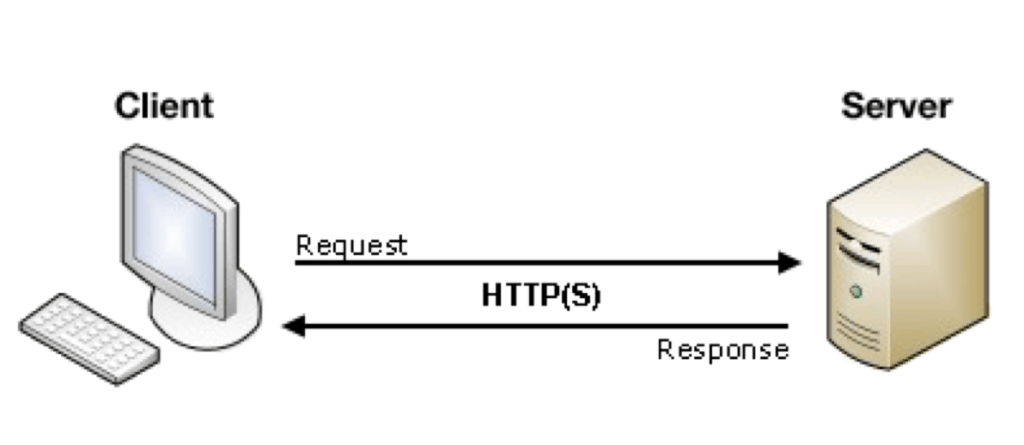


Рисунок 2.1 – Общая архитектура

Клиент-серверная архитектура позволяет разделять ответственность между клиентом и сервером. Клиент отвечает за пользовательский интерфейс, взаимодействие с пользователем и отображение данных, в то время как сервер отвечает за бизнес-логику, обработку запросов и управление данными. Это позволяет создавать масштабируемые и гибкие приложения, где клиенты и серверы могут быть разработаны и масштабированы независимо друг от друга.

**2.2 Серверная часть приложения**

При проектировании серверной части необходимо поддерживать низкий уровень избыточности, связи между составляющими частями, а также проверять корректное выполнение кода, так как от этого будет зависеть работа всего приложения, а не только серверной части. Добиться это можно благодаря распределению логики между различными слоями.

Для того, чтобы обмениваться данными между клиентами и сервисами был выбран архитектурный стиль REST. По своему определению он широко использует возможности протокола HTTP и поэтому не имеет состояния (stateless). Это современный подход, использующийся в большинстве веб-сервисов и поэтому имеющий множество примеров реализаций различных функций на его основе, таких как аутентификация, выполнение CRUD-операций и т.д. В REST-приложениях в подавляющем большинстве случаев используется JSON для передачи данных, его также было решено взять как формат данных при передаче между клиентами и сервером.

Учтя во внимание вышеперечисленно, правильным решением будет разбиться серверную логику на три слоя: контроллеры, сервисы, репозитории.

Контроллеры – Верхний уровень сервера, на котором располагают классы-контроллеры, которые отвечают за обработку входящих запросов и отправки ответов клиентам.

Сервисы – Слой, содержащий классы, отвечающие за бизнес-логику, среди них могут быть как и те, кто просто делегирует вызов репозиторию, так и те, который содержат в себе сложную логику.

Репозитории – Слой, содержащий в себе классы, отвечающий операции с базой данных.

**2.3 Клиентская часть приложения**

Клиентская часть приложения реализована в виде одностраничного приложения. Это означает, что приложение использует единственный HTML-документ в качестве оболочки для всех веб-страниц. Вместо полной перезагрузки страницы при каждом взаимодействии с пользователем, фреймворк динамически подгружает/создаёт HTML, CSS и JavaScript, основываясь на информации, хранящейся внутри каждого компонента.

Такой подход создавать высокопроизводительные приложения, так как меньше данных передаётся по сети и меньше ресурсов требуется для обновления интерфейса.

В проекте, для управления состоянием страницы и отдельных данных использовались сервисы в совокупности с компонентами, которые позволяют как хранить состояние клиента во время его пребывания на сайте, так и сохранять некоторую информацию внутри браузера.

Для обращения к серверной части, клиентская часть использует отдельный HTTP модуль, который уже встроен в фреймворк и нуждается только в подключении в соответствующем месте.

**2.4 Безопасность. Аутентификация и авторизация**

Важнейшим аспектом проектирования серверной части приложения является то, как пользователи будут входить в систему и как система будет понимать, что очередной присланный запрос прислан уже находящимся в системе пользователем.

Стандартными подходами при реализации этих функций является либо аутентификация с помощью логина и пароля, при этом используя собственную БД в качестве их места хранения, либо использование сторонних сервисов, предоставляющих своё API для того, чтобы можно было осуществить вход. Это может быть API на основе протокола OAuth, либо какие-то собственные разработки. При рассмотрении данного вопроса лучше всего обратить внимание на то, какими будут приложение-клиенты, то, в каком окружении, на каких системах они будут запускаться. В нашем случае это приложение под веб-клиент. В связи с этим было принято решение о авторизации с использованием JWT-токена.

После того, как пользователь совершит вход в приложение, необходимо поддерживать его статус и позволять выполнять запросы к API, в то время как запросы не аутентифицированных пользователей – отклонять. Как было описано ранее, серверная часть работает без сохранения состояния по протоколу HTTP, в таком случае достаточно распространённой практикой является использование JWT-токена. Он идеально вписывается в описанные методы взаимодействия с клиентами и поможет поддержать требуемый уровень безопасности.

**2.5 Общее описание работы**

В решении курсового проекта была использована REST архитектура. Рассмотрим компоненты, используемые в данной архитектуре.

Клиент – это программа, использующая API. Клиент делает запросы к API, чтобы получить некоторую информацию или что-то изменить в приложении. Веб-браузер является клиентом - он взаимодействует с API-интерфейсом, чтобы получить от него содержимое страницы. Запрошенная информация отправляется обратно в браузер и отображается на экране.

Ресурс – это любая информация, которую API может предоставить клиенту. Например, ресурсом в данном API может быть блог, фотография или комментарий. Каждый ресурс имеет уникальное имя, называемое идентификатором ресурса.

Сервер используется приложением, которое принимает запросы клиентов и содержит ресурсы, которые требуются клиенту. Сервер имеет API для взаимодействия с клиентами, не предоставляя им прямого доступа к контенту, хранящемуся в базе данных.

При разработке приложения будет использована N-Tier архитектура, которая позволяет добиться модульного разделения пользовательского интерфейса, бизнес-логики и хранением данных. Данные курсовой проект будет построен на основе примера ниже приведенной трёхуровневой архитектуры на рисунке 2.2.

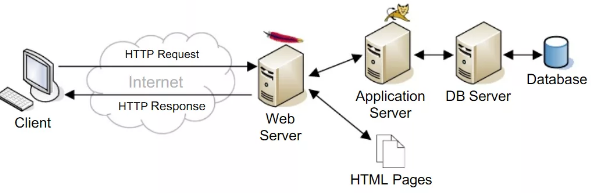


Рисунок 2.2 – Общая схема N-Tier архитектуры

Клиент (уровень представления) – представляет собой генерирующуюся на веб-сервере разметку или получаемые с уровня бизнес-логики данные для REST API в формате JSON, преобразующиеся впоследствии в разметку на стороне клиента.

Application Server (уровень бизнес-логики) – при взаимодействии с уровнем сервисов является посредником между сущностями базы данных и моделями UI. Сочетает в себе большую часть реализации всего функционала приложения.

DB Server (уровень доступа к данным) – содержит модели, соответствующие сущностям базы данных.

**2.6 Диаграмма вариантов использования**

Диаграмма UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта. Варианты использования могут быть связаны с участвующими в них действующими лицами (actors), изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: обобщением (generalization), расширением (extend relationship) и включением (include relationship).

Для каждой роли были разработаны UML-диаграммы вариантов использования. При первом открытии сайта пользователь не аутентифицирован. Диаграмма вариантов использование неаутентифицированного пользователя отображена на рисунке 2.3.



Рисунок 2.3 – Диаграмма использования для неаутентифицированного пользователя

Из диаграммы видно, что неаутентифицированный пользователь может либо войти под уже существующими данными либо зарегистрироваться.

После регистрации либо входа в аккаунт пользователь получает роль аутентифицированного пользователя, а также роль соискателя. Диаграмма вариантов использования приложения для роли «соискатель» на рисунке 2.4.

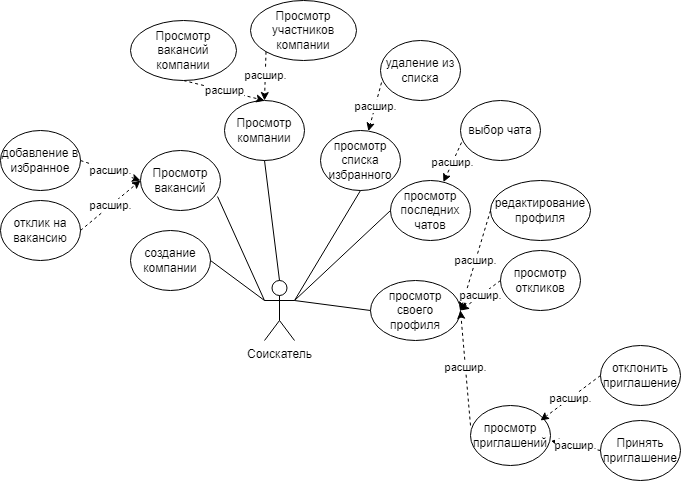


Рисунок 2.4 – Диаграмма использования для соискателя

На диаграмме видно, что соискатель может создать компанию, после чего он получит роль «работодателя». Диаграмма вариантов использования приложения для роли «работодателя» отображена на рисунке 2.5.



Рисунок 2.5 – Диаграмма использования для работодателя

**2.7 Проектирование базы данных**

При проектировании базы данных были разработаны следующие таблицы: account, account\_info, company, employer, favorite, message, reply, token, vacancy, invite.

Диаграмма таблиц базы данных предоставлена в приложении А.

Таблица account содержит данные о каждом пользователе. Охарактеризуем каждое поле таблицы account:

* id – идентификатор пользователя
* login – логин для входа
* password\_hash – захешированный пароль

Таблица account\_info содержит личную информацию пользователя, отличаясь от account тем, что содержит только информацию, без внешних ключей на другие таблицы. Охарактеризуем каждое поле таблицы account\_info:

* id – идентификатор пользователя
* name – имя пользователя
* webstie – вебсайт пользователя
* about – информация о пользователе

Таблица company содержит данные о компании. Охарактеризуем каждое поле таблицы company:

* id – идентификатор компании
* owner\_id – идентификатор пользователя, создавшего компанию
* name – название компании
* about – описание компании
* website – весбайт компании
* phone – основной телефон компании

Таблица employer содержит информацию об работниках компании. Охарактеризуем каждое поле таблицы employer:

* account\_id – идентификатор пользователя
* company\_id – идентификатор компании
* is\_hr – является ли работник HR

Таблица favorite содержит информацию о вакансиях, добавленных в избранное пользователем

* id – идентификатор
* account\_id – идентификатор пользователя
* vacancy\_id – идентификатор вакансии
* date – время добавления в избранное

Таблица message содержит в себе сообщение, пересылаем от пользователя к пользователю. Охарактеризуем каждое поле таблицы message:

* id – идентификатор сообщения
* from\_id – идентификатор пользователя, который отправил сообщение
* to\_id – идентификатор пользователя, которому было отправлено сообщение
* text – контент сообщения
* date – время отправки сообщения

Таблица reply содержит в себе информацию об откликах на вакансии. Охарактеризуем каждое поле таблицы reply:

* id – идентификатор отклика
* vacancy\_id – идентификатор вакансии
* account\_id – идентификатор пользователя
* date – время добавления отклика

Таблица token содержит информацию о refresh-токенах. Охарактеризуем каждое поле таблицы token:

* id – идентификатор токена
* account\_id – идентификатор пользователя
* token – refresh-токен
* date – время истечение токена

Таблица vcancy содержит информацию он вакансиях. Охарактеризуем каждое поле таблицы vacancy:

* id – идентификатор вакансии
* owner\_id – идентификатор сотрудника, создавшего вакансию
* company\_id – идентификатор компании
* title – заголовок вакансии
* description – описание вакансии
* salary – предлагаемая зарплата
* post\_date – время публикации вакансии
* is\_active – открытость/закрытость вакансии

Таблица invite содержит в себе приглашения пользователей в компании. Охарактеризуем каждое поле таблицы invite:

* id – идентификатор приглашения
* account\_id – идентификатор пользователя, которого пригласили
* company\_id – идентификатор компании
* inviter\_id – идентификатор сотрудника, который пригласил в компании
* date – время отправки приглашения

**3 Разработка web-приложения**

Приложение делится на две части сервер и клиент. Серверная часть написана на NestJS, и клиентская написана на Angular. Серверная часть взаимодействует с базой данных. Клиентская часть приложения взаимодействует с сервером через REST API.

**3.1 Серверная часть**

Для разработки серверной части приложения использовались следующие технологии:

* NestJS: фреймворк для написания веб-приложений. Дополнительно предоставляет необходимые пакеты для создания JWT, ORM моделей.
* пакет bcrypt для создания хэшей, соли и их сравнения;
* пакет socket.io для постоянной двухсторонней связи сервера и клиента;
* пакет class-validator для создания простой валидации объектов, передающихся в контроллеры через тело запроса;
* пакет swagger для написания документации.

Данные, необходимые для работы приложения, такие как секрет для JWT, порт, хост, строка для подключения к базе данных, содержатся в .env файле.

В классе main.ts происходит первоначальная настройка сервера: настройка ssl, cors, конфигурация Swagger, запуск сервера на данных, полученных с .env.

Классы-контроллеры являются представлением API. NestJS не предоставляет валидацию данных, приходящих в виде JSON, поэтому для каждого контроллера была созданы группы DTO и использован пакет class-validator для создания валидации для отдельных полей по различным параметрам.

Сервисы – классы, которым контроллеры делегируют выполнение различных действий, после поступления на них запроса.

Как было уже описано ранее, большинство сервисов лишь выполняют различные действия с модулями, иногда лишь делая небольшие преобразования в объектах, чтобы ORM-система смогла правильно их обработать. Работа сервисов совершается в неблокирующем асинхронном режиме.

Для взаимодействия с базой данных используется ORM prisma. Для это необходимо заранее создать схему, содержащую модели базы данных. Описание моделей происходит благодаря различным декораторам.

**3.2 JWT. Аутентификация и авторизация**

Как было написано ранее, для нужд аутентификации и авторизации будет использоваться JWT-токен. При аутентификации, он будет формироваться и отправляться клиенту, затем этот клиент должен будет его отправлять с каждым запросом в заголовке, а задача сервера – проверять этот токен на валидность. Сам токен состоит из трех частей, как показано на рисунке 3.1

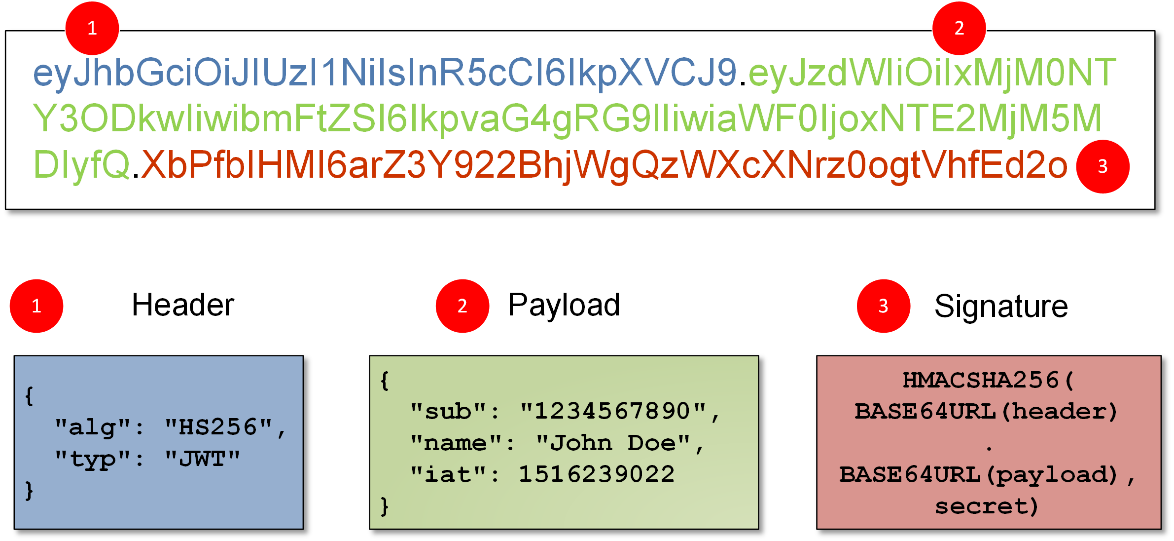


Рисунок 3.1 – Составные части JWT-токена

Для реализации аутентификации использовался пакет @nestjs/jwt для создания и валидации JWT-токена и Guard с canActivate методом для проверки токена, который можно использовать только в необходимых контроллерах.

**3.3 Обмен данными в реальном времени**

При разработке приложения использованы веб-сокеты для поддержки двусторонней связи.

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Для установления соединения WebSocket клиент и сервер используют протокол, похожий на HTTP. Клиент формирует особый HTTP-запрос, на который сервер отвечает определенным образом.

Пример запроса от клиента на сервер представлен на рисунке 3.2



Рисунок 3.2 – Запрос клиента для установки соединения WebSocket

Если сервер поддерживает такой протокол, то в ответ клиенту об этом, что протокол WebSocket будет использоваться для текущего соединения. Пример ответа можно увидеть на рисунке 3.3

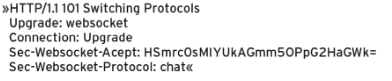


Рисунок 3.3 – ответ от сервера в случае успешной смены протокола

После установки соединения по данному протоколу начинается обмен между сервером и пользователем. Сервер пересылает сообщения и другие данные о коммуникации клиентского приложения и сервера.

В моём проекте для реализации чата был использован пакет socket.io, который позволяет облегчит и улучшить работу с WebSocket в сравнении с обычной реализацией.

Для работы чата был разработан gateway-класс, который обрабатывает отправку сообщений между пользователями. Установка соединения клиент-чат-сервер происходит следующим образом.

1. При успешной смене протокола, клиент отправляет запрос на установку соединения с чатом и конкретным пользователем, отправляя свой идентификатор и идентификатор пользователя, с которым начинает чат;
2. Сервер, получая идентификатор клиента, запоминает, какой клиент установил соединение и находится в данный момент в чате.

После установки соединения обработка сообщения происходит следующим образом:

1. Преобразование получаемого сообщения в объект, пригодный для дальнейшего использования. При не валидности объекта дальнейшая обработка сообщения прекращается и производится логирование неудачи;
2. Производится отправка сообщения для пользователя, который отправил сообщение, чтобы тот в свою очередь мог обновить список сообщений у себя;
3. Производится поиск среди пользователей, который сейчас подключены к чату и совпадают с идентификатором, который был передан в сообщении;
4. Производится отправка сообщения найденному пользователю;

Так же следует отметить, что алгоритм будет работать даже в том случае, если клиент имеет несколько соединений с различных окон браузеров.

## **3.4 разработки документации**

Swagger – это технология, которая позволяет документировать REST-сервисы. Она представляет интерфейс между клиентскими приложениями и сервером, рисунок 3.4.

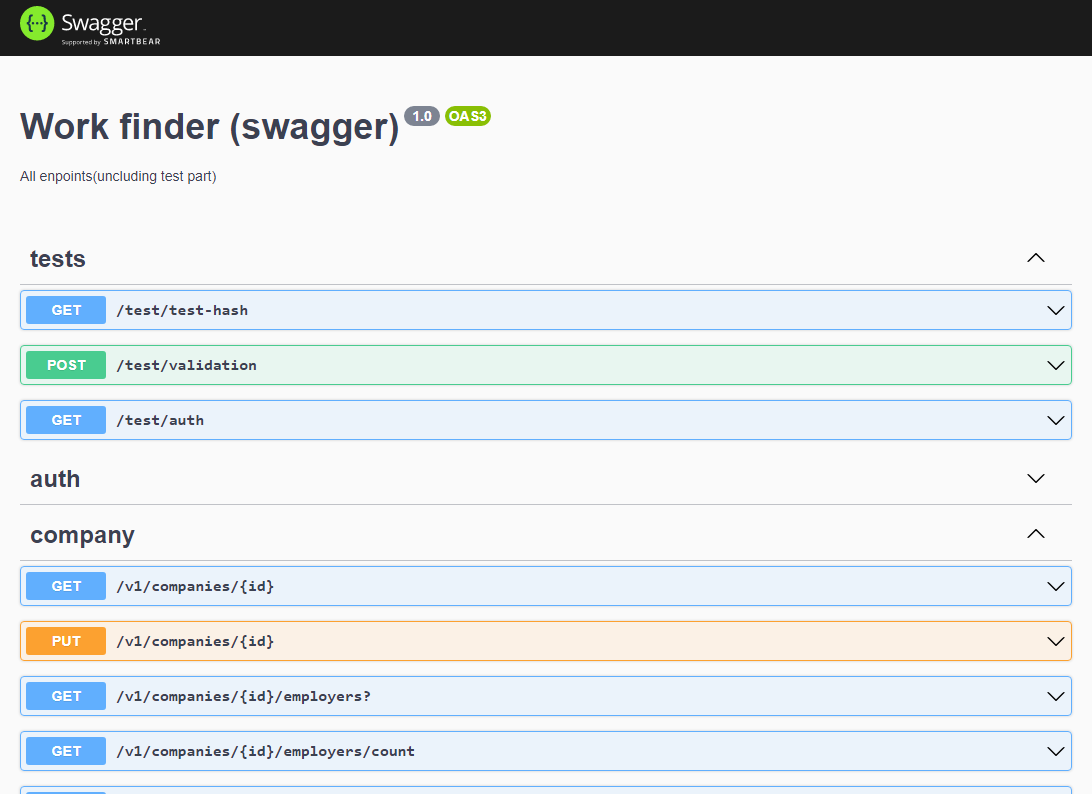


Рисунок 3.4 – Swagger-схема

Данная документация даст более лёгкий путь интеграции для других приложений, а также позволит упростить поддержку документации и погружения специалиста в данный проект.

## **3.5 Клиентская часть**

Для клиентской части приложения был использован фреймворк Angular. С помощью Angular создавать интерактивные пользовательские интерфейсы приятно и просто. Angular позволяет создавать компоненты, используя TypeScript и HTML, при это значения из компонента могут напрямую использоваться в разметке, а сами компоненты переиспользоваться в различных частях приложения.

Благодаря структурным директивам в Angular появляется возможность создавать страницы, в которых некоторые элементы не будут создаваться в DOM или будет создаваться только при выполнении некоторых условий, что даёт простой способ к созданию динамических страниц.

Из наиболее значимых частей клиентской части является класс-перехватчик, который обрабатывает каждый http-запрос, который выходит из приложения. В нашем случае существует два класса-перехватчика.

Первый отлавливает ошибки и преобразует их в удобную форму для использования в приложении в дальнейшем.

Второй, если клиент вошёл аккаунт, добавляет при каждом запросе авторизационный заголовок. В случае возвращения ответа со статусом 401 перехватчик действует по следующему принципу:

1. Удаление текущего access-токена;
2. Проверка, существует ли refresh-токен. Если нет, то происходит выход из аккаунта;
3. При существовании refresh-токена производится запрос к серверной части приложения для обновления пары токенов
4. В случае успеха производится повторная отправка последнего запроса.
5. В случае неудачи производится выход из аккаунта

## **3.6 Реализация базы данных**

В разрабатываемом программном средстве существует возможность добавления, изменения данных в базе данных, взаимодействие происходит через обращение к функциям моделей prisma.

Для хранения и управления данными в нашем приложении была выбрана база данных PostgreSQL.

PostgreSQL – это реляционная база данных с открытым кодом, является надежной, безопасной и расширяемой базой данных, а также располагает обширной экосистемой доступных средств, разработчики используют PostgreSQL в разнообразных сценариях. PostgreSQL совместим со всеми основными операционными системами, включая Linux, Windows и Macintosh.

В нашем приложении мы использовали PostgreSQL для хранения и обработки различных типов данных, таких как информация о пользователях, вакансиях, компаниях, откликах, приглашениях. Основные аспекты реализации базы данных в нашем приложении:

* Установка и настройка: для начала работы необходимо установить необходимые пакеты и настроить строку подключения для дальнейшей работы с базой данных. Строка подключения содержит информацию о хосте, порте, аутентификации и других параметров.
* Создание схемы и моделей. Для обращения к сущностям базы данных необходимо определить модели, которые описывают структуру и свойства таблиц в базе данных. Мы можем в первую очередь создать как базу данный и после на основе её создать модели, так и модели, на основе которых после смоделировать базу данных.
* Операции CRUD. Обращение к базе данных для модификации или выборки данных происходит благодаря моделям и функциям, которые предоставляет prisma.

Весь код описания сущностей базы данных предоставлен в приложении Б.

**3.7 Конечные точки**

Описание доступных запросов взаимодействия приведено в таблице 3.1. Для взаимодействия с частью запросов необходима авторизация. Токен должен содержаться в заголовках запросов.

Таблица 3.1 – Описания HTTP запросов

|  |  |  |
| --- | --- | --- |
| Адрес | Метод | Описание |
| /auth/register | POST | Запрос предназначен для создания пользователя |
| /auth/login | POST | Запрос предназначен для аутентификации |
| /auth/auth | POST | Запрос предназначен для обновления пары токенов |
| /companies/{id} | GET | Запрос предназначен для получения информации о компании |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| /companies/{id} | PUT | Запрос предназначен для обновления информации о компании |
|  |  |  |
| /companies/{id}/employers? | GET | Запрос предназначен для получения информации сотрудниках компании |
| /companies/{id}/vacancies? | GET | Запрос предназначен для получения информации о вакансиях компании |
| /companies/{id}/invites? | GET | Запрос предназначен для получения информации о приглашениях в компанию |
| /companies | POST | Запрос предназначен для создания компании |
| /companies/{id}/leave | POST | Запрос предназначен для выхода работника из компании |
| /vacancies/{id} | GET | Запрос предназначен для получения информации о вакансии |
| /vacancies/{id} | PUT | Запрос предназначен для обновления информации о вакансии |
| /vacancies/{id} | DELETE | Запрос предназначен для удаления вакансии |
| /vacancies? | GET | Запрос предназначен для получения информации о вакансиях |
| /vacancies/{id}/replies? | GET | Запрос предназначен для получения информации об откликах на вакансию |
| /vacancies | POST | Запрос предназначен для создания вакансии |
| /accounts/{id} | GET | Запрос предназначен для получения информации об аккаунте и информации о пользователе |
| /accounts/{id} | PUT | Запрос предназначен для обновления информации о пользователе |
| /accounts/{id}/invites | GET | Запрос предназначен для получения информации о приглашениях. Которые получил пользователь |
| /invites | POST | Запрос предназначен для отправки приглашения в компанию пользователю |
| /invites/{id}/accept | POST | Запрос предназначен для принятия приглашения пользователем в компанию |
| /invites/{id} | DELETE | Запрос предназначен для удаления приглашения |
| /messages | POST | Запрос предназначен для создания(отправки) сообщения |
| /messages? | GET | Запрос предназначен для получения сообщений между пользователями |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| /messages/chats? | GET | Запрос предназначен для получения информации о последних чатах |
| /replies? | GET | Запрос предназначен для получения информации пользователем о своих откликах |
| /replies? | POST | Запрос предназначен для отклика на вакансию |
| /replies/{id} | PUT | Запрос предназначен для изменения статуса отклика |
| /favorites? | POST | Запрос предназначен для добавления вакансии в избранное |
| /favorites? | DELETE | Запрос предназначен для удаления вакансии из избранного по идентификатору вакансии |
| /favorites/{id} | DELETE | Запрос предназначен для удаления из избранного |
| /favorites | GET | Запрос предназначен для получения пользователем информации об избранных вакансиях |
| /employers/{id}/kick | DELETE | Запрос предназначен для удаления сотрудников |
| /employers/{id}/update-status? | PUT | Запрос предназначен для обновления позиции сотрудника в компании |

Конечные точки (эндпоинты) приложения являются важным аспектом клиент-серверной архитектуры. Они представляют собой определенные URL-адреса на сервере, к которым клиенты могут обращаться для выполнения определенных операций или получения данных.

Показывая эндпоинты, мы видим описание доступных операций и ресурсов, которые клиенты могут использовать для взаимодействия с сервером. Каждый эндпоинт обычно связан с определенной функциональностью или сущностью в приложении.

Таким образом, предоставление эндпоинтов приложения позволяет ясно определить доступные операции и ресурсы, а также облегчает взаимодействие между клиентской и серверной частями приложения. Это важная составляющая разработки клиент-серверных приложений, которая помогает создать функциональное и эффективное приложениеа также позволяет более детально показать разделение эндпоинтов на сущности в веб-приложении.

# **4 Тестирование web-приложения**

В этой главе рассмотрим некоторые элементы интерфейса и протестируем веб-приложение.

**4.1 Форма регистрации**

Форма регистрации представляет собой четыре поля для ввода: имя пользователя, логин для входа, пароль и почта, три из них являются обязательными и отмечены специальным символом: «\*». После регистрации пользователь успешно войдёт в аккаунт и его перенаправит на главную страницу. Интерфейс формы регистрации можно увидеть на рисунке 4.1.

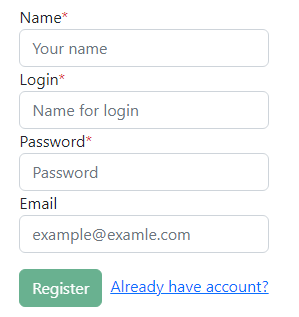


Рисунок 4.1 – Форма регистрации

При вводе логина, который уже используется другим пользователем, будет выведено уведомление о том, что логин уже занят, такой пример показан на рисунке 4.2.

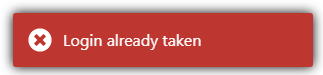


Рисунок 4.2 – Оповещение пользователя о занятом логине

При вводе данных, валидация которых не была предусмотрена, или критической ошибке на серверной части приложения пользователю так же будет выведено уведомления. Подобный подход используется во всём приложении. Пример такого оповещения приведён на рисунке 4.3

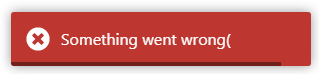


Рисунок 4.3 – Оповещение в непредвиденной ситуации

Большинство форм в приложении работают схожим образом, только некоторые отличаются наличием тех или иных валидаторов.

**4.2 Чат**

Чат собой представляет несколько объединённых компонентов, по левой стороне располагается список последних чатов с пользователями, справа непосредственно сам диалог. Пример этого показан на рисунке 4.4.

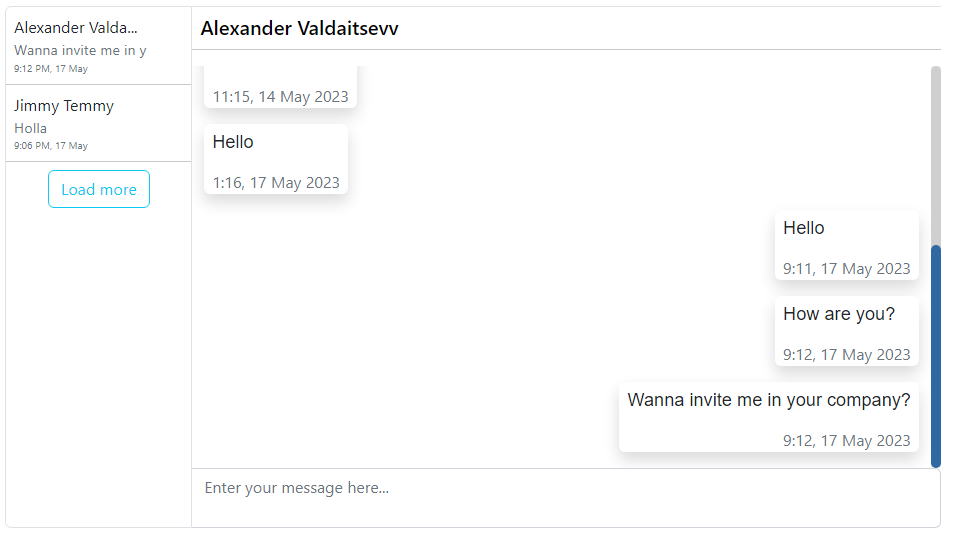


Рисунок 4.4 – Пример страницы чата

При получении или отправке сообщения происходит дополнительная передача сообщения через веб сокет. При условии того, что пользователь не является отправителем, ему приходит уведомление. Пример этого отображён на рисунке 4.5.

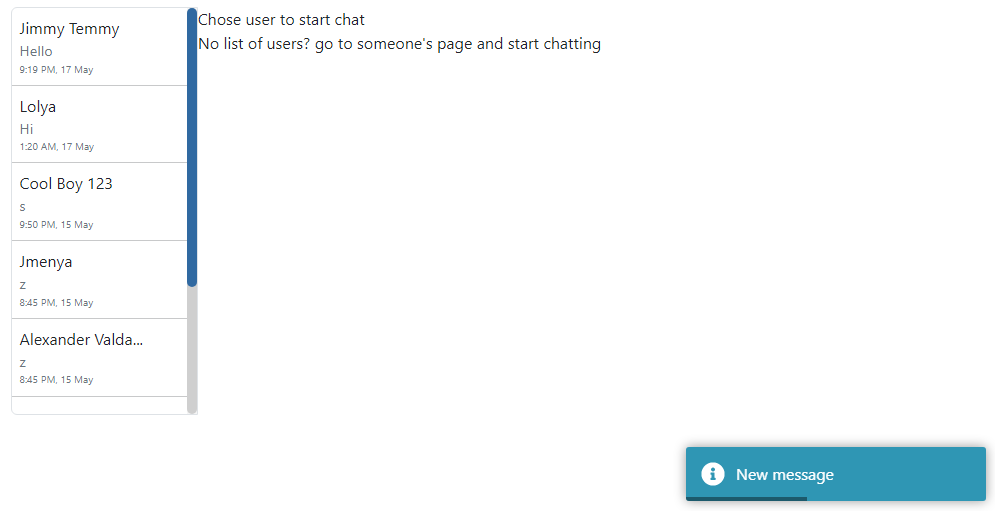


Рисунок 4.5 – Оповещение о новом сообщении

Ручное тестирование играет важную роль в обнаружении проблем и обеспечении качества приложения. Позволяет проверить поведение и взаимодействие различных компонентов в сборке, выявить проблемы с удобством использования и дизайном интерфейса, с которыми могут столкнуться реальные пользователи, выявить баги, которые могут проявиться только при ручном взаимодействии с клиентом, например, при вводе некорректных данных в формы. А также оценить скорость и стабильность работы клиента.

Таким образом, ручное тестирование позволяет выявить различные проблемы, с которыми может столкнуться реальный пользователь. Это делает ручное тестирование незаменимой частью тестирования веб-клиента.

Однако важно также учитывать, что ручное тестирование имеет свои ограничения, такие как ограниченная покрытие тестами и возможность пропустить некоторые проблемы. Для дополнительного обеспечения качества и улучшения процесса тестирования рекомендуется также внедрить автоматизированное тестирование.

По итогу, мы получаем приложение, в котором большая часть полей перед отправкой на сервер валидируются на клиентской стороне, также валидация присутствует на серверной стороне для защиты от прямых запросов.

**5 Руководство пользователя**

В данном разделе мы познакомимся с остальным функционалом доступным пользователя.

**5.1 Список вакансий**

Пользователь может получить список двумя способами, зайдя на главную страницу и воспользоваться поиском или же из навигационного меню.

На рисунке 5.1 показана главная страница и навигационное меню

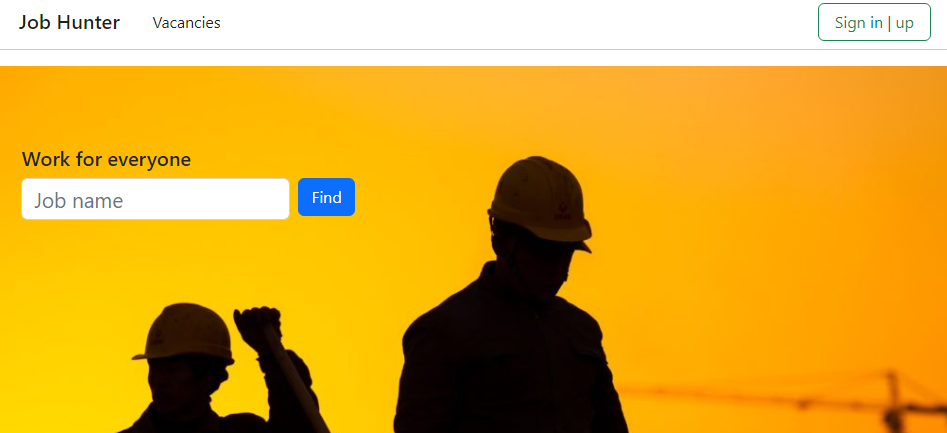


Рисунок 5.1 – Главная страница

В строку поиска можно ничего не вводить, в таком случае будут выведены все вакансии. На рисунке 5.2 показан пример отображения списка вакансий

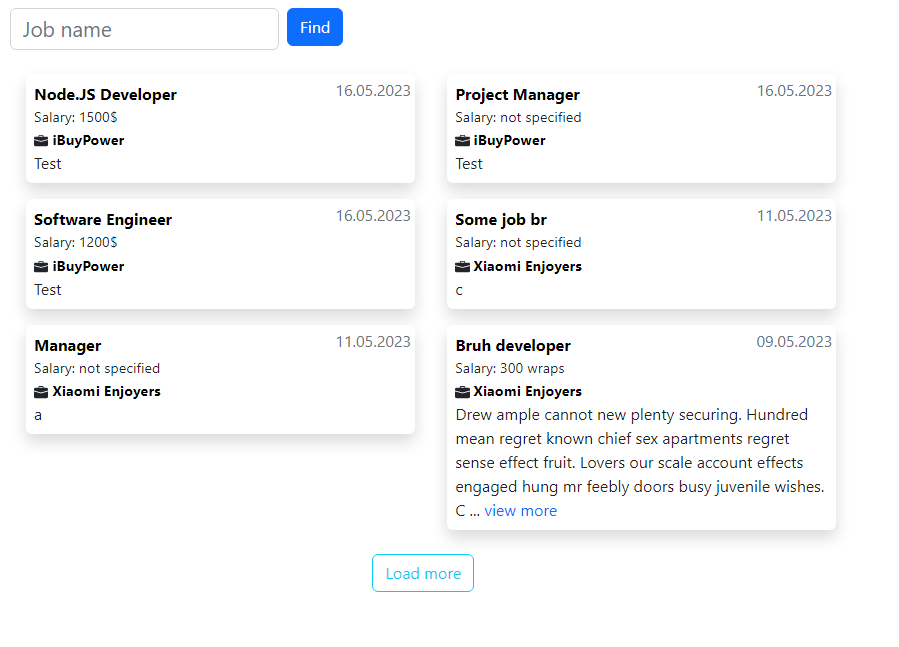


Рисунок 5.2 – Список вакансий

**5.2 Отклик на вакансию**

Перейдя на страницу вакансии, при условии того, что пользователь пошёл в аккаунт и не является членом компании, которая создала вакансию, пользователь может откликнуться на вакансию. Пример страницы отображён на рисунке 5.3

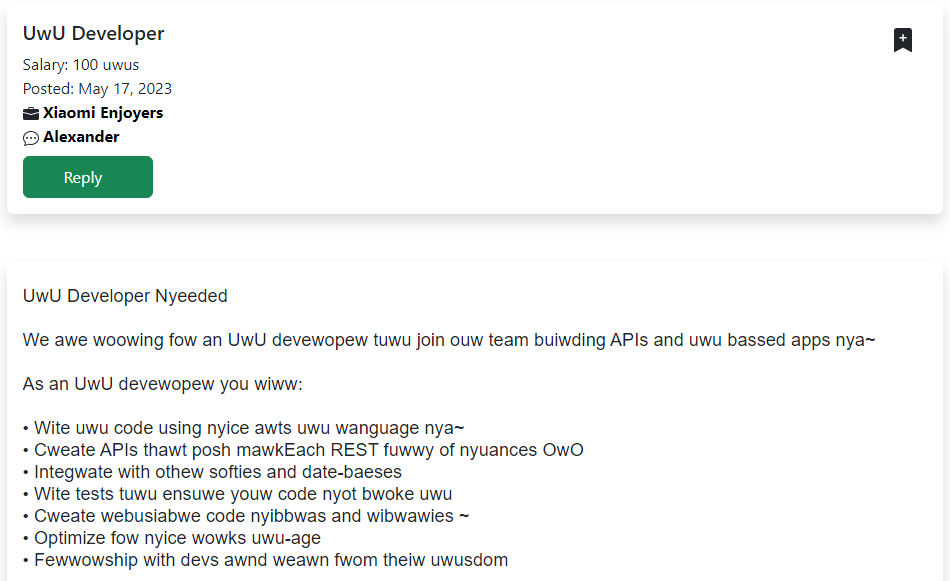


Рисунок 5.3 – Страница вакансии

Пример страницы, но уже со стороны работодателя изображен на рисунке 5.4.

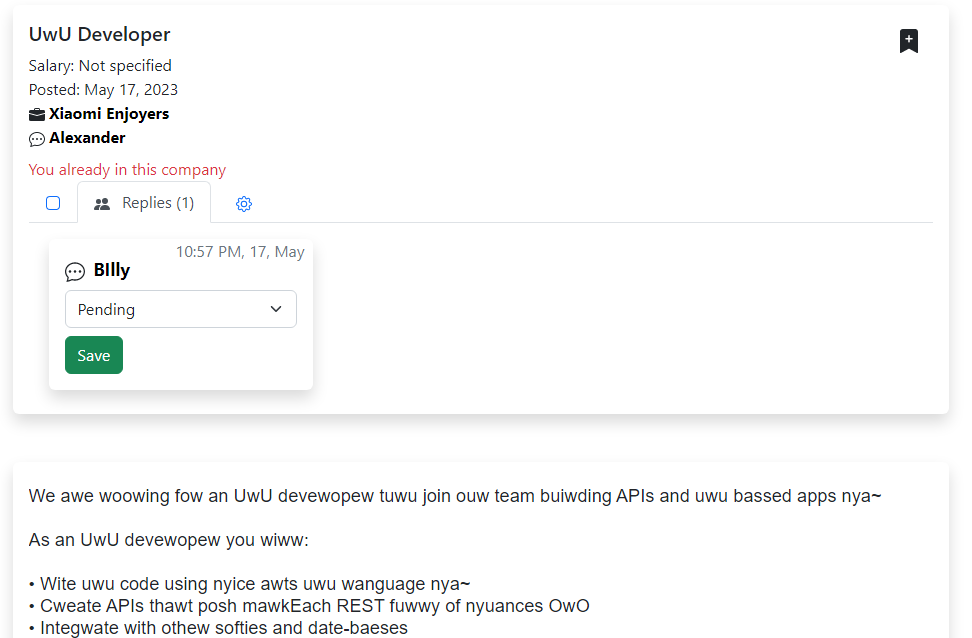


Рисунок 5.4 – Страница вакансии со стороны работодателя

**5.3 Добавление вакансии в избранное**

Зайдя на страницу с вакансией, авторизованный пользователь может добавить вакансию в избранное, нажав на соответствующую иконку сверху справа вакансии. Пример добавления вакансии в избранное приведён на рисунке 5.5.

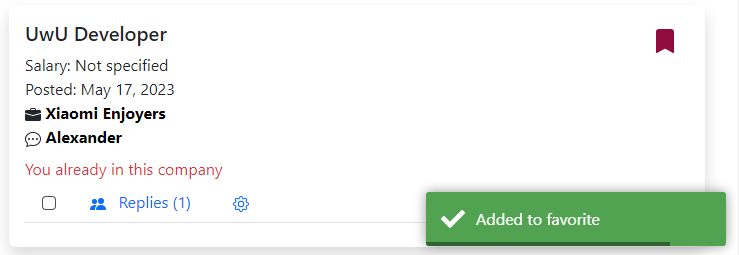


Рисунок 5.5 – Добавление вакансии в избранное

Перейти на страницу с списком избранных вакансий можно через выпадающее меню при нажатии на кнопку в правом верхнем углу. Пример такого изображен на рисунке 5.6.

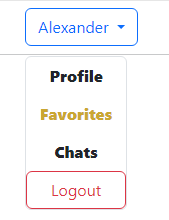


Рисунок 5.6 – Выпадающее меню

Перейдя на страницу со списком избранных вакансий мы просмотреть список вакансий, а так же удалить вакансию из избранного. Пример списка избранных вакансий отображен на рисунке 5.7.

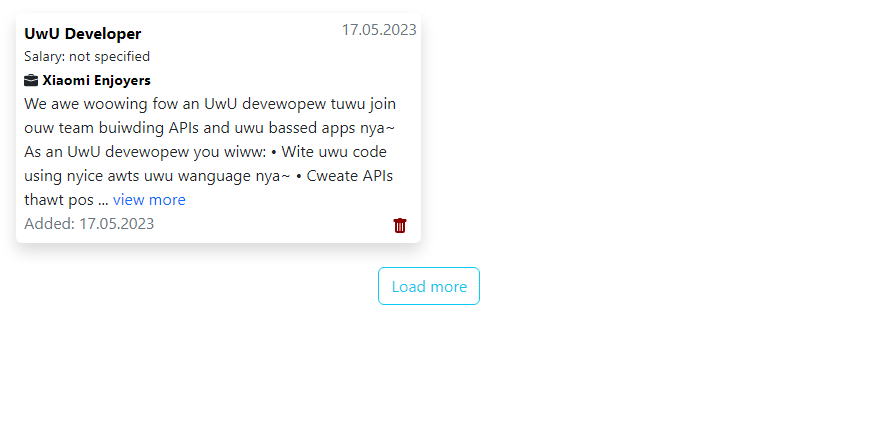


Рисунок 5.7 – Список избранных вакансий

**5.4 Создание компании**

Любой авторизованный пользователь, не являющийся членом какой-либо компании, может создать свою собственную. Для первоначального создания компании необходимо указать только имя компании. Форма создания компании отображена на рисунке 5.8.

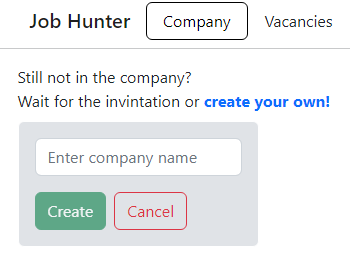


Рисунок 5.8 – Форма создания компании

После создания компании мы можем перейти на страницу компании из навигационного меню. Пример страницы компании отображён на рисунке 5.9.

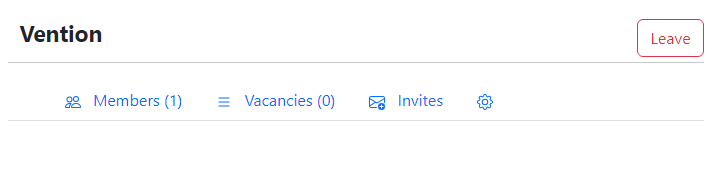


Рисунок 5.9 – Страница компании

Как мы видим, изначально на странице компании нет практически никакой информации, но мы можем изменить информацию нажав на иноку шестеренки. Пример формы для изменения компании изображен на рисунке 5.10.

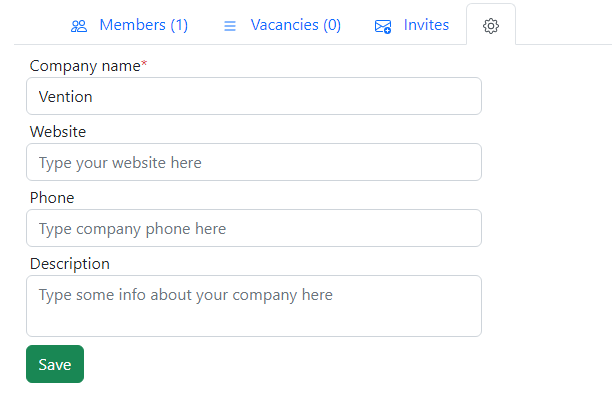


Рисунок 5.10 – Форма изменения информации о компании

Пример страницы, после заполнения необходимых полей отображён на рисунке 5.11.

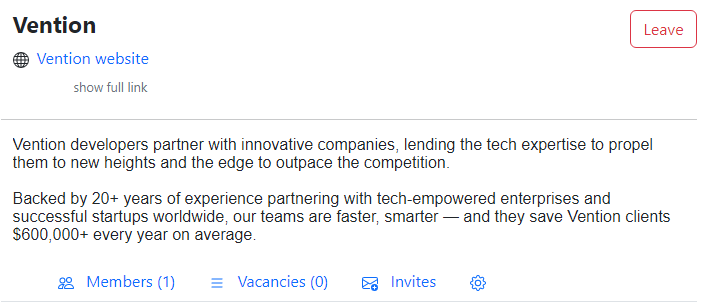


Рисунок 5.11 – Страница компании после заполнения полей

**Заключение**

При работе над курсовым проектом было проведено изучение предметной области, выявление в ней проблем и составление функционала программного средства, с их учётом, после чего последовало проектирование и разработка. Также в ходе работы проанализировано и проработано множество сторон и нюансов проектирования и разработки серверной части приложения и клиентской, например, выбор подходящих паттернов и технологий, составление и тестирование алгоритмов, структурирование проекта, построение пользовательского интерфейса.

Было получено большое количество опыта по работе с такими фреймворками как Angular и NestJS, базой данных PostgreSQL, ORM библиотекой prisma, реактивной библиотекой RxJS. Были улучшены навыки по проектированию, а также применению и связыванию друг с другом различных технологий. Также было написано руководство пользователя для веб-приложения.

Для повышения безопасности передачи данных в приложении используется HTTPS. В приложении были использованы сокеты и асинхронные запросы. Была подключена спецификация Swagger, с помощью которой были описаны точки доступа к ресурсам.

При разработке был реализован следующий функционал:

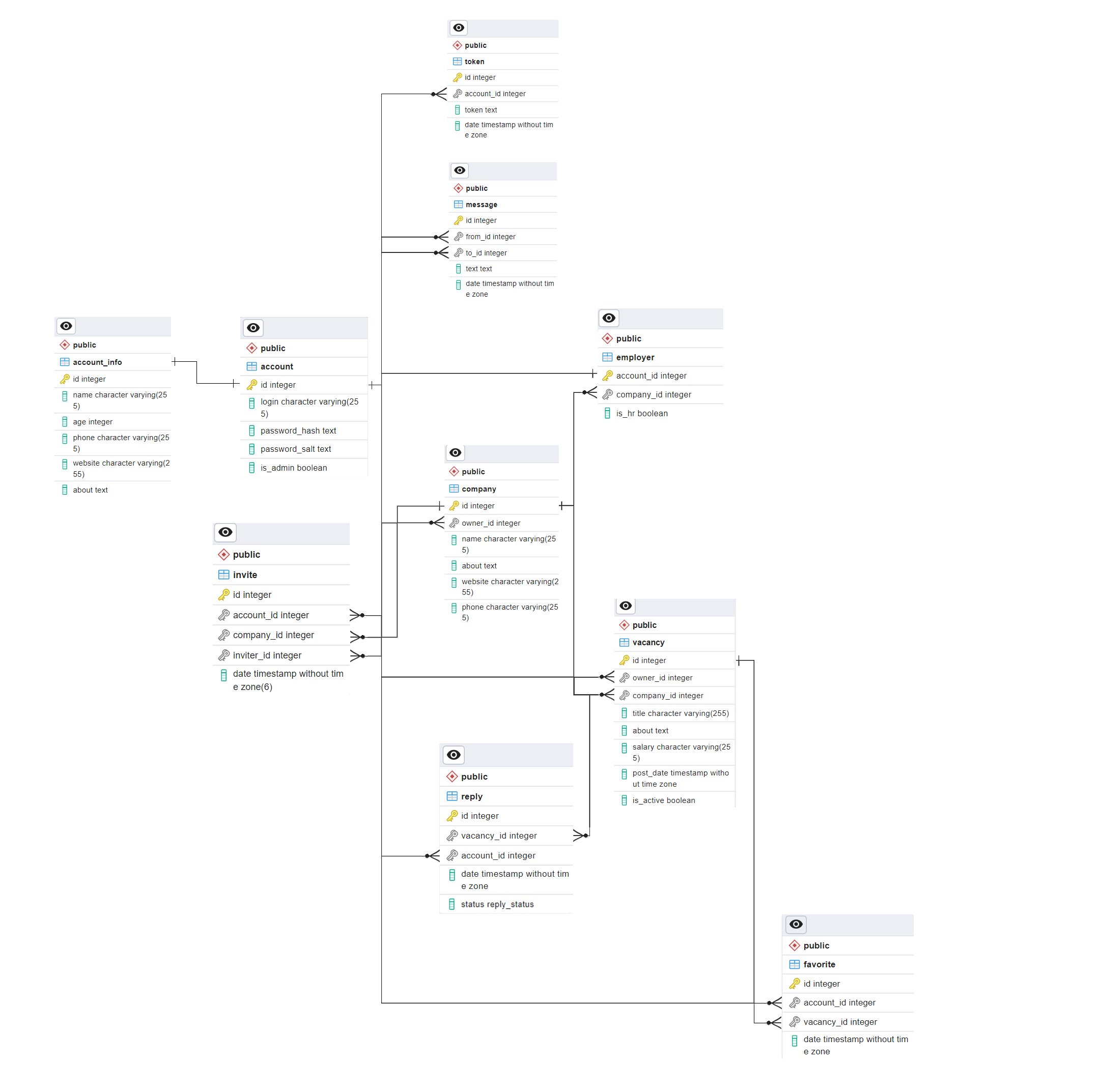
* обеспечивать возможность регистрации и авторизации;
* поддерживать роли работодателя и кандидата
* обеспечивать возможность создания компании;
* обеспечивать возможность добавления вакансии в избранное;
* обеспечивать возможность поиска вакансий по названию;
* обеспечивать возможность создания новых вакансий;
* обеспечивать возможность отклика на вакансии;
* обеспечивать возможность взаимодействия работодателя и нанимателя при помощи чата.

В итоге были реализован весь заявленный функционал. Как клиентская, так и серверная части проекта имеют хороший потенциал для будущих модификаций во многом благодаря грамотному структурированию, внимательности при написании кода, а также выборе известных и проверенных сопутствующих технологий.

**Список используемых источников**

1. Node.js [Электронный ресурс] – Режим доступа: https://nodejs.org – Дата доступа: 01.05.2023
2. NestJS [Электронный ресурс] – Режим доступа: https://docs.nestjs.com/ – Дата доступа: 05.05.2023
3. Prisma [Электронный ресурс] – Режим доступа: https://www.prisma.io/ – Дата доступа: 17.05.2023
4. Документация по Socket.IO [Электронный ресурс] – Режим доступа: https://socket.io/ – Дата доступа: 01.05.2023
5. Документация Bootstrap [Электронный ресурс]. — Режим доступа: https://getbootstrap.com/docs/5.3 Дата доступа: 09.05.2023
6. Документация по Angular [Электронный ресурс]. — Режим доступа: https://angular.io/ – Дата доступа: 15.05.2023
7. Документация по Swagger [Электронный ресурс]. — Режим доступа: https://swagger.io/ – Дата доступа: 01.05.2023
8. Документация RxJS [Электронный ресурс]. — Режим доступа: https://rxjs.dev/– Дата доступа: 03.05.2023

# **Приложение А**

****

# **Приложение Б**

model account {

id Int @id @default(autoincrement())

login String @db.VarChar(255)

password\_hash String

is\_admin Boolean @default(false)

accountInfo account\_info?

company company[]

employer employer?

favorite favorite[]

messageFrom message[] @relation("message\_from\_idToaccount")

messageTo message[] @relation("message\_to\_idToaccount")

reply reply[]

token token[]

invites invite[]

}

model account\_info {

id Int @id @default(autoincrement())

name String @db.VarChar(255)

email String? @db.VarChar(255)

website String? @db.VarChar(255)

about String?

account account @relation(fields: [id], references: [id],

onDelete: Cascade, onUpdate: NoAction)

}

model company {

id Int @id @default(autoincrement())

owner\_id Int?

name String @db.VarChar(255)

about String?

website String? @db.VarChar(255)

phone String? @db.VarChar(255)

owner account? @relation(fields: [owner\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

employers employer[]

vacancies vacancy[]

invites invite[]

}

model employer {

account\_id Int @id

company\_id Int?

is\_hr Boolean @default(false)

**Продолжение приложения Б**

account account @relation(fields: [account\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

company company? @relation(fields: [company\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

invites invite[]

vacancy vacancy[]

}

model favorite {

id Int @id @default(autoincrement())

account\_id Int?

vacancy\_id Int?

date DateTime @default(now()) @db.Timestamp(6)

account account? @relation(fields: [account\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

vacancy vacancy? @relation(fields: [vacancy\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

}

model message {

id Int @id @default(autoincrement())

from\_id Int?

to\_id Int?

text String

date DateTime @default(now()) @db.Timestamp(6)

accountFrom account? @relation("message\_from\_idToaccount", fields: [from\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

accountTo account? @relation("message\_to\_idToaccount", fields: [to\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

}

model reply {

id Int @id @default(autoincrement())

vacancy\_id Int?

account\_id Int?

date DateTime @default(now()) @db.Timestamp(6)

account account? @relation(fields: [account\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

status reply\_status

vacancy vacancy? @relation(fields: [vacancy\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

}

**Продолжение приложения Б**

enum reply\_status {

pending

accepted

declined

}

model token {

id Int @id @default(autoincrement())

account\_id Int

token String

date DateTime @db.Timestamp(6)

account account? @relation(fields: [account\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

}

model vacancy {

id Int @id @default(autoincrement())

owner\_id Int?

company\_id Int?

title String @db.VarChar(255)

description String? @db.VarChar(10000)

salary String? @db.VarChar(255)

post\_date DateTime @default(now()) @db.Timestamp(6)

is\_active Boolean @default(true)

favorite favorite[]

reply reply[]

company company? @relation(fields: [company\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

author employer? @relation(fields: [owner\_id], references: [account\_id], onDelete: Cascade, onUpdate: NoAction)

}

model invite {

id Int @id @default(autoincrement())

account\_id Int?

company\_id Int?

inviter\_id Int?

date DateTime @default(now()) @db.Timestamp(6)

account account? @relation(fields: [account\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

inviter employer? @relation(fields: [inviter\_id], references: [account\_id], onDelete: Cascade, onUpdate: NoAction)

company company? @relation(fields: [company\_id], references: [id], onDelete: Cascade, onUpdate: NoAction)}