Содержание

Введение ... 5
1 Постановка задачи и обзор аналогичных решений ... 6
1.1 Jobs Redefined ... 6
1.2 Headhunter ... 7
1.3 Вахтовик ... 9
1.4 Вывод по разделу ... 10
2 Проектирование web-приложения ... 11
2.1 Диаграмма вариантов использования ... 11
2.2 Проектирование базы данных ... 14
2.2.1 Логическая схема базы данных ... 14
2.2.2 Коллекции базы данных ... 14
2.2.3 Описание полей ... 15
2.2.4 Описание связей ... 19
2.3 Развертывание веб-приложения ... 20
2.4 Вывод по разделу ... 21
3 Разработка web-приложения ... 22
3.1 Программная платформа Node.js ... 22
3.2 База данных MongoDB ... 22
3.3 ODM Mongoose ... 22
3.3.1 Схемы в mongoose ... 22
3.4 Программные библиотеки ... 36
3.5 Использование сторонних сервисов ... 37
3.6 Применяемые паттерны ... 37
3.7 Скрипт для создания и заполнения базы данных ... 37
3.8 Реализация функций серверной части приложения ... 37
3.8.1 Функции пользователя с ролью «Гость» ... 37
3.8.1.1 Функция «Авторизация» ... 37
3.8.1.2 Функция «Регистрация» ... 40
3.8.2 Функции пользователя с ролью «Работодатель» ... 41
3.8.2.1 Функция «Запрос на присоединение к компании» ... 41
3.8.2.2 Функция «Отмена запроса на присоединение к компании» ... 43
3.8.2.3 Функция «Создание вакансии» ... 43
3.8.2.4 Функция «Удаление вакансии» ... 44
3.8.2.5 Функция «Изменение вакансии» ... 45
3.8.2.6 Функция «Одобрение откликов пользователей» ... 46
3.8.2.7 Функция «Отклонение откликов пользователей» ... 47
3.8.2.8 Функция «Просмотр откликов на вакансии» ... 48
3.8.2.9 Функция «Редактирование профиля» ... 48
3.8.2.10 Функция «Просмотр вакансий компании» ... 49
3.8.2.11 Функция «Выход из компании» ... 50
3.8.3 Функции пользователя с ролью «Соискатель» ... 50
3.8.3.1 Функция «Создание резюме» ... 50
3.8.3.2 Функция «Изменение резюме» ... 51

3.8.3.3 Функция «Удаление резюме» 52
3.8.3.4 Функция «Создание отклика на вакансию» 52
3.8.3.5 Функция «Отмена отклика на вакансию» 53
3.8.3.6 Функция «Просмотр откликов пользователя» 54
3.8.3.7 Функция «Просмотр кталога вакансий» 55
3.8.4 Функции пользователя с ролью «Администратор» 56
3.8.4.1 Функция «Удаление пользователей» 56
3.8.4.2 Функция «Создание отрасли» 57
3.8.4.3 Функция «Удаление отрасли» 58
3.8.4.4 Функция «Одобрение запроса на присоединение к компании» 58
3.8.4.5 Функция «Отклонение запроса на присоединение к компании» 60
3.9 Описание маршрутов и контроллеров 60
3.10 Реализация клиентской части приложения 62
3.11 Файлы dockerfile, docker-compose, файл конфигурации nginx 63
3.12 Выводы по разделу 67
4 Тестирование веб-приложения 68
4.1 Функциональное тестирование 68
4.2 Автоматизированнео тестирование 73
4.3 Вывод по разделу 75
5 Руководство программиста 76
5.1 Скачивание и установка 76
5.2 Проверка работоспособности приложения 76
5.3 Заключение по разделу 77
Заключение 78
Список использованных источников 78
Приложение А 80
Приложение Б 82
Приложение В 82
Приложение Г 84

Введение

Приложение для трудоустройства – это приложение, которое является местом встречи работодателей, желающих найти персонал, и соискателей – это безработные люди, находящиеся в активном поиске рабочего места.

Целью данного проекта является улучшение процесса поиска рабочего места и подбора персонала.

Для достижения цели проекта будет разработано веб-приложение для трудоустройства, предоставляющее пользователям комфортный и быстрый способ поиска рабочего места и подбора персонала.

Задачи в процессе выполнения курсового проекта:

- постановка задачи(Раздел 1);
- обзор аналогичных решений(Раздел 2);
- проектирование веб-приложения(Раздел 3);
- реализация веб-приложения(Раздел 4);
- тестирование веб-приложения(Раздел 5).

Целевой аудиторией веб-приложения будут являться: соискатели – пользователь, находящиеся в поиске работы, работодатели – пользователи, представляющие компании, которые находятся в поиске сотрудников.

Для разработки серверной части веб-приложения будет использована программная платформа Node.js 20.17.0 [1] с библиотеками Express.js 4.21.2 [2], mongoose 8.10.1 [3]. В качестве хранилища данных будет использована NoSQL база данных MongoDB 8.0.3 [4]. Для разработки клиентской части будет использован React.js 19.0.0 [5].

1 Постановка задачи и обзор аналогичных решений

Веб-приложение для трудоустройства – это платформа, предназначенная для публикации вакансий работодателями с целью поиска сотрудников, а также предоставления соискателям возможности поиска и подбора подходящей вакансии. После выбора желаемой вакансии соискатель оставляет отклик на данную вакансию. Работодатель, разместивший вакансию, может просматривать пользователей, которые оставили отклик на данную вакансию, и производить отбор кандидатов.

1.1 Jobs Redefined

«Jobs Redefined» – это веб-приложение для размещения вакансий, ориентированный на европейский рынок. Интерфейс приложения представлен на рисунке 1.1. Ссылка на приложение: https://jobs-redefined.co/.

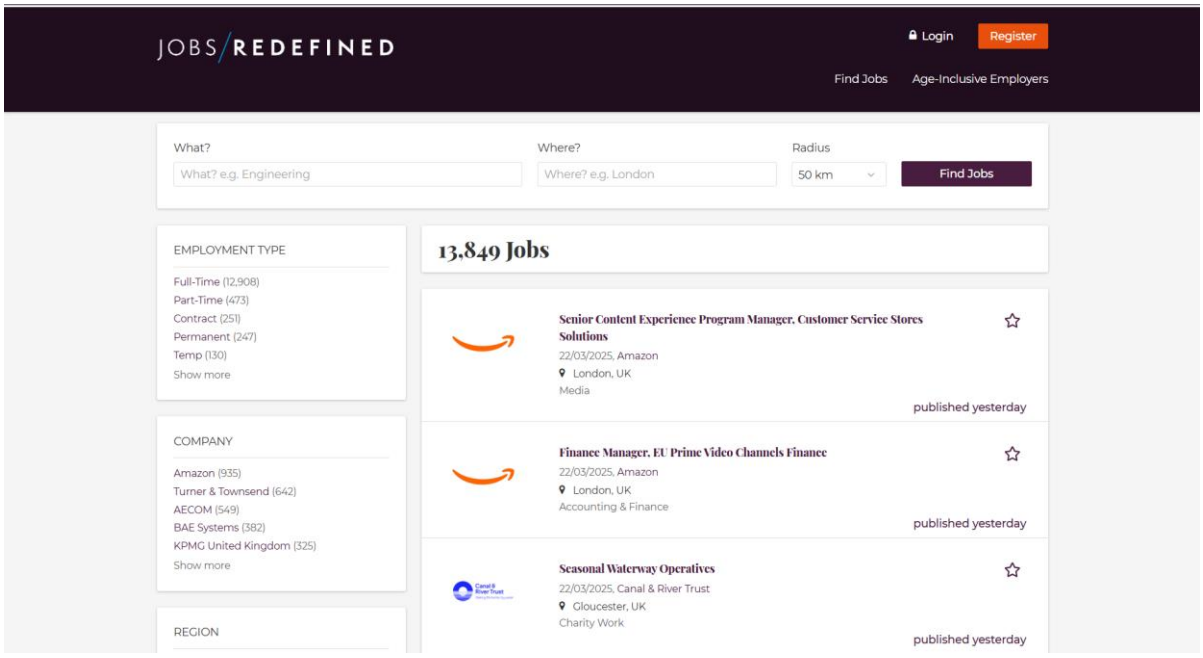


Рисунок 1.1 – Веб-приложение «Jobs Redefined»

Приложение ориентировано на 2 типа пользователей: работодатели и сосикатели. Функционал соискателей включает в себя:

- просмотр каталога вакансий;
- редактирование профиля;
- создание резюме;
- отклик на вакансии;
- поиск вакансий по фильтрам “что?”, “где?”, радиусу 5-200 км, по популярным отраслям и по типу работы.

Интерфейс страницы вакансии представлен на рисунке 1.2.

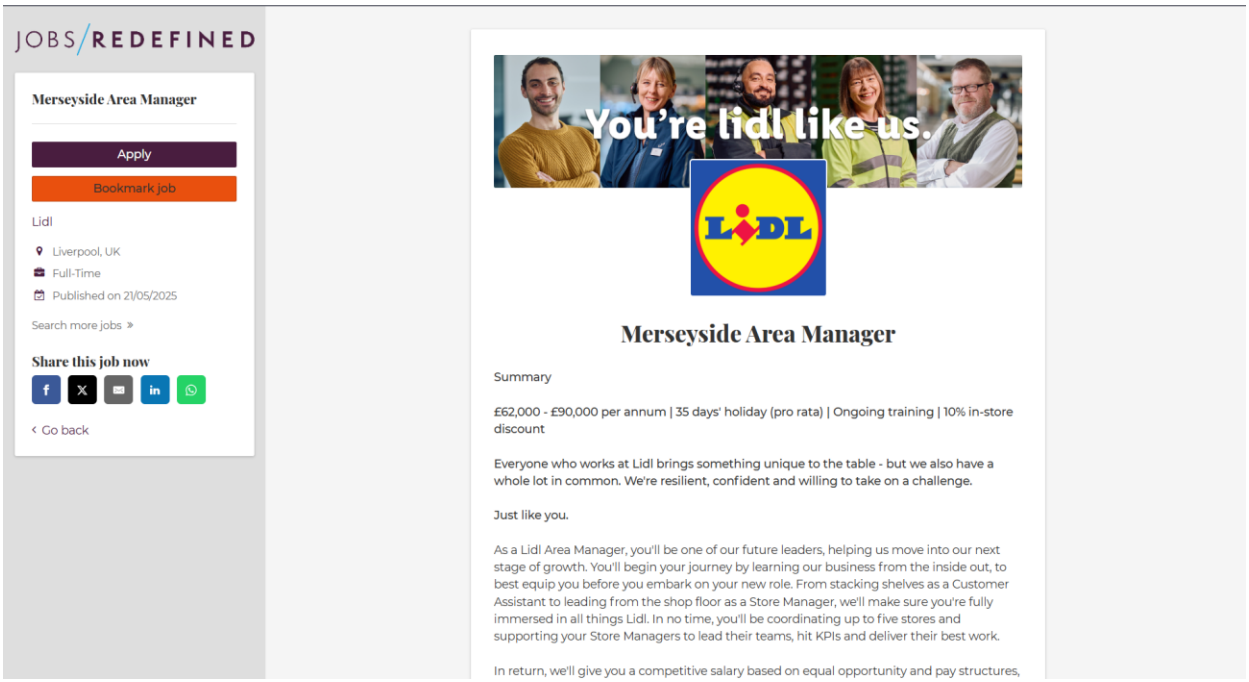


Рисунок 1.2 – Интерфейс страницы вакансии

Из недостатков данного приложения можно выделить создание единственного резюме на аккаунт. Также отклик на вакансию представляет из себя переход на сайт работодателя, на котором снова придется регистрироваться. Помимо всего вышеперечисленного у вакансий отсутствуют предлагаемые зарплаты.

Из плюсов данного приложения можно выделить поиск вакансий в определенном радиусе от пользователя.

Таким обрахом, приложение Jobs Redefined является не очень хорошим аналогичным решением.

1.2 Headhunter

Headhunter – одна из крупнейших платформ для поиска работы в странах СНГ. Позволяет размещать вакансии, создавать и отправлять резюме и проходить тестирования. Ориентирована на широкий круг специальностей. Интерфейс приложения представлен на рисунке 1.3. Ссылка на приложение: https://hh.ru/.

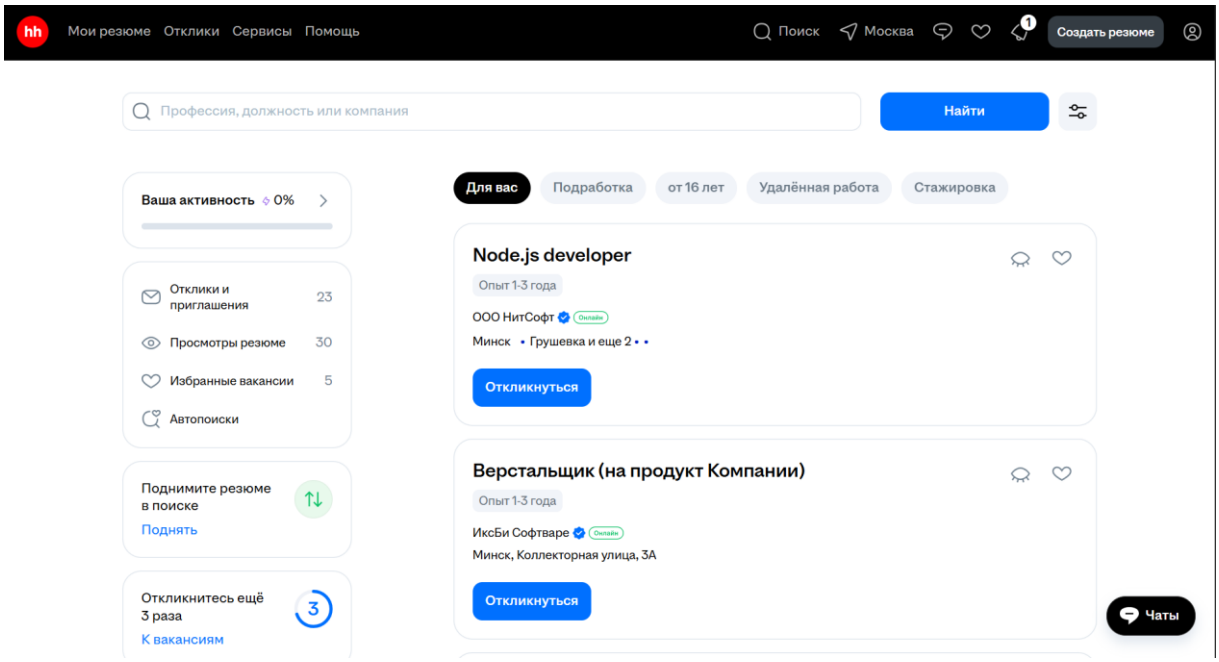


Рисунок 1.3 – Веб-приложение «Headhunter»

Отличительная черта hh.ru – обширная база вакансий, охватывающая практически все сферы занятости. Здесь можно найти предложения как для офисных сотрудников, так и для рабочих специальностей, стажеров, студентов и даже фрилансеров. Удобная система фильтров позволяет быстро находить вакансии по уровню зарплаты, опыту работы, формату занятости и другим критериям.

Также hh.ru предлагает воспользоваться удобным конструктором резюме. Заполнив данные, резюме можно скачать в определенном формате. Интерфейс конструктора представлен на рисунке 1.4.

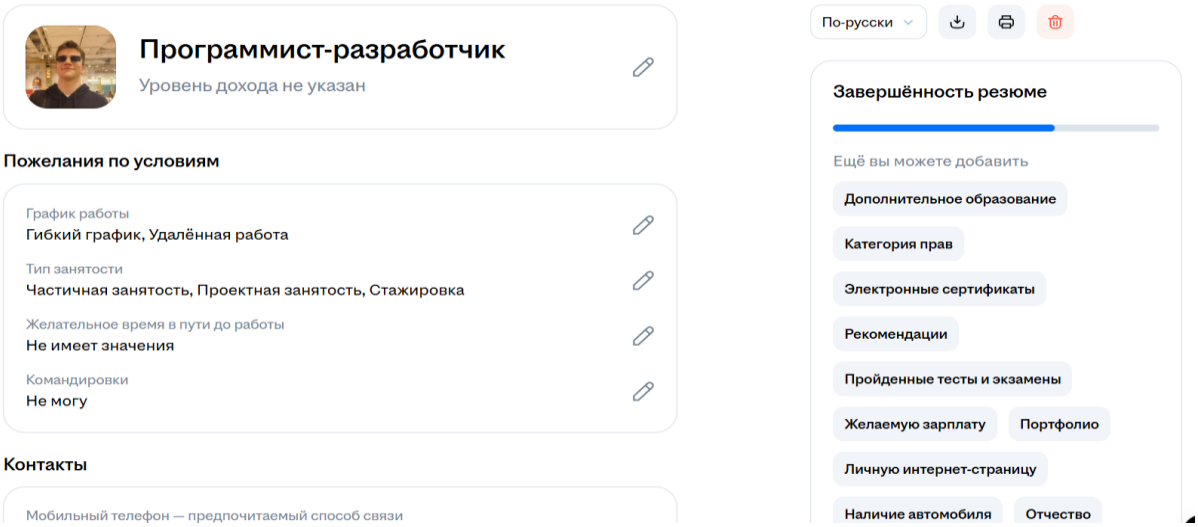


Рисунок 1.4 – Интерфейс конструктора резюме

Однако у hh.ru есть и слабые стороны. Многие вакансии остаются без ответа, так как работодатели не всегда проверяют отклики. Кроме того, некоторые компании не указывают зарплату в вакансиях, что затрудняет выбор. Интерфейс платформы удобный, но выглядит несколько устаревшим по сравнению с западными аналогами.

1.3 Вахтовик

Веб-приложение «Вахтовик» – это онлайн-ресурс, специализирующийся на вакансиях с вахтовым методом работы по всей России. Сайт представляет собой блог на WordPress, где каждое объявление оформлено в виде отдельного поста: заголовок вакансии, краткое описание условий и кнопка «Показать номер» для связи с работодателем. Интерфейс приложения представлен на рисунке 1.4. Ссылка на ресурс: https://vahtovk.ru.

Рисунок 1.4 – Веб-ресурс «Вахтовик»

Так как сайт представляет из себя блог, то публикацией вакансий занимается администратор ресурса. Вакансии представляют из себя изображения с текстом объявлений. Отклики на вакансии отсутствуют, вместо них кнопка «Показать номер», которая предоставляет способы связи с работодателем.
Из минусов можно выделить:

- отсутствие возможности создания личного кабинета;
- отсутствие фильтров для поиска вакансий;
- вакансии представляют из себя изображения с объявлениями.

Таким образом данное решение совершенно не подходит под требования разрабатываемого приложения и является самым худшим из всех решений, рассмотренных в этом разделе.

1.4 Вывод по разделу

В результате анализа аналогичных решений можно выделить основные функциональные требования характерные для разрабатываемого приложения:

- регистрация;
- авторизация;
- редактирование профиля;
- создание, отмена запроса на присоединение к компании;
- выход из компании;
- создание, удаление, изменение вакансий компании;
- отклонение, одобрение откликов пользователей;
- просмотр откликов на вакансию;
- создание, удаление, изменение резюме пользователя;
- создание, отмена откликов пользователей;
- просмотр каталога вакансий;
- сортировка вакансий (дата создания, зарплата);
- фильтрация вакансий (отрасль, зарплата, дата создания, опыт);
- удаление пользователей;
- создание, удаление отраслей;
- одобрение, отклонение запроса на присоединение к компании.

Также опередлено создание 4 ролей в приложении.

2 Проектирование web-приложения

2.1 Диаграмма вариантов использования

В результате обзора аналогичных решений приложения для трудоустройства можно выделить основные функциональные требования[6] и определить роли пользователей в приложении.

Пользователи приложения делятся на 4 роли:

- администратор;
- работодатель;
- соискатель;
- гость.

Описание ролей приведено в таблице 2.1.

Таблица 2.1 – Описание ролей

Роль	Описание
Гость	Неавторизованный пользователь в приложении. Функционал: • регистрация; • авторизация;
Работодатель	Зарегистрированный в приложении пользователь, представляющий компанию. Функционал: • запрос на присоединение к компании; • отмена запроса на присоединение к компании; • выход из компании; • редактирование профиля пользователя; • создание, изменение и удаление вакансий; • просмотр откликов на вакансию; • просмотр вакансий компании; • одобрение/отклонение откликов пользователей.
Соискатель	Зарегистрированный в приложении пользователь, находящийся в поиске работы. Функционал: • редактирование профиля пользователя; • создание, удаление, изменение резюме; • создание и отмена отклика на вакансию; • выход из компании; • просмотр каталога вакансий; • просмотр откликов пользователя; • сортировка вакансий(дата добавления, зарплата); фильтрация вакансий(отрасль, зарплата, дата добавления).

Администратор	Самый первый пользовать в приложении, профиль которого изначально находится в базе данных. Функционал: • удаление и изменение вакансий; • удаление пользователей; • создание и удаление отраслей; • просмотр каталога вакансий; • одобрение/отклонение запроса на присоединение к компании.

Далее приведены таблицы с описание функций каждой роли. Таблица 2.2 описывает функции гостя.

Таблица 2.2 – Описание функций гостя

№	Функция	Описание
1	Авторизация	Функция предоставляет гостю доступ к функционалу той роли, к которой принадлежит учетная запись пользователя.
2	Регистрация	Функция необходима для создания нового пользователя в приложении.

В таблице 2.3 приведено описание функция работодателя.

Таблица 2.3 – Описание функция работодателя

№	Функция	Описание
3	Запрос на присоединение к компании	Функция создает запрос на присоединение работодателя к компании для дальнейшего управления вакансиями компании. В дальнейшем админ может принять или отклонить данный запрос.
4	Отмена запроса на присоединение к компании	Функция позволяет работодателю отменить запрос на присоединение к компании.
5	Создание вакансии	Функция позволяет создать вакансию. Данная функция доступна только работодателям, состоящим в компании.
6	Удаление вакансии	Функция позволяет удалить вакансию. Данная функция доступна только работодателям, состоящим в компании. Вакансия должна относиться к компании работодателя.
7	Изменение вакансии	Функция позволяет изменять вакансию. Данная функция доступна только работодателям, состоящим в компании. Вакансия должна относиться к компании работодателя.

11	Редактирование профиля	Функция позволяет пользователю изменить данные профиля.
12	Просмотр вакансий компании	Функция позволяет работодателю получить список вакансий опубликованных его компанией.
10	Просмотр откликов на вакансию	Функция позволяет работодателю просмотреть пользователей, которые откликнулись на вакансию компании. Данная функция доступна только работодателям, состоящим в компании. Вакансия должна относиться к компании работодателя.
8	Одобрение отклика пользователя	Функция позволяет работодателю дать положительный ответ, пользователю откликнувшемуся на вакансию. Данная функция доступна только работодателям, состоящим в компании. Вакансия должна относиться к компании работодателя.
9	Отклонение отклика пользователя	Функция позволяет работодателю дать отрицательный ответ, пользователю откликнувшемуся на вакансию. Данная функция доступна только работодателям, состоящим в компании. Вакансия должна относиться к компании работодателя.
13	Выход из компании	Функция позволяет пользователю выйти из компании.

В таблице 2.4 приведено описание функций соискателя.

Таблица 2.4 – Описание функций соискателя

№	Функция	Описание
11	Редактирование профиля	Функция позволяет пользователю изменить данные профиля.
14	Создание резюме	Функция позволяет пользователю создать резюме.
15	Изменение резюме	Функция позволяет пользователю изменить резюме, которое принадлежит ему.
16	Удаление резюме	Функция позволяет пользователю удалить резюме, которое принадлежит ему.
17	Создание отклика	Функция позволяет соискателю сделать отклик на вакансию.
18	Отмена отклика	Функция позволяет соискателю отменить, сделанный отклик.
20	Просмотр каталога вакансий	Функция позволяет пользователю получить список вакансий.

19	Просмотр откликов пользователя	Функция позволяет соискателю просмотреть все свои отклики на вакансии.
20	Фильтрация вакансий	Функция позволяет соискателю производить фильтрацию вакансий по параметрам: название, зарплата, отрасль, опыт.
20	Сортировка вакансий	Функция позволяет сосикателю производить сортировку вакансий по параметрам: дата создания, зарплата.

В таблице 2.5 приведено описание функций администратора.

Таблица 2.5 – Описание функция администратора

№	Функция	Описание
21	Удаление пользователей	Функция позволяет удалить пользователя.
6	Удаление вакансии	Функция позволяет удалить вакансию.
7	Изменение вакансии	Функция позволяет изменить вакансию.
24	Одобрение запроса на присоединение к компании	Функция позволяет одобрить запрос работодателя на присоединение к компании.
25	Отклонение запроса на присоединение к вакансии	Функция позволяет отклонить запрос работодателя на присоединение к компании.
20	Просмотр каталога вакансий	Функция позволяет пользователю получить список вакансий.
22	Создание отрасли	Функция позволяет создать отрасль.
23	Удаление отрасли	Функция позволяет удалить отрасль.

По функционалу ролей можно составить диаграмму вариантов использования. Диаграмма вариантов использования представлена в приложении А.

UML диаграмма вариантов использования[7] позволяет определить функционал для каждой роли в веб-приложении.

2.2 Проектирование базы данных

2.2.1 Логическая схема базы данных

В приложении можно выделить несколько сущностей, которые будут соответствовать коллекциям в базе данных: пользователь, соискатель, работодатель, вакансия, резюме.

Логическая схема базы данных представлена приложении Б.

2.2.2 Коллекции базы данных

Описание коллекций базы данных приведено в таблице 2.6.

Таблица 2.6 – Описание коллекций

Коллекция	Описание
Users(пользователи)	Коллекция пользователей, которая хранит данные всех зарегистрированных пользователей: как соискателей, так и работодателей, и администраторов.
Employers(работодатели)	Коллекция работодателей, хранящая данные о работодателях.
Vacancies(вакансии)	Коллекция вакансий, содержащая вакансии, размещаемые работодателями, которые состоят в компании.
Applicants(сосикатели)	Коллекция соискателей, хранящая данные о соискателях.
Resumes(резюме)	Коллекция резюме, которая хранит резюме для соискателей.
Responses(отклики)	Коллекция откликов на вакансии, хранящая данные отклика соискателя на вакансию.
JoinCompanyRequests(запросы на присоединение к компании)	Коллекция запросов работодателей на присоединение к компании.
IndustryTypes(отрасли)	Коллекция отраслей вакансий.

2.2.3 Описание полей

Описание полей каждой коллекции приведено в таблице 2.7.

Таблица 2.7 – Описание полей коллекций

Поле	Описание
Users(пользователи)	
Name(имя)	Поле, содержащее имя пользователя. Строковое значение.
Contacts(контакты)	Содержит данные контактов пользователя, представляет из себя вложенный объект с полями: • Email – эл. почта, используемая пользователем для авторизации. Строковое значение.

	• Phone – номер телефона пользователя. Строковое значение.
Role(роль)	Определяет роль пользователя в приложении. Строковое значение.
Hashed_password(хешированный пароль)	Содержит пароль пользователя в захешированном виде. Строковое значение.
Employers(работодатели)	
User(пользователь)	Содержит id указывающий на документ в коллекции Users. Тип ObjectId.
Company(компания)	Содержит данные о компании к которой присоединен работодатель. Если null, то работодатель без компании и не может создавать, редактировать и удалять вакансии. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Строковое значение; • Activity – деятельность компании. Строковое значение; • Name – название компании. Строковое значение; • Boss_contacts – содержит данные контактов владельца компании, представляет из себя объект с полями: o Phone – номер телефона. Строковое значение; o Email – эл. Почта. Строковое значение.
Requested_company (регистрационный номер компании)	Содержит рег. номер компании, к которой работодатель отправил запрос на присоединение. Null если работодатель в компании или не находится в поиске компании. Строковое значение.
Vacancies(вакансии)	
Name(название)	Содержит название вакансии. Строковое значение.
Describe(описание вакансии)	Содержит описание вакансии. Строковое значение.
Salary_amount(зарплата)	Содержит размер зарплаты. Числовое значение.
Currency(валюта)	Содержит значение валюты. Строковое значение.

Required_experience (требуемый опыт)	Содержит необходимый опыт работы. Числовое значение.
Company(компания)	Содержит данные о компании к которой пренадлежит вакансия. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Строковое значение; • Activity – деятельность компании. Строковое значение; • Name – название компании. Строковое значение;
Industry_id(отрасль)	Содержит id отрасли ссылающийся на коллекцию IndustriesType. Тип ObjectId.
Applicants(сосикатели)	
User(пользователь)	Содержит id указывающий на документ в коллекции Users. Тип ObjectId.
Resumes(резюме)	Содержит массив резюме пользователя, представляет из себя массив id типа ObjectId, ссылающихся на коллекцию resumes.
Resumes(резюме)	
Applicant_id(соискатель)	Содержит id соискателя, которому пренадлежит резюме, типа ObjectId, ссылающийся на коллекцию Applicants.
Name(название)	Содержит название резюме. Строковое значение.
Biography(биография)	Содержит биографию в резюме соискателя. Строковое значение.
Skills(навыки)	Содержит данные о навыках в резюме соискателя. Массив строковых значений.
Work_experience(опыт работы)	Содержит данные об опыте работы, предыдущих местах работы. Массив объектов с полями: • Company – название компании в которой работал соискатель. Строковое значение; • Position – должность, которую занимал соискатель в указанной компании. Строковое значение. • Years_of_work – количество отработанных лет в указанной окмпании.

Responses(отклики)	
Applicant_id(соискатель)	Содержит id соискателя, который откликнулся на вакансию, типа ObjectId, ссылающийся на коллекцию Applicants.
Vacancy_id(вакансия)	Содержит id вакансии, на которую сделан отклик, типа ObjectId.
Resume_id(резюме)	Содержит id резюме, которое было прикреплено к отклику на вакансию, типа ObjectId и ссылается на коллекцию Resumes.
Is_approved(одобрено)	Флаг, указывающий одобрена ли вакансия или нет. Булевое значение. True - отклик одобрен, false - отклик отменен, null - отклик не просмотрен.
Applicant_pinned_message(прикрепленное сообщение соискателя)	Содержит прикрепленное соискателем сообщение к отклику на вакансию. Строковое значение.
Employer_pinned_message(прикрепленное сообщение работодателя)	Содержит прикрепленное работодателем сообщение к ответу на отклик на вакансию. Строковое значение.
IndustryTypes(отрасли)	
Industry_type(название)	Содержит название отрасли. Стрококвый тип.
JoinCompanyRequests(запросы на присоединение к компании)	
Company_regnum(регистрационный номер компании)	Содержит регистрационный номер компании, к которой желает присоединиться работодатель. Строковое значение
Employer(работодатель)	Содержит id работодателя, который отправил запрос на присоединение к компании, типа ObjectId, ссылающийся на коллекцию Employers.
Pinned_message(прикрепленное сообщение)	Содержит прикрепленное работодателем сообщение к запросу. Строковое значение.
Company(компания)	Содержит данные о компании к которой хочет присоединиться работодатель. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Строковое значение; • Activity – деятельность компании. Строковое значение;

	• Name – название компании. Строковое значение; • Boss_contacts – содержит данные контактов владельца компании, представляет из себя объект с полями: o Phone – номер телефона. Строковое значение; o Email – эл. почта. Строковое значение.

Помимо указанных в таблице 2.7 полей в базе данных автоматически генерируются поля для каждой коллекции:

- CreatedAt – временная метка создания документа;
- UpdatedAt – временная метка последнего обновления документа;
- _id – уникальный идентификатор документа в коллекции.

2.2.4 Описание связей

Описание связей между коллекциями приведено в таблице 2.8.

Таблица 2.8 – Описание связей

Коллекция	Поле	Связанная коллекция	Тип связи	Описнаие
Работодатели	User	Пользователи	Один к одному	Пользователь может иметь только один профиль работодателя
Запрос на присоединение к компании	Employer	Работодатели	Один к одному	Один один запрос принадлежит одному работодателю
Вакансии	Industry_id	Отрасли вакансий	Многие к одному	Несколько вакансий может относиться к одной отрасли
Соискатели	User	Пользователи	Один к одному	Пользователь может иметь только один профиль соискателя
Резюме соискателя	Applicant_id	Соискатели	Один к одному	Одно резюме относится к одному соискателю
Соискатели	Resumes	Резюме соискателя	Один ко многим	Один соискатель может иметь несколько резюме

Отклики на вакансии	Vacancy_id	Вакансии	Многие к одному	Несколько откликов может относиться к одной вакансии
Отклики на вакансии	Applicant_id	Соискатели	Многие к одному	Несколько откликов может относиться к одному сосикателю
Отклики на вакансии	Resume_id	Резюме соискателя	Один к одному	Один отклик может содержать одно резюме

Таким образом, база данных способна хранить данные необходимые для работы всех функций приложения.

2.3 Развертывание веб-приложения

Схема развертывания веб-приложения представлена в приложении В.
В таблице 2.9 описаны основные элементы системы

Таблица 2.9 – Назначение элементов системы

Элемент	Назначение
Database Server	Используется для хранения и предоставления доступа к данным, которые необходимы для работы веб-приложения.
Application Server	Обрабатывает запросы пользователя, запрашивать данные из базы данных.
Web Server Nginx v1.27.5[8]	Предоставляет доступ к статическим ресурсам веб-приложения.
Chrome Browser	Клиент, посылающий запросы на сервер.

Описание протоколов, используемых при работе web-приложений, представлено в таблице 2.10.

Таблица 2.10 – Описание протоколов

Протокол	Назначение
HTTP 1.1[9]	Обмен данными между Nginx и Application Server, Nginx и Chrome Browser.
TCP [10]	Обмен данными между Database Server и Application Server

Для развертывания веб-приложения использовался Docker[11] с контейнерами на базе Alpine Linux v3.20 [12]. Образы сервисов были собраны с использованием минималистичного дистрибутива Alpine v3.20 для оптимизации размера. Контейнеры объединены с помощью Docker Compose[13].

2.4 Вывод по разделу

В данном разделе была составлена UML диаграмма вариантов в использования, спроектирована нереляционная база данных веб-приложения состоящая из 8 коллекций, были определены структуры документов в этих коллекциях. Также была спроектирована схема развертывания веб-приложения, состоящая из 3 контейнеров, объединенных с помощью Docker Compose.

3 Разработка web-приложения

3.1 Программная платформа Node.js

Для разработки серверной части веб-приложения была выбрана платформа Node.js v20.17.0. В рамках проекта Node.js используется совместно с фреймворком Express.js, который упрощает разработку маршрутов и API. Express.js – это минималистичный и гибкий веб-фреймворк для Node.js, предназначенный для создания серверных приложений. Он предоставляет простой набор инструментов для обработки HTTP-запросов, маршрутизации и работы с middleware.

3.2 База данных MongoDB

В качестве хранилища данных для веб-приложения была выбрана база данных MongoDB v8.0.3. Так как данные в MongoDB могут храниться в разном формате, а также присутствует возможность хранения вложенных данных, то данная база данных идеально подходит для разрабатываемого приложения.

3.3 ODM Mongoose

Для взаимодествия с базой данных используется ODM[14](Object Data Modelling)-библиотека Mongoose v8.12.1 – это ODM-библиотека для Node.js, которая позволяет работать с базой данных MongoDB. С ее помощью взаимодействие с коллекциями базы данных происходит, как с программными объектами JS.

Взаимодействие приложения с базой данных начинается с подключения к ней. Листинг кода подключения к базе данных представлен в листинге 3.1.

```
mongoose.connect(process.env.DB_URI)
.then(() => console.log('connected to db'))
.catch((err) => console.log(`error connecting to db: ${err}`))
```

Листинг 3.3.1 – Контекст базы данных

3.3.1 Схемы в mongoose

Mongoose обеспечивая автоматическое управление схемами и типизированные модели данных. Она поддерживает сложные связи между документами. Mongoose включает встроенную валидацию данных, хуки и плагины. Сопоставление схем, используемых в Mongoose, с их коллекциями документов в базе данных представлено в таблице 3.3.1.

Таблица 3.3.1 – Сопоставление схем, используемых в Mongoose

Схема	Коллекция в базе данных
UserSchema	Users

ApplicantSchema	Applicants
EmployerSchema	Employers
VacancySchema	Vacancies
ResponseSchema	Responses
ResumeSchema	Resumes
IndustryTypeSchema	IndustryTypes
JoinCompanyRequestSchema	JoinCompanyRequests

Код схемы UserSchema представлен в листинге 3.3.2.

```
const UserSchema = new mongoose.Schema(
    {
        name: {
            type:String,
            required: true,
        },
        password_hash: {
            type: String,
            required: true,
        },
        contacts: {
            email: {
                type: String,
                required: false,
                match: [/^\S+@\S+\.\S+$/, 'некорректный email'],
            },
            phone: {
                type: String,
                required: false,
                match:   [/^\+?\d{10,15}$/,   'некорректный   номер
телефона'],
                default: null
            }
        },
        role: {
            type: String,
            enum: ['applicant', 'employer', 'admin'],
            required: true,
        }
    },
    {
        timestamps: true
    }
)
```

Листинг 3.3.2 – Код UserSchema

Схема UserSchema описывает структуру документов коллекции Users полями, представленными в таблице 3.3.2.

Таблица 3.3.2 – Поля схемы UserSchema

Название	Тип данных	Ограничение	Описание
Name	string	Обязательно	Поле, содержащее имя пользователя. Строковое значение.
Contacts	Object	Обязательно	Содержит данные контактов пользователя, представляет из себя вложенный объект с полями: • Email – эл. почта, используемая пользователем для авторизации. Тип данных string. Ограничения: обязательное, соответствие регулярному выражению /^\S+@+\.\S+$/. • Phone – номер телефона пользователя. Тип данных: string. Ограничения: необязательно, по умолчанию null, соответствие регулярному выражению: /^\+?\d{10,15}$/.
Role	string	Обязательно	Определяет роль пользователя в приложении.
Hashed_password	string	Обязательно	Содержит пароль пользователя в захешированном виде.

Код схемы ApplicantSchema представлен в листинге 3.3.3.

```
const ApplicantSchema = new mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User',
            required: true
        },
        resumes: [{
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Resume',
        }]
    },
    {
```

```
            timestamps: true,
        }
    )
```

Листинг 3.3.3 – Код ApplicantSchema

Схема ApplicantSchema описывает структуру документов коллекции Applicants полями, представленными в таблице 3.3.2.

Таблица 3.3.2 – Поля схемы ApplicantSchema

Название	Тип данных	Ограничения	Описание
User(пользователь)	ObjectId	Обязательно	Содержит id указывающий на документ в коллекции Users.
Resumes(резюме)	ObjectId[]	Обязательно, ссылка на коллекцию Resume	Содержит массив id резюме пользователя, ссылающихся на коллекцию resumes.

Код схемы EmployerSchema представлен в листинге 3.3.4.

```
    const EmployerSchema = new mongoose.Schema(
        {
            user: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'User',
                required: true
            },
            company: {
                type: {
                    company_regnum: {
                        type: String,
                        required: true,
                    },
                    boss_contacts: {
                        email: {
                            type: String,
                            required: false,
                        },
                        phone: [{
                            type: String,
                            required: false,
                        }]
                    },
                    activity: {
                        type: String,
                        required: false,
                    },
```

```
                    name: {
                        type: String,
                        required: true,
                    }
                },
                required: false,
                default: null,
            },
            requested_company: {
                type: String,
                required: false,
                default: null
            }
        },
        {
            timestamps: true
        }
    )
```

Листинг 3.3.4 – Код EmployerSchema

Схема EmployerSchema описывает структуру документов коллекции Employers полями, представленными в таблице 3.3.3.

Таблица 3.3.3 – Поля схемы EmployerSchema

Название	Тип данных	Ограничения	Описание
User	ObjectId	Обязательно, ссылка на коллекцияю User	Содержит id указывающий на документ в коллекции Users.
Company	Object	Необязательно, по умолчанию null	Содержит данные о компании к которой присоединен работодатель. Если null, то работодатель без компании и не может создавать, редактировать и удалять вакансии. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Тип данных: string. Ограничения: обязательно;

			• Activity – деятельность компании. Тип данных: string. Ограничения: необязательно; • Name – название компании. Тип данных: string. Ограничения: обязательно; • Boss_contacts – содержит данные контактов владельца компании, представляет из себя объект с полями: o Phone – номер телефона. Тип данных: string. Ограничения: необязательно; o Email – эл. Почта. Тип данных: string. Ограничения: неыобязательно.
Requested_company	string	Необязательно, по умолчанию null	Содержит рег. номер компании, к которой работодатель отправил запрос на присоединение. Null если работодатель в компании или не находится в поиске компании. Строковое значение.

Код схемы VacancySchema представлен в листинге 3.3.5.

```
const VacancySchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: true,
        },
        describe: {
            type: String,
            required: true,
        },
        salary_amount: {
            type: Number,
            required: function () { return !!this.currency; },
            default: null
        },
```

```
            currency: {
                type: String,
                enum: ['USD', 'EUR', 'RUB', 'BYN'],
                required: function () { return !!this.salary_amount;
}
            },
            required_experience: {  // в годах
                type: Number,
                required: false,
                default: null,
            },
            company: {
                type: {
                    company_regnum: {
                        type: String,
                        required: true,
                    },
                    name: {
                        type: String,
                        required: true
                    },
                    activity: {
                        type: String,
                        required: false
                    }
                },
                required: true,
            },
            industry_id: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'IndustryType',
                required: true,
            },
        },
        {
            timestamps: true,
        }
    )
```

Листинг 3.3.5 – Код VacancySchema

Схема VacancySchema описывает структуру документов коллекции Vacancies полями, представленными в таблице 3.3.4.

Таблица 3.3.4 – Поля схемы VacancySchema

Название	Тип данных	Ограничения	Описание
Name	string	Обязательно	Содержит название вакансии.
Describe	string	Обязательно	Содержит описание вакансии.

Salary_amount	number	По умолчанию null	Содержит размер зарплаты.
Currency	string	По умолчанию null	Содержит значение валюты.
Required_experience	number	Необязательно, по умолчанию null	Содержит необходимый опыт работы.
Company	Object	Обязательно	Содержит данные о компании к которой пренадлежит вакансия. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Тип данных: string. Ограничения: обязательно; • Activity – деятельность компании. Тип данных: string. Ограничения: необязательно; • Name – название компании. Тип данных: string. Ограничения: обязательно.
Industry_id(отрасль)	ObjectId	Обязательно, ссылка на коллекцию IndustryType	Содержит id отрасли ссылающийся на коллекцию IndustriesType.

Код схемы ResumeSchema представлен в листинге 3.3.6.

```
const ResumeSchema = new mongoose.Schema(
    {
        applicant_id: {
            type: mongoose.Schema.Types.ObjectId,
            required: true
        },
        name: {
            type: String,
            required: true,
        },
        biography: {
```

```
                type: String,
                required: false,
            },
            skills: {
                type: [{
                    type: String,
                    maxlength: 20
                }],
                validate: {
                    validator: function(arr) {
                        return arr.length <= 15
                    },
                    message: 'Массив skills не должен содержать
более 15 элементов'
                }
            },
            work_experience: [{
                company: {
                    type: String,
                    required: true,
                },
                position: {
                    type: String,
                    required: true,
                },
                years_of_work: {
                    type: Number,
                    required: false,
                }
            }]
        },
        {
            timestamps: true,
        }
    )
```

Листинг 3.3.6 – Код ResumeSchema

Схема ResumeSchema описывает структуру документов коллекции Resumes полями, представленными в таблице 3.3.5.

Таблица 3.3.5 – Поля схемы ResumeSchema

Название	Тип данных	Ограничения	Описание
Applicant_id	ObjectId	Обязательно	Содержит id соискателя, которому пренадлежит резюме.
Name	string	Обязательно	Содержит название резюме.
Biography	string	Необязательно	Содержит биографию в резюме соискателя.

Skills	String[]	Необязательно	Содержит данные о навыках в резюме соискателя.
Work_experience	Object[]	Необязательно	Содержит данные об опыте работы, предыдущих местах работы. Массив объектов с полями: • Company – название компании в которой работал соискатель. Тип данных: string. Ограничения: обязательно; • Position – должность, которую занимал соискатель в указанной компании. Тип данных: string. Ограничения: обязательно; • Years_of_work – количество отработанных лет в указанной окмпании. Тип данных: number. Ограничения: необязательно.

Код схемы ResponseSchema представлен в листинге 3.3.7.

```
const ResponseSchema = new mongoose.Schema(
    {
        applicant_id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Applicant',
            required: true
        },
        vacancy_id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Vacancy',
            required: true
        },
        resume_id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Resume',
            required: true
        },
        is_approved: {
            type: Boolean,
```

```
                required: false,
                default: null
            },
            applicant_pinned_message: {
                type: String,
                required: false,
                default: null,
            },
            employer_pinned_message: {
                type: String,
                required: false,
                default: null,
            }
        },
        {
            timestamps: true
        }
    )
```

Листинг 3.3.7 – Код ResponseSchema

Схема ResponseSchema описывает структуру документов коллекции Responses полями, представленными в таблице 3.3.6.

Таблица 3.3.6 – Поля схемы ResponseSchema

Название	Тип данных	Ограниряения	Описание
Applicant_id	ObjectId	Обязательно, ссылка на коллекцию Applicant	Содержит id соискателя, который откликнулся на вакансию.
Vacancy_id	ObjectId	Обязательно, ссылка на коллекцию Vacancy	Содержит id вакансии, на которую сделан отклик.
Resume_id	ObjectId	Обязательно, ссылка на коллекцию Resume	Содержит id резюме, которое было прикреплено к отклику на вакансию.
Is_approved	boolean	Необязательно, по умолчанию null	Флаг, указывающий одобрена ли вакансия или нет. True - отклик одобрен, false - отклик отменен, null - отклик не просмотрен.

Applicant_pinned_message	string	Необязательно, по умолчанию null	Содержит прикрепленное соискателем сообщение к отклику на вакансию.
Employer_pinned_message	string	Необязательно, по умолчанию null	Содержит прикрепленное работодателем сообщение к ответу на отклик на вакансию.

Код схемы IndustryTypeSchema представлен в листинге 3.3.8.

```
const IndustryTypeSchema = new mongoose.Schema(
    {
        industry_type: {
            type: String,
            required: true,
        }
    },
    {
        timestamps: true,
    }
)
```

Листинг 3.3.8 – Код IndustryTypeSchema

Схема IndustryTypeSchema описывает структуру документов коллекции IndustryTypes полями, представленными в таблице 3.3.7.

Таблица 3.3.7 – Поля схемы IndustryTypeSchema

Название	Тип данных	Ограничения	Описание
Industry_type	string	Обязательно	Содержит название отрасли.

Код схемы JoinCompanyRequestSchema представлен в листинге 3.3.9.

```
const JoinCompanyRequestSchema = new mongoose.Schema(
    {
        company_regnum: {
            type: String,
            required: true
        },
        employer: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Employer',
            required: true
        },
```

```
            pinned_message: {
                type: String,
                required: true
            },
            company: {
                type: {
                    company_regnum: {
                        type: String,
                        required: true,
                    },
                    boss_contacts: {
                        email: {
                            type: String,
                            required: false,
                        },
                        phone: [{
                            type: String,
                            required: false,
                        }]
                    },
                    activity: {
                        type: String,
                        required: false,
                    },
                    name: {
                        type: String,
                        required: true,
                    }
                },
                required: false,
                default: null,
            },
        },
    {
    timestamps: true
    }
    )
```

Листинг 3.3.9 – Код JoinCompanyRequestSchema

Схема JoinCompanyRequestSchema описывает структуру документов коллекции JoinCompanyRequestsполями, представленными в таблице 3.3.8.

Таблица 3.3.8 – Поля схемы JoinCompanyRequestSchema

Название	Тип данных	Ограничения	Описание
Company_regnum	string	Обязательно	Содержит регистрационный номер компании, к которой желает присоединиться работодатель.

Employer	ObjectId	Обязательно, ссылка на коллекцию Employer	Содержит id работодателя, который отправил запрос на присоединение к компании.
Pinned_message	string	Необязательно	Содержит прикрепленное работодателем сообщение к запросу. Строковое значение.
Company	Object	Необязательно, по умолчанию null	Содержит данные о компании к которой хочет присоединиться работодатель. Представляет из себя объект с полями: • Company_regnum – регистрационный номер компании. Тип данных: string. Ограничения: обязательно; • Activity – деятельность компании. Тип данных: string. Ограничения: необязательно; • Name – название компании. Тип данных: string. Ограничения: обязательно; • Boss_contacts – содержит данные контактов владельца компании. Ограничения: необязательно. Представляет из себя объект с полями: o Phone – номер телефона. Тип данных: string. Ограничения: необязательно; o Email – эл. почта. Тип данных: string.

			Ограничения: необязательно.

3.4 Программные библиотеки

В процессе разработки серверной части web-приложения были использованы различные программные библиотеки. Назначение библиотек представленно в таблице 3.4.1.

Таблица 3.4.1 – Программные библиотеки серверной части

Библиотека	Версия	Назначение
Express.js (https://expressjs.com/)	v4.21.2	Фреймворк для разработки серверных приложений на платформе Node.js
Mongoose (https://mongoosejs.com/docs/)	v8.12.1	ODM-библиотека для Node.js, которая позволяет работать с базой данных MongoDB
Jsonwebtoken (https://www.npmjs.com/package/jsonwebtoken)	v9.0.2	Библиотека Node.js для создания и проверки JWT
Bcryptjs (https://www.npmjs.com/package/bcryptjs)	v3.0.2	Библиотека для Node.js, которая используется для хеширования паролей. Позволяет безопасно хранить пароли в базе данных
Joi (https://joi.dev/api/?v=17.13.3)	v17.13.3	Библиотека для валидации данных в JavaScript
Cors (https://www.npmjs.com/package/cors)	v2.8.5	Библиотека, используемая для включения CORS-политики в серверных
Dotenv (https://www.npmjs.com/package/dotenv)	v16.4.7	Библиотека, позволяющая использовать переменные окружения из .env файлов
Jest(https://jestjs.io/)	V29.7.0	Библиотека, предназначенная для автоматизированного тестирования компонентов приложения

Назначение библиотек используемых при разработке клиентской части приложения представлено в таблице 3.4.2.

Таблица 3.4.2 – Программные библиотеки клиентской части

Библиотека	Версия	Назначение
ReactJS (https://react.dev/)	v19.0.0	JavaScript библиотека, предназначенная для создания одностраничных приложений(SPA)

React-router-dom (https://reactrouter.com/)	v7.3.0	Библиотека для организации маршрутизации в React-приложениях. Позволяет создавать многостраничную логику в SPA-приложениях
React-hook-form (https://react-hook-form.com/)	v7.56.2	Библиотека для управления формами в React, построенная на хуках. Предлагает оптимизацию рендеринга и валидацию
Yup (https://yup-docs.vercel.app/docs/intro)	v1.6.1	Библиотека для валидации данных в JavaScript
Zustand (https://zustand.docs.pmnd.rs/getting-started/introduction)	v5.0.3	Библиотека для управления состоянием (state management) в React-приложениях

3.5 Использование сторонних сервисов

Для разработки серверной части приложения используется сторонний сервис “Единый государственный регистр юридических лиц и индивидуальных предпринимателей”[15](https://egr.gov.by/egrn/). Документация API(Swagger) (https://egr.gov.by/api/swagger-ui.html#/main-controller/getEGRAllIPtoJurUsingGET). Данный сервис используется для получения данных о компаниях и ее владельцах.

3.6 Применяемые паттерны

При реализации серверной части приложения использовался паттерн репозиторий[16]. Данный паттерн используется для реализации уровня доступа к источнику данных (база данных). Репозиторий является прослойкой между бизнес логикой приложения и хранилищем данных.

3.7 Скрипт для создания и заполнения базы данных

Скрипт для создания базы данных MongoDB представлен в приложении В. Скрипт предназначен для создания коллекций в базе данных.

3.8 Реализация функций серверной части приложения

3.8.1 Функции пользователя с ролью «Гость»

3.8.1.1 Функция «Авторизация»

Метод login сервиса UserService предназначен для авторизации пользователя, принимает в качестве параметров email и password. Реализация метода login приведена в листинге 3.8.1.1.

```
login: async function (email, password) {
       const user = await Repositories.User.findUserByEmail(email);
        if (!user) {
              throw new AppError(401, 'Неверный email или пароль')
             }
const isMatch = await bcrypt.compare(password, user.password_hash);
        if (!isMatch) {
              throw new AppError(401, 'Неверный email или пароль')
             }
             const token = jwt.sign(
                 { id: user._id, },
                process.env.JWT_SECRET,
                 { expiresIn: "7d" }
             );
             const { _id, password_hash, ...copyUser } = user._doc
             return { token: token, user: copyUser }
         },
```

Листинг 3.8.1.1 – Код метода login

Метод проверяет наличие в базе данных пользователя с указанным email, если отсутствует то генерируется исключение и клиенту возвращается 401 код ответа. В случае, если документ с таким email существует, то происходит сравнение хешей паролей. В случае совпадения хешей генерируется JWT-токен, содержащий id пользователя из базы данных.

Также в процессе аутентификации участвует middleware authentificate, который проверяет наличие в заголовке запроса Authorization наличие JWT-токена клиента. Код middleware Authentificate приведен в листинге 3.8.1.2.

```
     const Authentificate= (req, res, next) => {
         const token = req.header("Authorization")?.split(" ")[1] ||
req.header("Authorization");
         if (!token) {
             return res.status(401).json(
                 {
                     success: false,
                     message: "Нет доступа, авторизуйтесь"
                 }
             );
         }
         try {
       const decoded = jwt.verify(token, process.env.JWT_SECRET);
             req.user = decoded;
             next();
```

```
        } catch (error) {
            res.status(401).json(
                {
                    success: false,
                    message: "Неверный токен"
                }
            );
        }
    };
```

Листинг 3.8.1.2 – Код middleware Authentificate

Если токен в заголовке Authorization отсутствует, то возвращается 401 код ответа. Иначе с помощью функции jwt.verify происходит расшифрование токена, если в процессе расшифрования возникает ошибка, то токен является невалидным. В случае успешного расшифрования токена объекту запроса добавляется поле user, которое содержит id пользователя в базе данных.

Пример использования middleware Authentificate в конвейере обработки запросов представлено в листинге 3.8.1.3.

```
    app.use('/api/admin',                    Middleware.Authentificate,
Middleware.IsUserExists, Middleware.CheckAdmin, Routes.Admin)
```

Листинг 3.8.1.3 – Пример использования middleware Authentificate

Эффективнее всего данный middleware располагать вначале конвейера обработки запросов. Так как изначально надо убедиться имеет и пользователь доступ к данной группе uri.

Также был разработан пользовательский класс исключения, который принимает параметры: код ответа сервера и сообщение. Код данного класса представлен в листинге 3.8.1.4.

```
    class AppError extends Error {
        constructor(statusCode, message) {
            super(message);
            this.statusCode = statusCode;
            Error.captureStackTrace(this, this.constructor);
        }
    }
```

Листинг 3.8.1.4 – Класс AppError

Также на сервере реализована асинхронная обработка ошибок с помощью библиотеки express-async-errors и middleware ErrorHandler для обработки этих исключений. Код middleware приведен в листинге 3.8.1.5.

```
    const ErrorHandlerMiddleware = async (err, req, res, next) => {
        console.error(err);
        res.status(err.statusCode || 500).json({
            success: false,
            message: err.message || 'Внутренняя ошибка сервера',
        });
    }
```

Листинг 3.8.1.5 – Класс AppError

3.8.1.2 Функция «Регистрация»

Метод register предназначен для создания профиля пользователя и определения его роли в приложении. В качестве параметра принимает объект user, содержащий данные пользователя (email, пароль, роль и имя). Реализация метода register приведена в листинге 3.8.1.6.

```
    register: async function (user) {
            const existingUser = await
Repositories.User.findUserByEmail(user.contacts.email);
            if (existingUser) {
                throw new AppError(400, 'email уже используется')
            }
      const hashedPassword = await bcrypt.hash(user.password, 10);
            user.password_hash = hashedPassword
            const newUser = await Repositories.User.add(user)

            let applicant, employer

            if (user.role === "applicant") {
                applicant = await
Repositories.Applicant.addApplicant(newUser.id)
            }
            if (user.role === "employer") {
                employer = await
Repositories.Employer.add(newUser.id)
            }
            if (!applicant && !employer) {
                throw new AppError(500, 'Пользователь не
зарегестрирован')
            }

            const token = jwt.sign(
```

```
                { id: newUser.id },
                process.env.JWT_SECRET,
                { expiresIn: '7d' }
            );
            const { _id, password_hash, ...copyUser } = newUser._doc
            return { token: token, data: copyUser }
        },
```

Листинг 3.8.1.6 – Код метода register

Изначально происходит проверка наличия в базе данных пользователя с таким же email, если пользователь существует, то возвращается 400 код ответа, наче создается новый документ в коллекции User, а также взависимости от параметра role создается документ в коллекции Applicant или Employer. В результате выполнения функции генерируется JWT-токен для клиента.

3.8.2 Функции пользователя с ролью «Работодатель»

3.8.2.1 Функция «Запрос на присоединение к компании»

Метод sendRequest сервиса JoinCompanyRequestService предназначен для создания запроса на присоединение работодателя к компании, в качестве параметров принимает id пользователя и объект запроса на присоединение к компании. Реализация метода sendRequest приведена в листинге 3.8.2.1.

```
sendRequest: async (userId, request) => {
if (await Repositories.User.getRole(userId) != 'employer') {
throw new AppError(400, 'Пользователь не является работодателем')}
const employer = await Repositories.Employer.getProfile(userId)
if                                                          ((await
Repositories.JoinCompanyRequest.findDuplicate(employer.id,
request.company_regnum)).length != 0) {
throw new AppError(400, 'Такой запрос уже отправлен')        }
request.employer = employer.id
const company_regnum = request.company_regnum
        const company = {}
        try {
            const                          [company_contacts_response,
company_activity_response,     company_name_response]     =     await
Promise.all([
fetch(`http://egr.gov.by/api/v2/egr/getAddressByRegNum/${company_reg
num}`),

fetch(`http://egr.gov.by/api/v2/egr/getVEDByRegNum/${company_regnum}
```

```
`),
fetch(`http://egr.gov.by/api/v2/egr/getShortInfoByRegNum/${company_r
egnum}`)            ]);
                if   (company_contacts_response.status   ==   400   ||
company_activity_response.status          ==          400          ||
company_name_response.status == 400) {
                    throw    new    AppError(400,    'Неверный    УНП
компании');
                }
const company_contacts = await company_contacts_response.json();
const company_activity = await company_activity_response.json();
const company_name = await company_name_response.json();

            company.boss_contacts = {
                email: company_contacts[0].vemail,
                phone:   company_contacts[0].vtels.split(',').map(el
=> el.trim())  }
            company.activity = company_activity[0].nsi00114.vnvdnp
            company.name = company_name[0].vfn
            company.company_regnum = company_regnum
        } catch (error) {
            console.error('Произошла ошибка:', error.message);
            throw error;
        }
        request.company = company
        const            sendedRequest            =            await
Repositories.JoinCompanyRequest.add(request)
        if (!sendedRequest) {
            throw new AppError(500, 'Запрос не отправлен')
        }
        const           updatedEmployer           =           await
Repositories.Employer.editEmployerCompany(userId,              null,
request.company_regnum)
        return updatedEmployer
    },
```

Листинг 3.8.2.1 – Код метода sendRequest

В данном методе происходит 3 запроса на сторонний сервис для получения данных о компании. Если при запросе на сторонний сервис какой-то один запрос выполнился неудачно, происходит ответ клиенту с кодом ответа 400. Затем если при добавлении данных в базу данных через метод репозитория JoinCompanyRequest add происходит ошибка, то клиенту отправялется ответ с кодом 500. Иначе новая запись

успешно сохраняется в базе данных и метод возвращает обновленный профиль работодателя.

3.8.2.2 Функция «Отмена запроса на присоединение к компании»

Метод cancelRequest сервиса JoinCompanyRequestService предназначен для отмены запроса работодателя на присоединение к компании, в качетсве параметров принимает userId. Код метода cancelRequest представлен в листинге 3.8.2.2.

```
cancelRequest: async (userId) => {
            const            employer            =            await
Repositories.Employer.getOne(userId)
            const             request             =            await
Repositories.JoinCompanyRequest.getByEmployerId(employer.id)
            if (!request) {
                throw new AppError(404, 'Запрос не найден')
            }
            const          deletedRequest          =          await
Repositories.JoinCompanyRequest.delete(request.id)
            if (!deletedRequest) {
                throw new AppError(500, 'Запрос не отменен')
            }
            const          updatedEmployer          =          await
Repositories.Employer.editEmployerCompany(userId, null, null)
            return updatedEmployer
        }
```

Листинг 3.8.2.2 – Код метода cancelRequest

В методе мы получаем документ работодателя и его запрос на присоединение к компании. Если запроса на присоединение не существует возвращаем ответ с кодом 404. Иначе удаляем запрос на присоединение и возвращаем обновленный профиль пользователя.

3.8.2.3 Функция «Создание вакансии»

Метод addVacancy сервиса VacancyService предназначен для создания вакансии, принимает в качестве параметров userId и объект vacancy. Реализация метода представлена в листинге 3.8.2.3.

```
addVacancy: async function (userId, vacancy) {
        const employer = await Repositories.Employer.getOne(userId)
        if (!employer) {
            throw new AppError(404, 'Пользователь не найден')
```

```
        }
        if (!employer.company) {
            throw new AppError(400, 'Работодатель без компании')
        }
        vacancy.company = employer.company
        const newVacancy = await Repositories.Vacancy.add(vacancy)
        if (!newVacancy) {
            throw new AppError(500, 'Вакансия не добавлена')
        }
        return newVacancy
    },
```

Листинг 3.8.2.3 – Код метода addVacancy

В методе происходит проверка на существование пользователя с ролью работодателя. Если такой пользователь отсутствует, производится ответ с кодом 404. Иначе просиходит добавление вакансии в базу данных и метод возвращает добавленную вакансию.

3.8.2.4 Функция «Удаление вакансии»

Метод deleteVacancy сервиса VacancyService предназначен для удаления вакансии, в качестве параметров принимает id пользователя и id удаляемой вакансии. Реализация метода приведена в листинге 3.8.2.4.

```
deleteVacancy: async function (userId, vacancyId) {
            if (!await Repositories.Vacancy.isExists(vacancyId)) {
                throw new AppError(404, 'Вакансия не найдена')
            }
            const userRole = await Repositories.User.getRole(userId)
            if (userRole == 'admin') {
                const deletedVacancy = await
Repositories.Vacancy.delete(vacancyId)
                if (!deletedVacancy) {
                    throw new AppError(500, 'Вакансия не удалена')
                }
                return deletedVacancy
            }
            const employer = await
Repositories.Employer.getOne(userId)
            if (!employer) {
                throw new AppError(404, 'Пользователь не найден')
            }
```

```
            const vacancy = await
Repositories.Vacancy.getOne(vacancyId)
            if (employer.company.company_regnum !=
vacancy.company.company_regnum) {
                throw new AppError(400, 'Вакансия не принадлежит
компании работодателя')
            }
            const deletedVacancy = await
Repositories.Vacancy.delete(vacancyId)
            if (!deletedVacancy) {
                throw new AppError(500, 'Вакансия не удалена')
            }
            const responses = await
Repositories.Response.deleteResponsesByVacancyId(vacancyId)
            return deletedVacancy
        },
```

Листинг 3.8.2.4 – Код метода deleteVacancy

В методе происходит проверка роли пользователя, пытающегося удалить вакансию. Удалить вакансию могут только пользователи с ролью employer и admin. Затем если пользователь имеет роль employer, происходит проверка принадлежности вакансии к компании, в которой пользователь состоит. Пользователь с ролью admin может удалять любые вакансии. В результате метод возвращает удаленную вакансию.

3.8.2.5 Функция «Изменение вакансии»

Метод editVacancy сервиса VacancyService предназначен для изменения вакансии, принимает в качестве параметров id пользователя, id изменяемой вакансии и объект измененной вакансии. Реализация метода editVacancy представлена в листинге 3.8.2.5.

```
editVacancy: async function (userId, vacancyId, newVacancy) {
        if (!await Repositories.Vacancy.isExists(vacancyId)) {
            throw new AppError(404, 'Вакансия не найдена')
        }
        const userRole = await Repositories.User.getRole(userId)
        if (userRole == 'admin') {
            const editedVacancy = await
Repositories.Vacancy.edit(vacancyId, newVacancy)
            if (!editedVacancy) {
                throw new AppError(500, 'Вакансия не изменена')
```

```
            }
            return editedVacancy
        }
     const employer = await Repositories.Employer.getOne(userId)
        if (!employer) {
            throw new AppError(404, 'Пользователь не найден')
        }
        const vacancy = await Repositories.Vacancy.getOne(vacancyId)
        if            (employer.company.company_regnum            !=
vacancy.company.company_regnum) {
          throw new AppError(400, 'Вакансия не принадлежит компании
работодателя')    }
       const            editedVacancy            =            await
Repositories.Vacancy.edit(vacancyId, newVacancy)
        if (!editedVacancy) {
            throw new AppError(500, 'Вакансия не изменена')
        }
        return editedVacancy
    },
```

Листинг 3.8.2.5 – Код метода editVacancy

В методе происходит проверка наличия пользователя с userId в базе данных. Затем проверяется роль пользователя. Доступ к изменению вакансий может иметь только работодатель, при этом работодатель должен состоять в той же компании, к которой принадлежит вакансия, иначе ответ с кодом 400.

3.8.2.6 Функция «Одобрение откликов пользователей»

Метод approveUserResponse сервиса ResponseService предназначен для одобрения отклика соискателя на вакансию, в качестве параметров принимает id отклика и прикрепленное сообщения работодателя. Реализация метода приведена в листинге 3.8.2.6.

```
     approveResponse: async function (responseId, pinnedMessage) {
            const            response            =            await
Repositories.Response.getOne(responseId)
            if (!response) {
                throw new AppError(404, 'Отклик не найден')
            }

            response.is_approved = true
            response.employer_pinned_message = pinnedMessage
```

```
            const editedResponse = await
Repositories.Response.edit(responseId, response)
            if (!editedResponse) {
                throw new AppError(500, 'Отклик не был одобрен')
            }

            return editedResponse
        },
```

Листинг 3.8.2.6 – Код метода approveUserResponse

В функции происходит проверка существования отклика и его дальнейшее изменение(перевод в состояния одобрен). В случае отсутствия отклика в базе данных возвращаем клиенту ответ с кодом 404. В случае неудачного изменения документа отклика возвращаем ответ с кодом 500.

3.8.2.7 Функция «Отклонение откликов пользователей»

Метод rejectUserResponse сервиса ResponseService предназначен для отклонения откликов соискателей на вакансию, в качестве параметров принимает id отклика и прикрепленное сообщения работодателя. Реализация метода приведена в листинге 3.8.2.7.

```
    rejectResponse: async function (userId, responseId,
pinnedMessage) {
            const response = await
Repositories.Response.getOne(responseId)
            if (!response) {
                throw new AppError(404, 'Отклик не найден')
            }
            response.is_approved = false
            response.employer_pinned_message = pinnedMessage
            const editedResponse = await
Repositories.Response.edit(responseId, response)
            if (!editedResponse) {
                throw new AppError(500, 'Отклик не был одобрен')
            }
            return editedResponse
        },
```

Листинг 3.8.2.7 – Код метода rejectUserResponse

В функции происходит проверка существования отклика и его дальнейшее изменение(перевод в состояния отклонен). В случае отсутствия отклика в базе

данных возвращаем клиенту ответ с кодом 404. В случае неудачного изменения документа отклика возвращаем ответ с кодом 500.

3.8.2.8 Функция «Просмотр откликов на вакансии»

Метод getResponsesForVacancy сервиса ResponseService предназанчен для просмотр откликов на вакансии, в качестве параметров принимает id пользователя и id вакансии. Реализация метода представлена в листинге 3.8.2.8.

```
    getResponsesForVacancy: async function (userId, vacancyId) {
            if (!await Repositories.Vacancy.isExists(vacancyId)) {
                throw new AppError(404, 'Вакансия не найдена')
            }
            const responses = await
Repositories.Response.getAllForVacancy(vacancyId)
            if (!responses) {
                throw new AppError(404, 'Отклики не найдены')
            }
            return responses
        },
```

Листинг 3.8.2.8 – Код метода getResponsesForVacancy

В методе getResponsesForVacancy происходит проверка на наличие вакансии с vacncyId в базе данных и откликов для нее. Если ничего не найдено, возвращаем ответ с кодом 404. При успешном выполнении функция вернет массив откликов на вакасию.

3.8.2.9 Функция «Редактирование профиля»

В качестве параметров метод editProfile сервиса EmployerService(аналогичный метод editProfile представлен и в ApplicantService) принимает id пользователя и объект профиля с новыми данными. Реализация метода прелдставлена в листинге 3.8.2.9.

```
    editEmployerProfile: async function (userId, profile) {
            const editedEmployer = await
Repositories.Employer.editProfile(userId, profile.name,
profile.contacts)
            if (!editedEmployer) {
                throw new AppError(404, 'Профиль не изменен')
            }
            return editedEmployer
```

```
        },
```

Листинг 3.8.2.9 – Код метода editProfile

Метод обновляет профиль работодателя. В случае ошибки отсутствия в базе данных пользователя с userId возвращает ответ с кодом 404. Иначе возвращает отредактированный профиль.

3.8.2.10 Функция «Просмотр вакансий компании»

Метод getCompanyVacancies сервиса EmployerService предназначен для получения вакансий для компании, в которой состоит работодатель. В качестве параметров принимает id пользователя. Реализация метода представлена в листинге 3.8.2.10.

```
    getCompanyVacancies: async function (userId) {
            const employer = await
Repositories.Employer.getOne(userId)
            if (!employer.company) {
                throw new AppError(400, 'Работодатель не состоит в
компании')
            }
            const vacancies = await
Repositories.Vacancy.getCompanyVacancies(employer.company.company_re
gnum)
            vacancies.map(async (vacancy) => {
                const responses = await
Repositories.Response.getAllForVacancy(vacancy.id)
                vacancy.responses_count = responses.length
            })
            if (!vacancies) {
                throw new AppError(404, 'Вакансии не найдены')
            }
            return vacancies
        },
```

Листинг 3.8.2.10 – Код метода getCompanyVacancies

В методе происходи проверка наличия у работодателя компании, если работодатель без компании, отправялем клиенту ответ с кодом 400. Иначе получаем список вакансий компании и добавляем каждой вакансии в массиве поле responses с массивом откликов на эту вакансию. Если вакансии не найдены, возвращаем ответ клиенту с кодом 404. Иначе возвращаем массив вакансий.

3.8.2.11 Функция «Выход из компании»

Метод leaveCompnay сервиса EmployerService позволяет работодателю выйти из компании. В качестве параметров принимает id пользователя. Реализация метода представлена влистинге 3.8.2.11.

```
     leaveCompany: async function (userId) {
             const          editedEmployer          =          await
Repositories.Employer.editEmployerCompany(userId, null, null)

             if (!editedEmployer) {
                 throw new AppError(500, 'Ошибка выхода работодателя
из компании')
             }
             return editedEmployer
         }
```

Листинг 3.8.2.11 – Код метода leaveCompnay

В методе происходит изменение документа коллекции Employer: поле company становится null, что означает отсутствие компании у работодателя. В случае неудачного изменения документа возвращается ответ клиенту с кодом 500. Иначе возвращается измененный документ коллекции Employer.

3.8.3 Функции пользователя с ролью «Соискатель»

Так как пользователи с ролью «Соискатель» и «Работодатель» имеют одинаковую функцию: редактирование профиля, то описание данной функции будет приведено только в разделе 3.8.2.9.

3.8.3.1 Функция «Создание резюме»

Метод createResume сервиса ResumeService предназаначен для создания резюме, в качестве параметров принимает id пользователя и объект резюме resume. Реализзация метода приведена в листинге 3.8.3.1.

```
     createResume: async function (userId, resume) {
             const            applicant            =            await
Repositories.Applicant.getByUserId(userId)
             resume.applicant_id = applicant.id
             const            newResume            =            await
Repositories.Resume.addResume(resume)
             if (!newResume) {
                 throw new AppError(500, 'Резюме не создано')
```

```
            }
            await          Repositories.Applicant.addResume(userId,
newResume.id)
            return newResume
        },
```

Листинг 3.8.3.1 – Код метода createResume

В методе происходит поиск пользователя по userId. В объекте resume полю applicant_id присваивается значение id соискателя. Затем с помощью метода репозитория ResumeRepository создается новый документ newResume в коллекции Resume. В случае если newResume принимает знаение null отправялем ответ с кодом 500. Иначе возвращаем новый документ.

3.8.3.2 Функция «Изменение резюме»

Метод editResume сервиса ResumeService, используемый для изменения резюме, принимает параметры id пользователя, id резюме и объект резюме с новыми данными резюме. Реализация метода editResume приведена в листинге 3.8.3.2.

```
editResume: async function (userId, resumeId, resume) {
        if (!await Repositories.Resume.isResumeExists(resumeId)) {
            throw new AppError(404, 'Резюме не найдено')
        }
        if    (!await    Repositories.Resume.checkResumeOwner(userId,
resumeId)) {
            throw   new   AppError(403,   'Пользователь   не   является
владельцем резюме')
        }
        const             editedResume             =             await
Repositories.Resume.editResume(resumeId, resume)
        if (!editedResume) {
            throw new AppError(500, 'Резюме не изменено')
        }
        return editedResume
    },
```

Листинг 3.8.3.2 – Код метода editResume

В методе происходит проверка наличия резюме, в случае отсутствия резюме отправляется ответ с кодом 404. Затем происходит проверка владельца резюме, если пользователь с userId не явлется владельцем резюме, то отправляем ответ с 403 кодом. Затем происходит изменение документа коллекции Resume. В случае

неудачного изменения отправляем ответ с кодом 500. Иначе возвращаем измененное резюме.

3.8.3.3 Функция «Удаление резюме»

Метод deleteResume сервиса ResumeService в качестве параметров принимает id пользователя и id резюме, которое надо удалить. Реализация метода deleteResume приведена в листинге 3.8.3.3.

```
deleteResume: async function (userId, resumeId) {
        if (!await Repositories.Resume.isResumeExists(resumeId)) {
            throw new AppError(404, 'Резюме не найдено')
        }
        if    (!await    Repositories.Resume.checkResumeOwner(userId,
resumeId)) {
            throw   new   AppError(403,   'Пользователь   не   является
владельцем резюме')
        }
const               deletedResume               =               await
Repositories.Resume.deleteResume(resumeId)
        if (!deletedResume) {
            throw new AppError(500, 'Резюме не удалено')
        }
await Repositories.Applicant.deleteResume(userId, deletedResume.id)
        return deletedResume
    },
```

Листинг 3.8.3.3 – Код метода deleteResume

В методе происходит проверка существования резюме resumeId, в случае отсутствия резюме в базе данных возвращаем ответ с кодом 404. Иначе проверяем владельца резюме, если пользователь с userId не является владельцем резюме с resumeId, возвращаем ответ с кодом 403. Иначе удаляем резюме и возвращаем удаленное резюме. В случае неудачного удаления резюме отправляем клиенту ответ с кодом 500.

3.8.3.4 Функция «Создание отклика на вакансию»

Метод doResponse сервиса ResponseService предназначен для создания отклика на вакансию, принимает в качестве параметров id пользователя и объект отклика. Реализация метода doResponse представлена в листинге 3.8.3.4.

```
    doResponse: async function (userId, response) {
```

```
        if (!await
Repositories.Vacancy.isExists(response.vacancy_id)) {
                throw new AppError(404, 'Вакансия не найдена')
            }
            const applicantId = await
Repositories.Applicant.getByUserId(userId)
            response.applicant_id = applicantId
            response.employer_pinned_message = null
            response.is_approved = null
            if (await
Repositories.Response.isApplicantAlreadyRespond(applicantId,
response.vacancy_id)) {
                throw new AppError(400, 'Пользователь уже дал отклик
на эту вакансию')
            }
            const newResponse = await
Repositories.Response.add(response)
            if (!newResponse) {
                throw new AppError(500, 'Отклик не был произведен')
            }
            return newResponse
         },
```

Листинг 3.8.3.4 – Код метода doResponse

Изначально в методе doResponse происходит проверка наличия вакансии, id которой указан в объекте отклика response. В случае отсутствия вакансии клиенту возвращается ответ с кодом 404. Иначе происходит проверка на повторный отклик того же пользователя на ту же вакансию. Если проверка не прошла успешно, возвращаем клиенту ответ с кодом 400. Иначе создаем документ в коллекции Response. В случае неудачного создания документа, возвращаем клиенту ответ с кодом 500. Иначе возвращаем новый документ.

3.8.3.5 Функция «Отмена отклика на вакансию»

Метод cancelResponse сервиса ResponseService предназанчен для отмены отклика на вакансию, принимает в качестве параметров id пользователя и id отклика, который надо удалить. Реализация метода cancelResponse представлена в листинге 3.8.3.5.

```
cancelResponse: async function (userId, responseId) {
        if (!await Repositories.Response.isExists(responseId)) {
            throw new AppError(404, 'Отклик не найден')
        }
```

```
        const applicantId = await
Repositories.Applicant.getByUserId(userId)
        const response = await
Repositories.Response.getOne(responseId)
        if (!await
Repositories.Response.isApplicantAlreadyRespond(applicantId,
response.vacancy_id)) {
            throw new AppError(400, 'Пользователь не давал отклик на
эту вакансию')
        }
        const deletedResponse = await
Repositories.Response.delete(responseId)
        if (!deletedResponse) {
            throw new AppError(500, 'Отклик не отменен')
        }
        const deletedId = deletedResponse.id
        return deletedId
    },
```

Листинг 3.8.3.5 – Код метода doResponse

Изначально в методе cancelResponse происходит проверка наличия отклика в базе данных. В случае отсутствия документа в базе данных отправялем клиенту ответ с кодом 404. Затем проверяет существует ли у сосикателя отклик с responseId. В случае если не существует, отправялем клиенту ответ с кодом 400. Иначе удаляем отклик и возвращаем id удаленного отклика. В случае неудачного удаления документа возвращаем клиенту ответ с кодом 500.

3.8.3.6 Функция «Просмотр откликов пользователя»

Метод getResponsedVacancies сервиса VacancyService предназначен для получения откликов, сделанных пользователем. В качестве параметров метод принимает id пользователя. Реализация метода getResponsedVacancies приведена в листинге 3.8.3.6.

```
    getResponsedVacancies: async function (userId) {
            const vacancies = await
Repositories.Vacancy.getResponsedVacancies(userId)
            if (!vacancies) {
                throw new AppError(404, 'Вакансии не найдены')
            }
            return vacancies
        }
```

Листинг 3.8.3.6 – Код метода getResponsedVacancies

В методе происходит получение вакансий, на которые пользователь сделал отклик. Если вакансии не найдены, возвращаем клиенту ответ с кодом 404. Иначе возвращаем массив откликов.

3.8.3.7 Функция «Просмотр кталога вакансий»

Метод getAllVacancies сервиса VacancyService принимает в качестве параметров опции сортировки и фильтрации вакансий. Так что помимо функции «Просмотр кталога вакансий» данный метод выполняет функции «Фильтрации вакансий» и «Сортировки вакансий». Реализация метода getAllVacacies представлена в листинге 3.8.3.7.

```
     getVacancies: async function (options) {
 const { vacancy, min_salary, industry, max_experience, sort } = 
options
             let filter = {}
             if (vacancy) {
                 filter.name = { $regex: vacancy, $options: "i" };
             }
             if (min_salary) {
          filter.salary_amount = { $gte: parseInt(min_salary, 10) };
             }
             if (industry) {
                 filter.industry_id = industry;
             }
             if (max_experience) {
            const maxExpNumber = parseInt(max_experience, 10);
            if (maxExpNumber == 0) {
                filter.$or = [
                { required_experience: { $gte: 0 } }, // Вакансии с 
опытом 0 лет
            { required_experience: null } // Вакансии, где опыт не 
требуется (null)
          ];
                 } else if (maxExpNumber) {
                     filter.required_experience     =     {     $gte: 
maxExpNumber };
                 }
                }
       const vacancies = await Repositories.Vacancy.findVacancies({ 
filter, sort })
```

```
            if (!vacancies) {
                return []
            }
            return vacancies
        },
```

Листинг 3.8.3.7 – Код метода getVacancies

Опции, передаваемые в параметрах метода getVacancies, содержат параметры фильтрации вакансий по названи, зарплате, отрасли и опыту, а также содержат параметры для сортировки вакансий по зарплате и дате создания. В результате работы метода мы получаем массив вакансий.

3.8.4 Функции пользователя с ролью «Администратор»

Так как пользователь с ролью «Администратор» имеет сходие функции с пользователями других ролей: удаление и изменение вакансий; просмотр каталога вакансий. Данные функции будут рассмотрены в разделах 3.8.2.5(удаление вакансии), 3.8.2.4(изменение вакансии) и 3.8.3.7(просмотр каталога вакансий).

3.8.4.1 Функция «Удаление пользователей»

Метод deleteUser сервиса UserService предназначен для удаления пользователя по id передаваемому в параметрах метода. Реализация метода приведена в листинге 3.8.4.1.

```
     deleteUser: async function (userId) {
            if   ((await   Repositories.User.getRole(userId))   ===
'admin') {
                throw new AppError(400, 'Нельзя удалить админа')
            }
            const           deletedUser           =           await
Repositories.User.delete(userId)
            switch (deletedUser.role) {
                case 'applicant': {
                    const         applicant         =         await
Repositories.Applicant.deleteApplicant(userId)
                    await
Repositories.Resume.deleteResumesForApplicant(applicant.id)
                    await
Repositories.Response.deleteResponsesForApplicant(applicant.id)
                    break;
                }
                case 'employer': {
```

```
                    const employer = await
Repositories.Employer.delete(userId)
                    await
Repositories.JoinCompanyRequest.deleteRequestForEmployer(employer.id
)
                    break;
                }
            }
            return deletedUser
        }
```

Листинг 3.8.4.1 – Код метода deleteUser

В методе deleteUser изначально происходит проверка удаляемого пользователя, если пользователь является администратором, возвращаем ответ с кодом 400. Иначе удаляем пользователя с и связанные с ним документы каскадно. В результате возвращаем удаленного пользователя.

3.8.4.2 Функция «Создание отрасли»

Метод addIndustryType сервиса IndustryTypeService предназанчен для создания отрасли и принимает в качестве параметра название отрасли. Реализация метода представлена в листинге 3.8.4.2.

```
    addIndustryType: async function (industryName) {
            if (await
Repositories.IndustryType.isExists(industryName)) {
                throw new AppError(400, 'Отрасль уже существует')
            }
            const addedIndustry = await
Repositories.IndustryType.add(industryName)
            return addedIndustry
        },
```

Листинг 3.8.4.2 – Код метода addIndustryType

В методе перед созданием документа происходит проверка наличия отрасли с таким же названием в базе данных. В случае присутствия в базе данных отрасли с таким же названием возвращаем клиенту ответ с кодом 400. Иначе создаем документ и возвращаем его.

3.8.4.3 Функция «Удаление отрасли»

Метод deleteIndustryType сервиса IndustryTypeService, предназначенный для удаления отрасли, принимает параметр id удаляемой отрасли. Реализация метода представлена в листинге 3.8.4.3.

```
     deleteIndustryType: async function (industryId) {
             const            industry            =            await
Repositories.IndustryType.getOne(industryId)
             if                                                (!await
Repositories.IndustryType.isExists(industry.industry_type)) {
                 throw new AppError(404, 'Отрасль не найдена')
             }
             const          deletedIndustry          =          await
Repositories.IndustryType.delete(industryId)
             return deletedIndustry
         },
```

Листинг 3.8.4.3 – Код метода deleteIndustryType

Перед удалением отрасли происходит проверка существует ли отрасль с заданными id в базе данных. В случае если отрасль отсутствует возвращаем ответ клиенту с кодом 404. Иначе удаляем отрасль и возвращаем удаленный документ.

3.8.4.4 Функция «Одобрение запроса на присоединение к компании»

Метод approveRequest сервиса JoinCompanyRequestService предназначен для одобрения запроса работодателя на присоединение к компании. В качестве параметров принимает id запроса на присоединение. Реализация метода представлена в листинге 3.8.4.4.

```
     approveRequest: async (requestId) => {
             const            request            =            await
Repositories.JoinCompanyRequest.getById(requestId)
             const company_regnum = request.company_regnum
             const company = {}
             try {
                 const                    [company_contacts_response,
company_activity_response,     company_name_response]     =     await
Promise.all([
fetch(`http://egr.gov.by/api/v2/egr/getAddressByRegNum/${company_reg
num}`),
fetch(`http://egr.gov.by/api/v2/egr/getVEDByRegNum/${company_regnum}
`),
```

```
fetch(`http://egr.gov.by/api/v2/egr/getShortInfoByRegNum/${company_r
egnum}`)
                ]);
                if        (!company_contacts_response.ok        ||
!company_activity_response.ok || !company_name_response.ok) {
                    throw   new   AppError(400,   'Неверный   УНП
компании');
                }
                const       company_contacts       =       await
company_contacts_response.json();
                const       company_activity       =       await
company_activity_response.json();
                const        company_name        =        await
company_name_response.json();
                company.boss_contacts = {
                    email: company_contacts[0].vemail,
                    phone:
company_contacts[0].vtels.split(',').map(el => el.trim())
                }
                company.activity                                =
company_activity[0].nsi00114.vnvdnp
                company.name = company_name[0].vfn
                company.company_regnum = company_regnum
            } catch (error) {
                throw new AppError(500, 'Ошибка получения данных о
компании');
            }
            const            employer            =            await
Repositories.Employer.getByEmployerId(request.employer)
            const          editedEmployer          =          await
Repositories.Employer.editEmployerCompany(employer.user,    company,
null)
            const      deletedJoinCompanyRequest      =      await
Repositories.JoinCompanyRequest.delete(request.id)

            return company
        },
```

Листинг 3.8.4.4 – Код метода approveRequest

В данном методе происходит 3 запроса на сторонний ресурс для получения данных о компании, к которой желает присоединиться работодатель. Если один из запросов выполняется неудачно, возвращаем ответ клиенту с кодом 400. Иначе

создаем объект company с данными компании и присваивается полю company документа коллекции Employer. Возвращаем сформированный объект company.

3.8.4.5 Функция «Отклонение запроса на присоединение к компании»

Метод rejectRequest сервиса JoinCompanyRequestService предназначен для отклонения запроса работодателя на присоединение к компании. В качестве параметров принимает id запроса на присоединение. Реализация метода представлена в листинге 3.8.4.5.

```
     rejectRequest: async (requestId) => {
             const             request             =             await
Repositories.JoinCompanyRequest.getById(requestId)
             const             employer             =             await
Repositories.Employer.getByEmployerId(request.employer)
             const           deletedRequest           =           await
Repositories.JoinCompanyRequest.delete(requestId)
             const           editedEmployer           =           await
Repositories.Employer.editEmployerCompany(employer.user, null, null)
             return editedEmployer
         },
```

Листинг 3.8.4.5 – Код метода rejectRequest

В методе rejectRequest происходит ищменение документа коллекции Employer: поле requested_company становится null. Документ коллекции Request удаляется. Возвращаем измененный документ коллекции Employer.

3.9 Описание маршрутов и контроллеров

Так как серверная часть приложения разрабатывается с использованием архитектурного стиля REST, где сервер представляет набор обработчиков HTTP-запросов. Описание маршрутов сервера приведено в таьлице 3.9.1.

Таблица 3.9.1 – Описание маршрутов

№ Функции	Метод запроса	Шаблон маршрута	Контроллер	Метод контроллера
1	POST	/api/guest/login	GuestController	login
2	POST	/api/guest/register	GuestController	register
3	POST	/api/employer/joincompnayrequest	EmployerController	sendJoinCompanyRequest
4	DELETE	/api/employer/joincompnayrequest	EmployerController	cancelJoinCompanyRequest

5	POST	/api/employer/vacancy	EmployerController	createVacancy
6	DELETE	/api/employer/vacancy/:id, /api/admin/vacancy/:id	EmployerController, AdminController	deleteVacancy
7	PUT	/api/employer/vacancy/:id, /api/admin/vacancy/:id	EmployerController, AdminController	editVacancy
8	PUT	/api/employer/response/:id/approve	EmployerController	approveUserResponse
9	PUT	/api/employer/response/:id/reject	EmployerController	rejectUserResponse
10	GET	/api/employer/response/:id	EmployerController	getResponsesForVacancy
11	PUT	/api/employer/profile, /api/applicant/profile	EmployerController, AdminController	editProfile
12	GET	/api/employer/vacancy/company	EmployerController	getCompanyVacancies
13	PUT	/api/employer/leavecompany	EmployerController	leaveCompany
14	POST	/api/applicant/resume	ApplicantController	createResume
15	PUT	/api/applicant/resume/:id	ApplicantController	editResume
16	DELETE	/api/applicant/resume/:id	ApplicantController	deleteResume
17	POST	/api/applicant/response	ApplicantController	doResponse
18	DELETE	/api/applicant/response/:id	ApplicantController	cancelResponse
19	GET	/api/applicant/responsedvacancies	ApplicantController	getResponsedVacancies
20	GET	/api/applicant/vacancy, /api/admin/vacancy	ApplicantController, AdminController	getAllVacancies
21	DELETE	/api/admin/user/:id	AdminController	deleteUser
22	POST	/api/admin/industry	AdminController	addIndustry
23	DELETE	/api/admin/industry/:id	AdminController	deleteIndustry
24	PUT	/api/admin/joincompanyrequest/:id/approve	AdminController	approveRequest
25	PUT	/api/admin/joincompanyrequest/:id/reject	AdminController	rejectRequest

3.10 Реализация клиентской части приложения

Клиентская часть приложения реализована с помощью библиотеки React. Она позволяет применить компонентный подход при создании web-сайтов. Так, страницу можно разделить на несколько отдельных компонентов, которые могут взаимодействовать между собой. В таблице 3.10.1 приведено описание компонентов.

Таблица 3.10.1 – Описание компонентов

Компонент	Описание
AuthForm	Комопнент, содержащий формы авторизации и регистрации.
CompanyResponseItem	Компонент, представляющий элемент списка откликов на вакансию для работодателя.
ErrorNotification	Компонент окошка с уведомлением.
NavBar	Навигационная панель.
ApplicantProfile	Профиль соискателя.
EmployerProfile	Профиль работодателя.
EditProfileModal	Модальное окно для редактирования профиля.
ProtectedRoute	Защищенный маршрут, разрешенный определенным пользователям с определенными ролями.
ResponseItem	Элемент списка откликов для соискателя.
ResponseModal	Модальное окно для создания отклика на вакансию.
CompactResumeItem	Минимизированное резюме с минимальным набором данных в списке резюме в профиле соискателя.
ExperienceList	Содержит список опыта работы в резюме.
FullResumeItem	Полное развернутое резюме со всеми данными.
ResumeSidebar	Боковая панель, в которой открывается полное резюме.
SkillsList	Содержит список навыков в резюме.
UserItem	Элемент списка пользователей для администратора.
AdminVacanciesList	Список вакансий для администратора.
AdminVacancyItem	Элемент списка вакансий для администратора.
CompanyVacanciesList	Список вакансий для компании.
CompanyVacancyForm	Форма используемая для создания и изменения вакансий.
CompanyvacancyItem	Элемент спсика вакансий для компании.
FilterVacancies	Компонент с параметрами фильтрации и сортировки каталога вакансий.
VacanciesList	Каталог вакансий, для соискателя.
Vacancy	Элемент каталога вакансий, для соискателя.
AdminIndustriesPage	Страница с отраслями для админа, где можно дабвлять и удалять отрасли.

AdminUsersPage	Страница со списком пользователей, для администратора.
AdminVacanciesPage	Страница сос списком пользователей, для администратора.
JoinCompanyRequestItem	Элемент списка запросов на присоединение к компании.
JoinCompanyRequestPage	Страница, на которой выводится список запросов на присоединение к компании, для администратора.
CompanyResponsesPage	Страница с откликами на вакансию, для работодателя.
CompanyVacanciesPage	Страница с вакансиями компании, для работодателя.
HomePage	Домашняя страница, начальная.
LoginPage	Страница для авторизации.
RegisterPage	Страница для регистрации.
NotFoundPage	Страница с кодом 404.
ProfilePage	Страница, на которой размещаются компоненты профилея пользователя. Определяет отображение того или иного компонента профиля.
UserResponsesPage	Страница с откликами соискателя.
VacanciesPage	Страница с каталогом вакансий.
VacancyPage	Страница вакансии с полной информацией о вакансии.

Также были разработаны сторы с помощью стейт-менеджера Zustand[17] для упрощения управления состоянием React-приложения. Сторы содержат данные и логику для управления этими данными в одном месте. Описание сторов приведено в таблице 3.10.2.

Стор	Описание
useAdminStore	Предназначен для управления админ-панелью.
useApplicantStore	Предназначен для управления данными соискателя.
useAuthStore	Объединяет логику авторизации и управления профилем.
useEmployerStore	Предназначен для управления данными работодателя.
useErrorsStore	Предназначен для управления ошибками.
useResponsesStore	Предназначен для управления откликами на вакансии.
useResumeStore	Предназначен для управления резюме сиоискателя.
useVacanciesStore	Предназначен для управления вакансиями.

3.11 Файлы dockerfile, docker-compose, файл конфигурации nginx

Содержимое dockerfile серверной части приложения представлено в листинге 3.11.1.

```
    FROM node:20.17.0-alpine3.20 AS production

    WORKDIR /app
    COPY package*.json ./
    RUN npm install --production && \
        npm install -g pm2
    COPY . .
    RUN addgroup -S appgroup && adduser -S appuser -G appgroup &&
chown -R appuser:appgroup /app
    EXPOSE 3000
    USER appuser

    CMD ["pm2-runtime", "start", "app.js"]
```

Листинг 3.11.1 – Серверный dockerfile

Содержимое dockerfile сервера nginx для раздачи статики приложения представлен в листинге 3.11.2.

```
    FROM node:20-alpine3.20 AS build

    WORKDIR /app
    COPY package*.json ./
    RUN npm install
    COPY . ./
    RUN npm run build
    FROM nginx:alpine
    COPY --from=build /app/dist /usr/share/nginx/html
    COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
    EXPOSE 80

    CMD ["nginx", "-g", "daemon off;"]
```

Листинг 3.11.2 – dockerfile сервера nginx

Содержимое файла конфигурации nginx.conf приведено в листинге 3.11.3.

```
    server {
        listen 80;
        server_name localhost;
        location / {
            root /usr/share/nginx/html;
            index index.html index.htm;
            try_files $uri $uri/ /index.html;
```

```
        }
        location /api/ {
            add_header 'Access-Control-Allow-Origin' '*' always;
            add_header  'Access-Control-Allow-Methods'  'GET,  POST,
OPTIONS, PUT, DELETE' always;
            add_header   'Access-Control-Allow-Headers'   'DNT,User-
Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-
Type,Range,Authorization' always;
            add_header   'Access-Control-Expose-Headers'   'Content-
Length,Content-Range' always;
            if ($request_method = 'OPTIONS') {
                add_header 'Access-Control-Allow-Origin' '*';
                add_header   'Access-Control-Allow-Methods'   'GET,
POST, OPTIONS, PUT, DELETE';
                add_header           'Access-Control-Allow-Headers'
'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-
Control,Content-Type,Range,Authorization';
                add_header    'Access-Control-Max-Age'    1728000;
add_header 'Content-Type' 'text/plain; charset=utf-8';
                add_header 'Content-Length' 0;
                return 204;
            }
            proxy_pass http://backend:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
            root /usr/share/nginx/html;
        }
    }
```

Листинг 3.11.3 – Файл конфигурации nginx

Содержимое файла docker-compose.yml приведено в листинге 3.11.4.

```
    services:
      frontend:
        build:
          context: ../frontend
          dockerfile: Dockerfile
        image: shashlyk55/frontend:1.0
        container_name: frontend
        restart: always
```

```
        ports:
          - "5173:80"
        networks:
          - network
      backend:
        build:
          context: ./
          dockerfile: Dockerfile
        container_name: backend
        image: shashlyk55/backend:1.0
        restart: always
        environment:
          - DB_URI=mongodb://db:27017/JSearch_db
        env_file:
          - .env
        ports:
          - "3000:3000"
        networks:
          - network
        depends_on:
          db:
            condition: service_healthy
      db:
        image: mongo:8.0.3-noble
        container_name: db
        healthcheck:
          test:   echo   'db.runCommand("ping").ok'   |   mongosh
localhost:27017 --quiet
          interval: 5s
          timeout: 10s
          retries: 5
        ports:
          - "27017:27017"
        volumes:
          - db_data:/data/db
        networks:
          - network
    networks:
      network:
    volumes:
      db_data:
```

Листинг 3.11.4 – Содержимое docker-compose.yml

3.12 Выводы по разделу

В рамках данного раздела были выполнены следующие задачи:

- реализовано 8 сервисов;
- реализовано 5 контроллеров;
- реализовано 8 репозиториев;
- создано 56 компонентов;
- разработана серверная часть приложения;
- разработана клиентская часть приложения;
- создано 4 файла конфигурации для развертывания приложения в docker.

4 Тестирование веб-приложения

4.1 Функциональное тестирование

В таблице 4.1 представлено описание выполненных ручных тестов[18]. Все тесты имеют позивный сценарий, то есть проверялся сценарий правильного использования приложения. Причина заключается в том, что выполнения всех ручных тестов занимает большое количество времени. Поэтому ручным способом было протестировано только самое необходимое.

Таблица 4.1 – Описание тестов

Тестируемая функция	Последовательность действий	Ожидаемый результат
Авторизация	1. Перейти на страницу авторизации(/login). 2. В поле «Email» ввести логин, существующего пользователя. 3. В поле «Пароль» ввести пароль, существующего пользователя. 4. Нажать на кнопку «Войти».	В браузере: переход на главную страницу (/). В базе данных: без изменений.
Регистрация	1. Перейти на страницу регистрации(/register). 2. В поле «Email» ввести email. 3. В поле «Пароль» ввести пароль. 4. В поле «Имя» ввести имя. 5. Выбрать роль пользователя. 6. Нажать на кнопку «Регистрация».	В браузере: переход на главную страницу(/) В базе данных: добавление в коллекцию User нового документа; добавление в коллекцию Applicant/Employer (в зависимости от выбранной роли) нового документа.
Создание запроса на присоединение к компании	1. Авторизоваться через профиль работодателя, не состоящего в компании и не отправившего запрос на присоединение. 2. Перейти в профиль пользователя(/profile). 3. В поле «Регистрационный номер» ввести УНП компании. 4. Нажать на кнопку «Присоединиться».	В браузере: появляется окно с сообщением об успешной отправке запроса. В базе данных: новый документ в коллекции joincompanyrequests; в документе работодателя поле requested_company устанавливается УНП компании.
Отмена запроса на присоединение к компании	1. Авторизоваться через профиль работодателя, не состоящего в компании и отправившего запрос на присоединение.	В браузере: появляется форма с вводом УНП компании. В базе данных: документ в коллекции

	2. Перейти в профиль пользователя(/profile). 3. Нажать на кнопку «Отменить».	joincompanyrequests удаляется; в документе работодателя поле requested_company устанавливается null.
Выход из компании	1. Авторизоваться через профиль работодателя, не состоящего в компании и отправившего запрос на присоединение. 2. Перейти в профиль пользователя(/profile). 3. Нажать на кнопку «Уйти из компании».	В браузере: появляется форма с вводом УНП компании. В базе данных: в документе работодателя поле requested_company устанавливается null и в поле company устанавливается объект с данными о компании.
Редактирование профиля	1. Авторизоваться через профиль работодателя или соискателя. 2. Перейти в профиль пользователя(/profile). 3. Нажать кнопку «Редактировать». 4. В появившейся форме ввести новые значения данных пользователя. 5. Нажать кнопку «Сохранить»	В браузере: данные в профиле пользователя изменятся. В базе данных: изменение документа в коллекции Users.
Добавление вакансии	1. Авторизоваться через профиль работодателя, состоящего в компании. 2. Перейти в раздел «Вакансии компании». 3. Нажать кнопку «Добавить вакансию». 4. Заполнить появившуюся форму. 5. Нажать кнопку «Сохранить».	В браузере: в список добавится новая вакансия. В базе данных: новый документ в коллекции Vacancies.
Изменение вакансии	1. Авторизоваться через профиль работодателя, состоящего в компании или через администратора. 2. Перейти в раздел «Вакансии компании» («Вакансии» для администратора). 3. Нажать кнопку на против любой вакансии в списке «Редактировать».	В браузере: данные выбранной вакансии изменятся. В базе данных: поля документ в коллекции Vacancies изменят значение.

	4. Изменить значения полей появившейся формы. 5. Нажать кнопку «Сохранить».	
Удаление вакансии	1. Авторизоваться через профиль работодателя, состоящего в компании или через администратора. 2. Перейти в раздел «Вакансии компании» («Вакансии» для администратора). 3. Нажать кнопку на против любой вакансии в списке «Удалить».	В браузере: выбранная вакансия пропадает из списка вакансий. В базе данных: удаляется документ в коллекции Vacancies.
Отклонение отклика пользователя	1. Авторизоваться через профиль работодателя, состоящего в компании. 2. Перейти в раздел «Вакансии компании». 3. Выбрать вакансию из списка. 4. В появившемся списке выбрать отклик. 5. В появившемся окне нажать кнопку «Одобрить».	В браузере: переход на страницу вакансий. В базе данных: поле is_approved документа из коллекции Responses устанавливается в значение false.
Одобрение отлкика пользователя	1. Авторизоваться через профиль работодателя, состоящего в компании. 2. Перейти в раздел «Вакансии компании». 3. Выбрать вакансию из списка. 4. В появившемся списке выбрать отклик. 5. В появившемся окне нажать кнопку «Отклонить».	В браузере: переход на страницу вакансий. В базе данных: поле is_approved документа из коллекции Responses устанавливается в значение true.
Просмотр откликов на вакансию	1. Авторизоваться через профиль работодателя, состоящего в компании. 2. Перейти в раздел «Вакансии компании». 3. Выбрать вакансию из списка.	В браузере: появляется список откликов на вакансию. В базе данных: без изменений.
Созданеи резюме	1. Авторизоваться через профиль соискателя. 2. Перрейти в профиль пользователя(/profile). 3. Нажать кнопку «Создать резюме». 4. Заполнить поля формы.	В браузере: в проифле пользователя в список резюме добавляется новое резюме. В базе данных: в коллекцию Resumes

	5. Нажать кнопку «Сохранить».	добавялется новый документ.
Изменение резюме	1. Авторизоваться через профиль соискателя. 2. Перрейти в профиль пользователя(/profile). 3. Выбрать резюме из списка. 4. Изменить значения полей формы. 5. Нажать кнопку «Сохранить».	В браузере: в проифле пользователя в списке резюме изменятся данные одного резюме. В базе данных: в коллекции Resume изменится один документ.
Удаление резюме	1. Авторизоваться через профиль соискателя. 2. Перрейти в профиль пользователя(/profile). 3. Выбрать резюме из списка. 4. Изменить значения полей формы. 5. Нажать кнопку «Сохранить».	В браузере:из списка резюме удаляется одно резюме. В базе данных: из коллекции Resumes каскадно удаляется один документ.
Создание отклика на вакансию	1. Авторизоваться через профиль соискателя. 2. Перейти в раздел каталога вакансий(/vacancies). 3. Нажать на кнопку «Откликнуться». 4. В появившемся окне выбрать резюме. 5. Нажать кнопку «Отправить»	В браузере: кнопка «Откликнуться» станет неактивной. В базе данных: В коллекцию Responses добавится документ.
Отмена отклика на вакансию	1. Авторизоваться через профиль соискателя. 2. Перейти в раздел «Отклики»(/responsedvacancies). 3. Напротив нужного отклика в списке нажать кнопку «Отменить».	В браузере: отклик пропадет из списка. В базе данных: из коллекции Responses удалится один документ.
Просмотр каталога вакансий	1. Авторизоваться через профиль соискателя. 2. Перейти в раздел каталога вакансий(/vacancies).	В браузере: появится список вакансий. В базе данных: без изменений.
Сортировка вакансий	1. Авторизоваться через профиль соискателя. 2. Перейти в раздел каталога вакансий(/vacancies). 3. В панели справа выбрать опцию сортировки.	В браузере: изменяется порядок следования вакансий в каталоге. В базе данных: без изменений.

	4. Нажать кнопку «Применить».	
Фильтрация вакансий	1. Авторизоваться через профиль соискателя. 2. Перейти в раздел каталога вакансий(/vacancies). 3. В панели справа выбрать опции фильтрации. 4. Нажать кнопку «Применить».	В браузере: фильтрация каталога вакансий. В базе данных: без изменений.
Удаление пользователей	1. Авторизоваться через профиль администратора. 2. Перейти во вкладку «Пользователи»(/users). 3. Напротив пользователя нажать кнопку «Удалить».	В браузере: запись пользователя пропадает из списка. В базе данных: из коллекции Users каскадно удаляется документ.
Создание отрасли	1. Авторизоваться через профиль администратора. 2. Перейти во вкладку «Отрасли»(/industries). 3. Ввести название новой отрасли в поле. 4. Нажать кнопку «Доабавить»	В браузере: В базе данных:
Удаление отрасли	1. Авторизоваться через профиль администратора. 2. Перейти во вкладку «Отрасли»(/industries). 3. Напротив отрасли нажать кнопку «Удалить».	В браузере: отрасль пропадает из списка. В базе данных: из коллекции industryTypes удаляется один документ.
Одобрение запроса на присоединение к компании	1. Авторизоваться через профиль администратора. 2. Перейти в раздел «Запросы на присоединение к компании». 3. Напротив нужного запроса нажать кнопку «Принять».	В браузере: запрос пропадет из списка. В базе данных: из коллекции JoinCompanyRequests удалится один документ; документу из коллекции Employers поле company заполнит объект с данными о компании, а поле requested_company станет null.
Отклонение запрос на присоединение к компании	1. Авторизоваться через профиль администратора. 2. Перейти в раздел «Запросы на присоединение к компании».	В браузере: запрос пропадет из списка. В базе данных: из коллекции

	3. Напротив нужного запроса нажать кнопку «Принять».	JoinCompanyRequests удалится один документ; документу из коллекции Employers поле company заполнит объект с данными о компании, а поле requested_company станет null.

4.2 Автоматизированнео тестирование

Одной из целей автоматизированного тестирования[19] является проверка того, что новый функционал не нарушат работу существующего функционала. Также, тесты этого типа выполняется гораздо быстрее, чем ручные тесты. А многократное выполнение тестов позволяет выявлять ошибки на ранних этапах.

Для тестирования будет использоваться библиотека Jest v29.7.0[20]. Будут реализованы модульные тесты[21] репозиториев и сервисов. При модульном тестировании проверяется работа отдельного компонента компонента программы.

В тестах используется такая библиотека как mongo-memory-server v10.1.4[22] предоставляющая возможность создания временной базы данных mongodb в оперативной памяти. После завершения тестов база данных удаляется.

Прежде чем начать тестировать необходимо настроить тестовое окружение. Пример настройки тестовго окружения приведен в листинге 4.1.

```
beforeAll(async () => {
    mongoServer = await MongoMemoryServer.create();
    const uri = mongoServer.getUri();
    await mongoose.connect(uri);
});
afterAll(async () => {
    await mongoose.disconnect();
    await mongoServer.stop();
});
beforeEach(async () => {
    await mongoose.connection.db.dropDatabase();
});
```

Листинг 4.1 – Настройка тестового окружения

К примеру в настройках тестовго окружения можно задавать установку и разрыв соединения с базой данных.

В листинге 4.2 приведен код тестирования метода добавления документа из репозитория UseryRepository.

```
     const testUser = {
             contacts: { email: 'test@example.com' },
             name: 'Test User',
             role: 'applicant',
             password_hash: 'hashed_password'
         };
         describe('add', () => {
             it('should add a new user to the database', async () =>
{
                 const newUser = await userRepository.add(testUser);
                 const          userInDb          =          await
Models.User.findById(newUser._id);
                 expect(userInDb).not.toBeNull();

expect(userInDb.contacts.email).toBe(testUser.contacts.email);

expect(userInDb.contacts.phone).toBe(testUser.contacts.phone);
                 expect(userInDb.name).toBe(testUser.name);

expect(userInDb.password_hash).toBe(testUser.password_hash);
             });
         });
```

Листинг 4.2 – Тестирование метода из UserRepository

Вызываем тестируемый метод добавления пользователя в базу данных, а затем проверяем есть ли такой пользователь в базе данных.

В листинге 4.3 представлен тест метода addVacancy сервиса VacancyService, который добавляет вакансию.

```
describe('addVacancy', () => {
        it('should add new vacancy for employer with company', async
() => {
            const mockEmployer = { company: { company_regnum: '123'
} };
            const mockVacancy = { _id: '1', name: 'Developer' };
Repositories.Employer.getOne.mockResolvedValue(mockEmployer);

Repositories.Vacancy.add.mockResolvedValue(mockVacancy);

            const result = await VacancyService.addVacancy('user1',
{ name: 'Developer' });
                expect(result).toEqual(mockVacancy);
             });
```

Листинг 4.3 – Тестирование метода addVacancy из VacancyService

Результат выполнения тестовых методов, тестирующих методы репозиториев и сервисов, приведены на рисунке 4.1.

```
PASS  tests/service/resume.service.test.js


PASS  tests/service/industryType.service.test.js
PASS  tests/service/employer.service.test.js
PASS  tests/service/vacancy.service.test.js
PASS  tests/service/response.service.test.js
PASS  tests/service/applicant.service.test.js
PASS  tests/service/user.service.test.js
PASS  tests/repository/industryType.repository.test.js
PASS  tests/repository/joinCompanyRequest.repository.test.js
PASS  tests/repository/resume.repository.test.js
PASS  tests/repository/employer.repository.test.js
PASS  tests/repository/vacancy.repository.test.js
PASS  tests/repository/applicant.repository.test.js
PASS  tests/repository/user.repository.test.js
PASS  tests/repository/response.repository.test.js
PASS  tests/service/joinCompanyRequest.service.test.js

Test Suites: 16 passed, 16 total
Tests:       180 passed, 180 total
Snapshots:   0 total
Time:        5.295 s
Ran all test suites.
```

Рисунок 4.1 – Результат выполнения всех тестов

По результатам тестирования приложения видно, что бизнес логика приложения работает исправно.

4.3 Вывод по разделу

В рамках данного раздела выполнено следующее:

- выполнено 25 ручных тестов;
- создано и успешно выполнено180 автоматизированных тестов;
- протестировано 8 репозиториев;
- протестировано 8 сервисов.

5 Руководство программиста

5.1 Скачивание и установка

В данном разделе представлена инструкция по развертыванию и проверке работы веб-приложения.

Для сборки образов в контейнеры и объединения их с помощью docker compose на устройстве понадобится Docker Engine v20.10+ и Docker Compose v2.0+. Образы находятся в репозитории DockerHub. Так как файл docker-compose.yml в репозиторий DockerHub поместить нельзя, он находится в репозитории GitHub проекта.

Команды для загрузки docker-compose.yml приведены в листинге 5.1.

```
git clone https://github.com/shashlyk55/jsearch-app.git
cd jsearch-app
```

Листинг 5.1 – Команды для скачивания docker-compose.yml

После скачивания репозитория с GitHub выполняем команду для скачивания образов с DockerHub, сборки и запуска контейнеров. Команда приведена в листинге 5.2.

```
Docker compose pull
docker compose up -d
```

Листинг 5.2 – Команды для скачивания образов, сборки и запуска контейнеров

5.2 Проверка работоспособности приложения

Для проверки работоспособности приложения проверим все ли контейнеры запущены успешно. Команда вывода состояний контейнеров представлена в листинге 5.3.

```
docker-compose ps
```

Листинг 5.3 – Команда вывода состояния контейнеров

В лучшем случае статус всех контейнеров должен быть Up.

Следующим шагом проверки работоспособности приложения будет проверка функционирования его отдельных компонентов. Команды и ожидаемые результаты приведены в таблице 5.1.

Таблица 5.1 – Команды для проверки работоспособности

Команда	Ожидаемй результат

Проверка работоспособности nginx: `ps aux | grep nginx`	Вывод в консоль: «2025/05/21 22:52:48 [notice] 1#1: start worker processes 2025/05/21 22:52:48 [notice] 1#1: start worker process 29»
Проверка работоспособность express сервера: pm2 list	Статус сервера должен быть «online»

Для проверки подключения к базе данных требуется проверить логи express сервера. В случае успешного подключения к базе данных в консоли будет сообщение: «connected to db».

Если все вышеописанные проверки пройдены успешно, то можно пробовать регистрироваться в приложении.

Также необходимо первого зарегистрировавшегося пользователя сделать администратор приложения. Для этого необходимо в базе данных сменить его роль на роль «admin»

5.3 Заключение по разделу

В данном разделе была приведена инструкция по скачиванию и установке веб-приложения. Также была приведена инструкция по проверке работоспособности приложения.

Заключение

В ходе выполнения курсового проекта было реализовано веб-приложение для трудоустройства. Пользователи данного приложения делятся на 4 роли: гость, соискатель, работодатель, администратор. Каждая из ролей имеет свой функционал. Всего в приложении было реализовано 25 функций.

В качетве зранилища данных использовалась нереляционная база данных MongoDb, для которой создано 8 коллекций.

Серверная часть приложения разработана с использованием фреймворка Express.js и нескольких дополнительных библиотек. Проект серверного приложения разделен на модели, репозитории, сервисы и контроллеры. Количество строк кода составило 2600.

Клиентская часть веб-приложения реализована с использованием библиотеки React.js. Всего разработано 40 компонентов и 8 сторов для клиентской части. Количество строк кода составило 4100.

Для взаимодействия клиента и сервера используется сервер nginx, которые используется для раздачи статических файлов или проксирования запросов на сервер express.

Также веб-приложение было упаковано в набор состоящий из 3-х docker-контейнеров, объединенных docker-compose.

Выполнено ручное и автоматизированное тестирование приложения. Для автоматизированного тестирования было разработано 180 тестовых методов. Для ручного тестирования было разработано 25 тестовых сценариев.

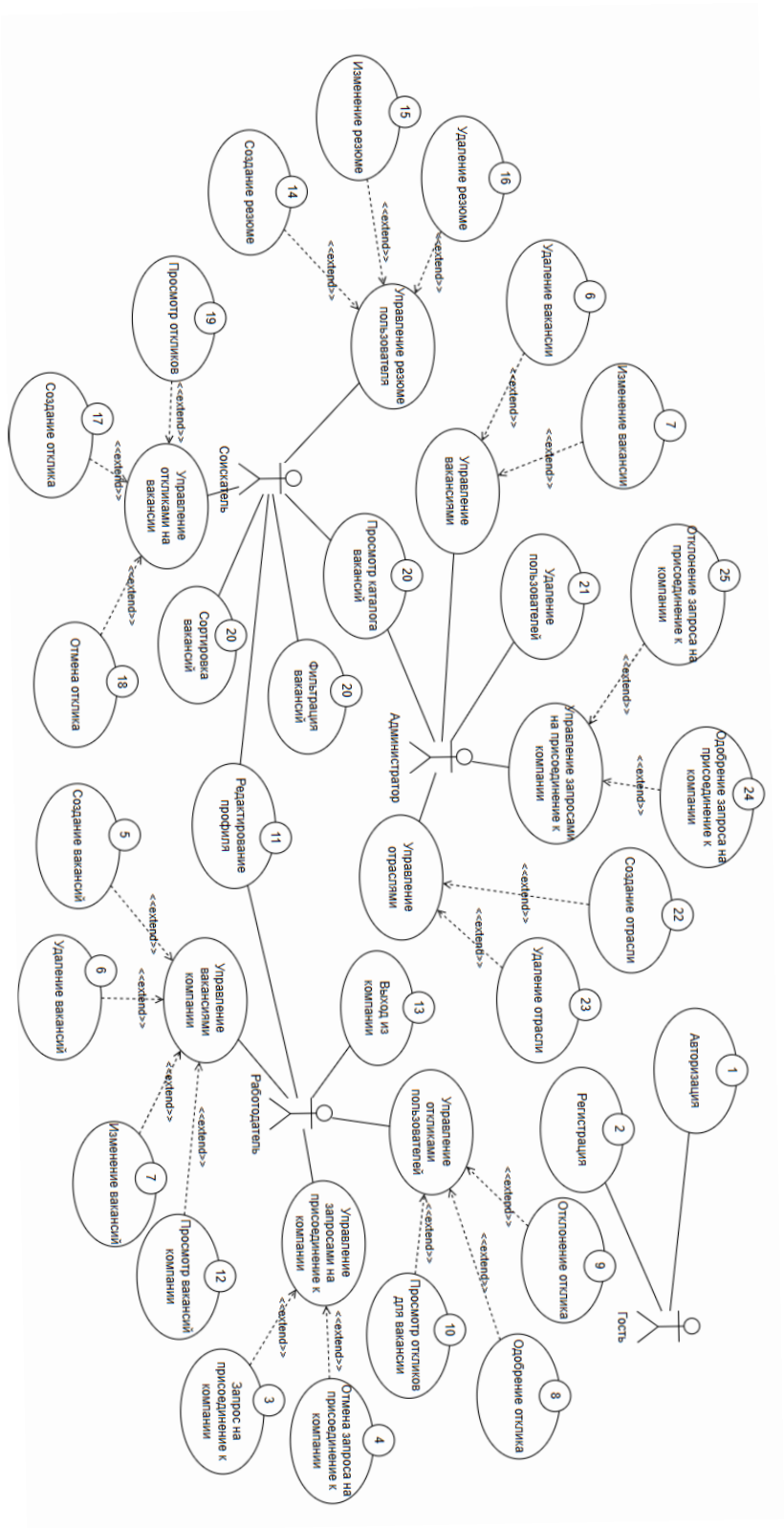
Список использованных источников

1 Node.js [Электронный ресурс]. – Режим доступа: https://nodejs.org/docs/latest/api/,
2 Express.js [Электронный ресурс]. – Режим доступа: https://expressjs.com/ru/,

3 Mongoose v8.15.0 [Электронный ресурс]. – Режим доступа: https://mongoosejs.com/docs/,
4 MongoDB [Электронный доступ]. – Режим доступа: https://www.mongodb.com/docs/,
5 React.js v19.0.0 [Электронный ресурс]. – Режим доступа: https://react.dev/,
6 Функциональные требования приложений [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/funktsionalnye-i-nefunktsionalnye-trebovaniya-k-po/,
7 UML диаграмма вариантов использования [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/566218/,
8 Nginx v1.27.5 [Электронный ресурс]. – Режим доступа: https://nginx.org/ru/,
9 HTTP 1.1 [Электронный ресурс]. – Режим доступа: https://datatracker.ietf.org/doc/html/rfc2616,
10 TCP [Электронный ресурс]. – Режим доступа: https://sky.pro/wiki/sql/chto-takoe-tcp-soedinenie/,
11 Docker [Электронный ресурс]. – Режим доступа: https://www.docker.com/,
12 ОС Alpine Linux 3.20 [Электронный доступ]. – Режим доступа: https://alpinelinux.org/,
13 Docker Compose [Электронный доступ]. – Режим доступа: https://docs.docker.com/compose/,
14 ODM [Электронный ресурс]. – Режим доступа: https://www.dctacademy.com/blog/what-is-object-document-mapper-odm,
15 EGR [Электронный ресурс]. – Режим доступа: https://egr.gov.by/egrmobile/providing-information/api,
16 Паттерн репозиторий [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/248505/,
17 Zustand v5.0.3 [Электронный ресурс]. – Режим доступа: https://zustand.docs.pmnd.rs/getting-started/introduction,
18 Ручное тестирование [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/chto-takoe-ruchnoe-testirovanie/,
19 Автоматизированное тестирование [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/avtomatizirovannoe-testirovanie/,
20 Jest v29.7.0 [Электронный ресурс]. – Режим доступа: https://jestjs.io/,
21 Модульное тестирование [Электронный ресурс]. – Режим доступа: https://logrocon.ru/news/unit_testing,
22 Mongo-memory-server v10.1.4 [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/mongodb-memory-server.

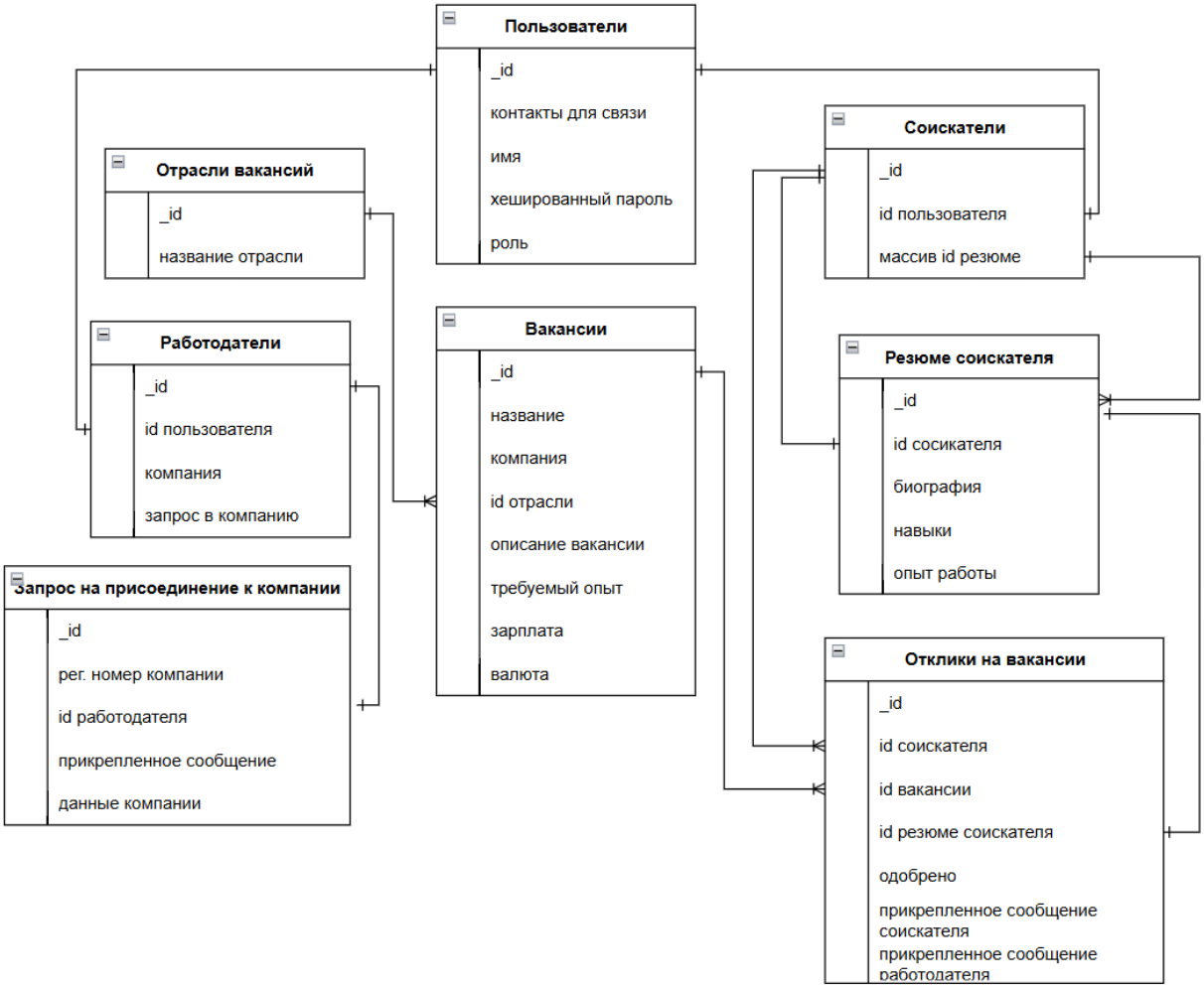
Приложение А

Диаграмма вариантов использования


Авторизация
1
Регистрация
2
Отклонение отклика
9
Одобрение отклика
8
Гость
Запрос на присоединение к компании
3
Отмена запроса на присоединение к компании
4
Просмотр откликов для вакансии
10
Просмотр вакансий компании
12
Изменение вакансий
7
Удаление вакансий
6
Создание вакансий
5
Управление вакансиями компании
Управление запросами на присоединение к компании
Управление откликами пользователей
Редактирование профиля
11
Выход из компании
13
Работодатель
Удаление отрасли
23
Создание отрасли
22
Управление отраслями
Одобрение запроса на присоединение к компании
24
Отклонение запроса на присоединение к компании
25
Управление запросами на присоединение к компании
Удаление пользователей
21
Просмотр каталога вакансий
20
Фильтрация вакансий
20
Управление вакансиями
Администратор
Удаление вакансии
6
Изменение вакансии
7
Удаление резюме
16
Изменение резюме
15
Создание резюме
14
Управление резюме пользователя
Просмотр откликов
19
Создание отклика
17
Отмена отклика
18
Сортировка вакансий
20
Управление откликами на вакансии
Соискатель
<<extend>>

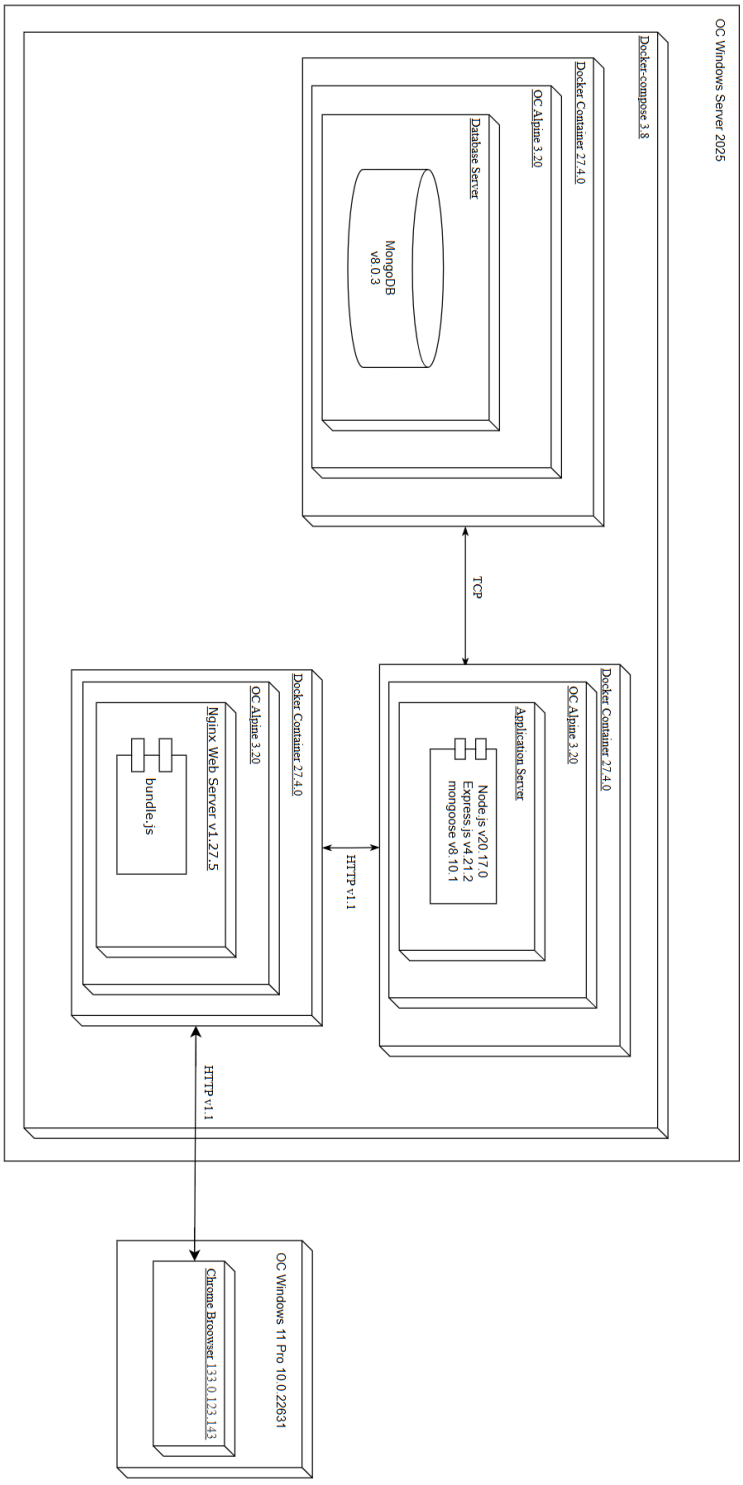
Приложение Б

Логическая схема базы данных



Приложение В

Схема развертывания веб-приложения


OC Windows Server 2025
Docker-compose 3.8
Docker Container 27.4.0
OC Alpine 3.20
Database Server
MongoDB
v8.0.3
TCP
Docker Container 27.4.0
OC Alpine 3.20
Application Server
Node.js v20.17.0
Express.js v4.21.2
mongoose v8.10.1
HTTP v1.1
Docker Container 27.4.0
OC Alpine 3.20
Nginx Web Server v1.27.5
bundle.js
HTTP v1.1
OC Windows 11 Pro 10.0.22631
Chrome Broowser 133.0.123.143

Приложение Г

Скрипт создания базы данных

```
use("JSearch_db");

db.createCollection("User");

db.createCollection("Applicant");

db.createCollection("Resume");

db.createCollection("Employer");

db.createCollection("IndustryType");

db.createCollection("JoinCompanyRequest");

db.createCollection("Response");

db.createCollection("Vacancy");
```