

Содержание

Введение	3
1 Постановка задачи	4
1.1 Обзор аналогичных решений	4
1.2 Спецификация требований	6
2 Проектирование web-приложения	7
2.1 Проектирование вариантов использования	7
2.2 Структура web-приложения	8
2.3 Проектирование базы данных	8
2.4 Проектирование сервера web-приложения	13
3 Разработка web-приложения	19
3.1 Разработка бэкэнда	19
3.2 Разработка фронтэнда	22
4 Тестирование web-приложения	25
5 Руководство пользователя	28
Заключение	34
Список используемых источников	35
Приложение А	36

Введение

В современном мире поиск и предложение работы становятся все более актуальными, поскольку технологические инновации меняют традиционные способы трудоустройства. Сегодня множество людей стремятся найти и предложить работу через специализированные приложения, которые облегчают этот процесс и создают связь между работодателями и соискателями.

Целью курсового проектирования является разработка распределённого web-приложения, предоставляющего удобную и эффективную платформу для взаимодействия между работодателями и соискателями. Приложение должно позволять соискателям создавать профили и свои резюме и представлять свои навыки, образование и опыт работы. Также приложение должно предоставлять соискателям возможность просматривать доступные вакансии и отправлять свои заявки на эти вакансии. Приложение должно предоставлять работодателям возможность размещать вакансии и просматривать профили и резюме откликнувшихся кандидатов. Помимо базовых функций поиска и предложения работы, приложение должно предоставлять возможность оценивать и оставлять отзывы о работодателях и соискателях.

В приложении должны быть разграничены возможности гостей, соискателей, работодателей и администраторов. Так, гости должны иметь возможность только просматривать публичные вакансии, страницы компаний и отзывы на них; соискателям доступно изменение своих персональных данных, изменение списка резюме и возможность отклика на доступные вакансии; работодатели изменять список своих вакансий, просматривать список откликов, принимать и отклонять их; работодателям и соискателям должна быть доступна возможность оставлять отзывы друг на друга, но только при условии, что соискатель откликнулся на вакансию работодателя, и последний принял отклик; администраторам возможности по удалению учётных записей, подтверждению статусов компаний и удалению комментариев.

В ходе выполнения курсового проектирования будут решены следующие задачи:

- анализ литературы по теме работы;
- изучение требований и определение вариантов использования;
- анализ и проектирование архитектуры приложения, модели базы данных;
- тестирование приложения;
- создание руководства пользователя.

1 Постановка задачи

1.1 Обзор аналогичных решений

В качестве первого аналогичного решения была рассмотрена платформа Fiverr, на которой предлагаются услуги фрилансеров со всего мира. Она специализируется на небольших задачах, а не на поиске постоянного места работы. Пример страницы Fiverr представлен на рисунке 1.1.

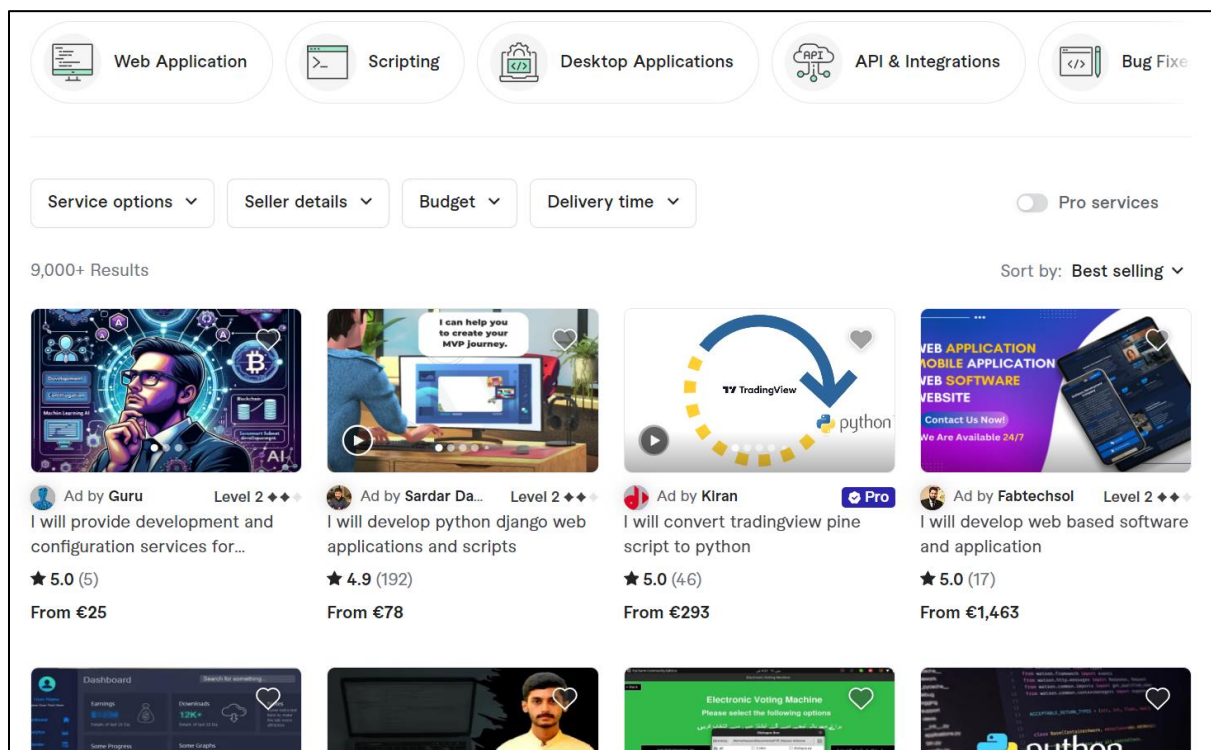


Рисунок 1.1 – Страница Fiverr

Пользователи могут найти исполнителей по различным областям, таким как дизайн, письменное творчество, программирование и многое другое. Особенностью Fiverr является ориентация на заказчиков, то есть соискатели должны размещать свои резюме, а заказчики должны выбирать из имеющихся на данный момент соискателей.

Преимуществом платформы являются простой процесс заказа, возможность просмотра портфолио и отзывов о соискателях. К недостаткам относится недостаточная возможность фильтрации заказов и портфолио: платформа разбивает их по категориям, и осуществлять фильтрацию внутри категорий можно только по нескольким признакам, таким как время выполнения и уровень навыков исполнителя.

В качестве второго аналогичного решения была рассмотрена платформа Upwork, предлагающая более широкий спектр услуг и проектов для фрилансеров. Платформа предоставляет доступ к множеству заказов в различных областях, включая разработку ПО, маркетинг и так далее. Upwork обеспечивает более сложный процесс поиска и найма, включая возможность просмотра профилей, проведения собеседований и оценки работы. Также на платформе присутствует

система обратной связи и защиты платежей. Пример страницы платформы представлен на рисунке 1.2.

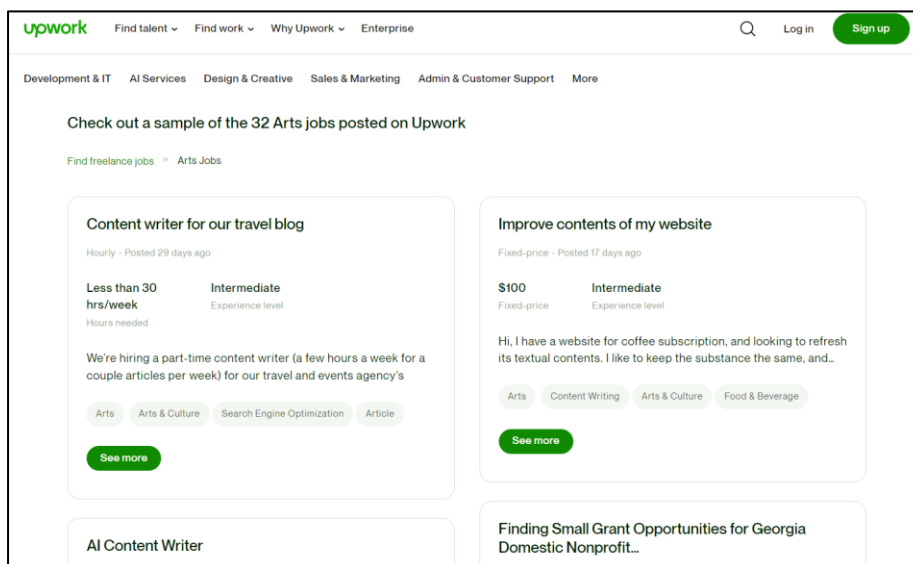


Рисунок 1.2 – Страница Upwork

К преимуществам платформы относится возможность размещения как вакансий, так и публичных резюме, а также возможность предлагать работу как на основе проектной занятости, так и на полный рабочий день. К недостаткам относится недостаточно интуитивная навигация по сайту и отсутствие фильтров, кроме общего направления работы.

В качестве последнего аналогичного решения была рассмотрена платформа HeadHunter – платформа с мощным функционалом по поиску работы и работников в странах СНГ. Пример страницы сайта представлен на рисунке 1.3.

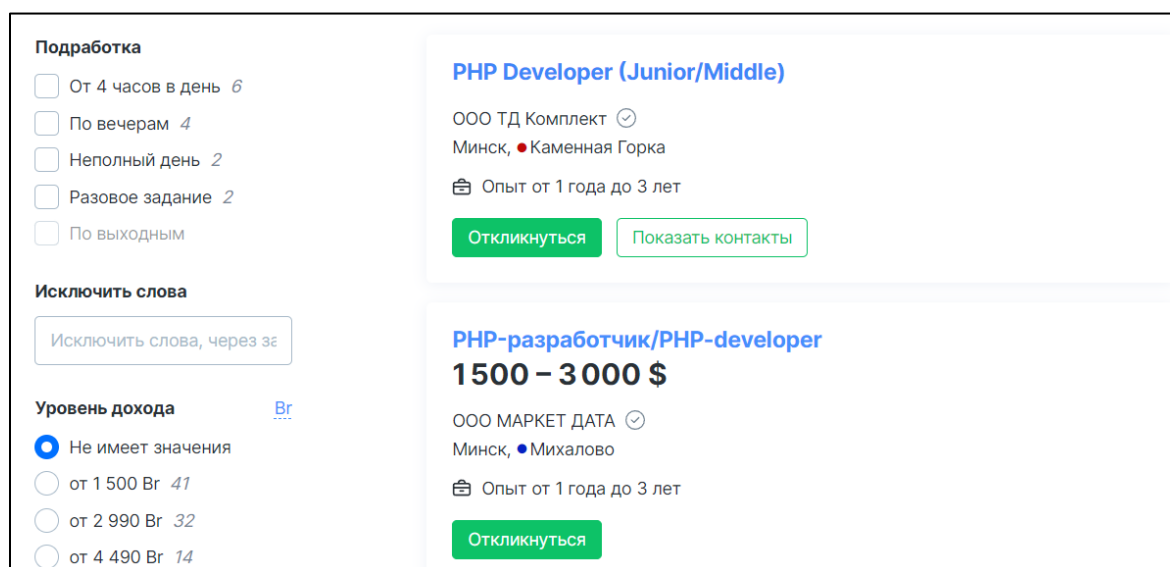


Рисунок 1.3 – Страница Upwork

Она предлагает широкий функционал по размещению вакансий и резюме, а также обширные возможности фильтрации по множеству критериев. Для связи

соискателя и работодателя предусмотрен чат.

Недостатком всех рассмотренных аналогов является невозможность узнать, была ли изменена вакансия, без перезагрузки страницы. В худшем случае соискатель может откликнуться не на ту вакансию, на которую собирался, так как работодатель изменил её, пока соискатель находился на её странице.

1.2 Спецификация требований

На основе рассмотренных аналогичных решений были сформированы следующие требования к программному продукту:

- возможность размещать, изменять и удалять вакансии;
- возможность создавать, изменять и удалять резюме;
- возможность откликаться на вакансии и отзывать отклики;
- возможность принимать и отклонять отклики;
- ограничение на тип размещаемых вакансий для неподтверждённых работодателей;
- возможность оценивать соискателей и работодателей;
- автоматическое обновление данных о просматриваемой вакансии в случае её изменения.

Всем пользователям, включая гостей, должна быть доступна возможность просматривать публичные вакансии, страницы компаний и отзывы о них. Также всем пользователям должна быть доступна возможность жаловаться на отзывы о компаниях.

Соискателю должна быть доступна возможность добавлять резюме, изменять и удалять их, откликаться и удалять свои отклики на публичные вакансии и оставлять отзывы на работодателей, которые приняли его отклик.

Работодателю должна быть доступна возможность добавлять вакансии, изменять и удалять их, получать список откликов на свои вакансии, принимать и отклонять их, просматривать данные о соискателях, откликнувшихся на его вакансии, просматривать отзывы о них и оставлять отзывы о тех соискателях, чьи отклики он принял и запрашивать подтверждение компании у администратора.

Работодателям и соискателям должна быть доступна возможность редактировать данные о себе, менять пароль, удалять свои отзывы и запрашивать удаление своей учётной записи.

Администратору должна быть доступна возможность блокировать возможность пользователя, оставлять отзывы о другом пользователе, удалять комментарии и учётные записи и подтверждать компании.

Администратору не должны быть доступны возможности соискателей и работодателей, такие как создание, изменение и удаление резюме и вакансий, отправка и удаление откликов, изменение персональных данных, так как данные возможности не нужны администратору.

2 Проектирование web-приложения

2.1 Проектирование вариантов использования

Согласно сформулированным требованиям была создана диаграмма вариантов использования, представленная на рисунке 2.1.

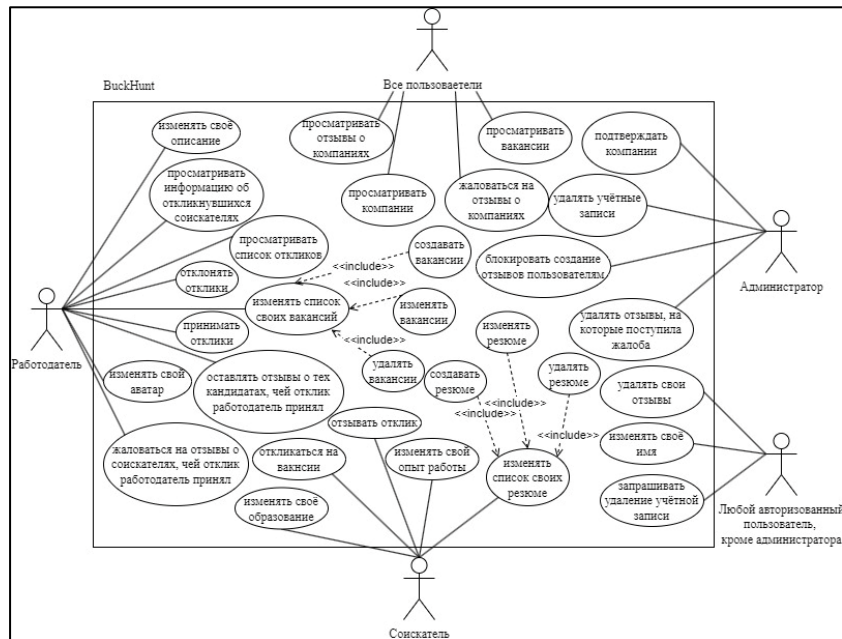


Рисунок 2.1 – Диаграмма вариантов использования

На данной диаграмме под любым пользователем, кроме администратора понимается соискатель или работодатель. Все действия, доступные гостю, доступны также всем остальным пользователям.

Разным категориям пользователей доступны разные действия. Так, соискатели могут изменять список своих резюме, что включает создание новых резюме и изменение и удаление существующих. Также соискатели могут откликаться на вакансии работодателей и отзываться свои отклики, изменять своё образование и опыт работы.

В свою очередь работодатели могут изменять список своих вакансий, что включает в себя создание новых вакансий и изменение и удаление существующих. Работодатели также могут изменять параметр активности вакансии, делая её видимой для всех пользователей или недоступной для просмотра. Также работодатели могут просматривать отклики на свои вакансии и принимать или отклонять их. Работодатель может оставить отзыв о соискателе, который отправил отклик на вакансию работодателя, и данный отклик был принят. Работодатели могут просматривать публичную информацию о соискателях, которые отправили отклики на любую из вакансий работодателя.

Администраторам доступны возможности по блокировке возможности пользователей оставлять отзывы о других пользователях и удалении жалоб на отзывы. Также они могут принимать и отклонять различные запросы пользователей: запросы на удаление учётной записи и запросы на подтверждение компании.

2.2 Структура web-приложения

Обобщённая структура web-приложения представлена на рисунке 2.2.

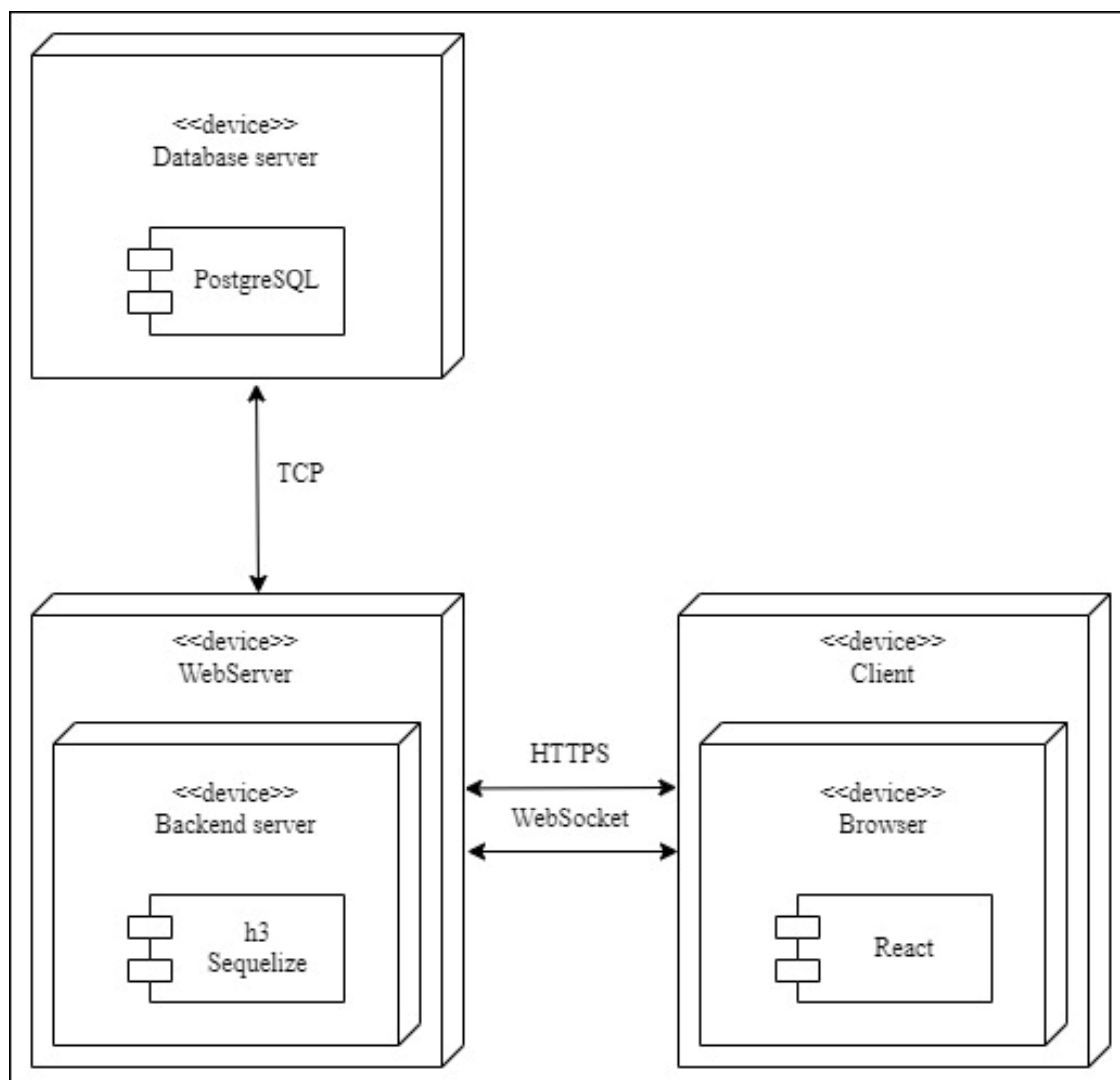


Рисунок 2.2 – Диаграмма развёртывания web-приложения

Согласно данной схеме, клиент и сервер находятся на разных устройствах. Клиент использует браузер для отправки запросов к web-серверу. Клиент и сервер могут обменяться сообщениям по протоколам HTTPS и WebSocket.

Web-сервер обрабатывает запросы при помощи сервера, созданного при помощи фреймворка h3. В случае необходимости сервер отправляет запросы к базе данных, находящейся на отдельном устройстве в контейнере Docker и находящейся под управлением СУБД PostgreSQL.

Для выполнения запросов к базе данных используется ORM Sequelize.

2.3 Проектирование базы данных

Согласно схеме вариантов использования была создана база данных. Её логическая схема представлена на рисунке 2.3. Описание моделей представлено в приложении А.

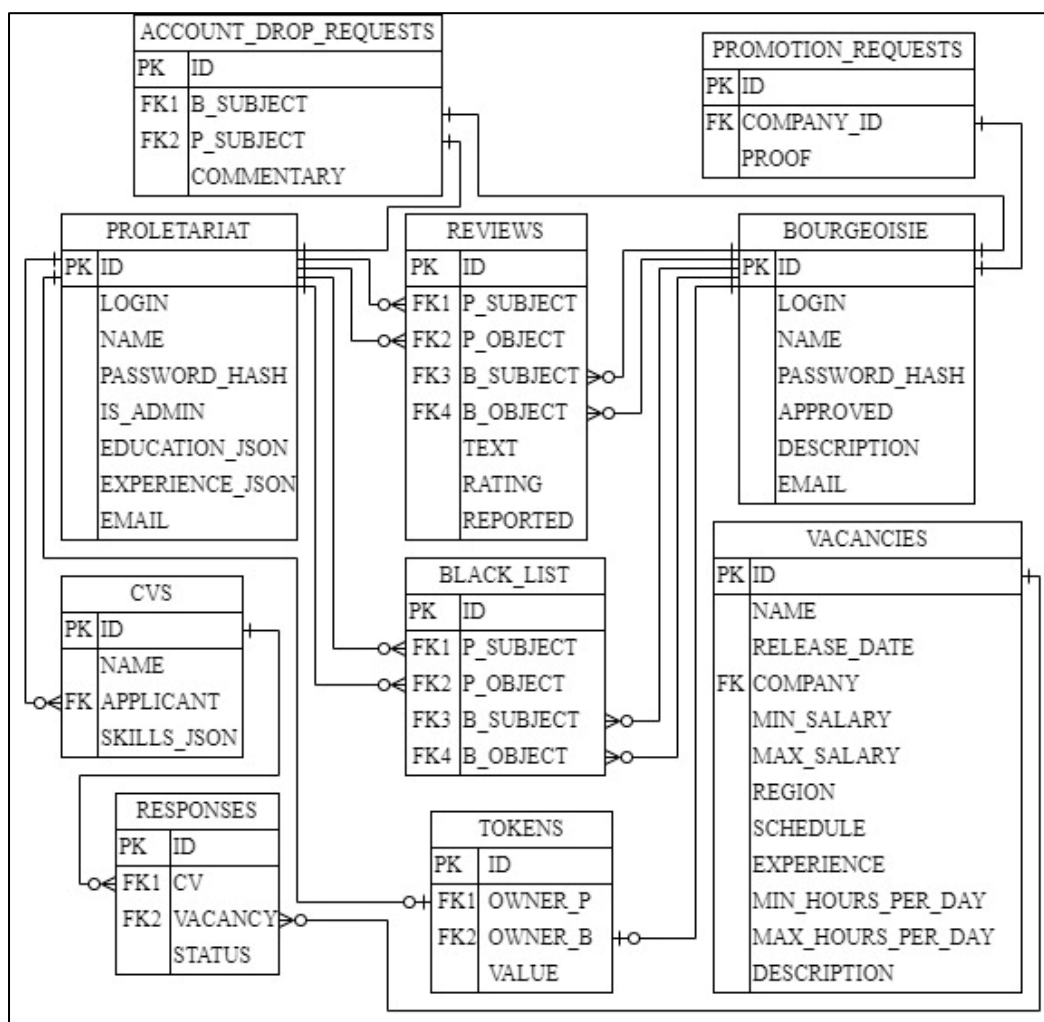


Рисунок 2.3 – Логическая схема базы данных

База данных содержит десять таблиц, хранящих информацию о пользователях, резюме, вакансиях, откликах и прочих данных. Типы данных были выбраны согласно документации [1].

Таблица PROLETARIAT хранит информацию о соискателях. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы PROLETARIAT

Название столбца	Тип данных	Описание
id	integer	идентификатор соискателя, первичный ключ
login	varchar(20)	логин соискателя
name	varchar(50)	имя соискателя
password_hash	varchar(60)	хеш пароля соискателя
is_admin	char(1)	является ли пользователь администратором
education_json	varchar(200)	образование соискателя в формате JSON
experience_json	varchar(500)	опыт работы соискателя в формате JSON
email	varchar(30)	адрес электронной почты соискателя

Таблица CVS хранит информацию о резюме соискателей. Описание её столбцов представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы CVS

Название столбца	Тип данных	Описание
id	integer	идентификатор резюме, первичный ключ
name	varchar(30)	название резюме
applicant	integer	идентификатор соискателя, внешний ключ
skills_json	varchar(100)	массив навыков в формате JSON

Таблица BOURGEOISIE хранит информацию о работодателях. Описание её столбцов представлено в таблице 2.3.

Таблица 2.3 – Описание таблицы BOURGEOISIE

Название столбца	Тип данных	Описание
id	integer	идентификатор работодателя, первичный ключ
login	varchar(20)	логин работодателя
name	varchar(70)	имя работодателя
password_hash	varchar(60)	хеш пароля работодателя
approved	char(1)	показатель того, подтверждена ли компания
description	varchar(2000)	описание работодателя
email	varchar(30)	адрес электронной почты работодателя

Таблица VACANCIES хранит информацию о вакансиях. Описание её столбцов представлено в таблице 2.4.

Таблица 2.4 – Описание таблицы VACANCIES

Название столбца	Тип данных	Описание
id	integer	идентификатор вакансии, первичный ключ
name	varchar(30)	название вакансии
release_date	date	дата публикации вакансии
company	integer	идентификатор компании, внешний ключ
active	char(1)	показатель того, доступна ли вакансия публично
min_salary	integer	минимальная зарплата
max_salary	integer	максимальная зарплата
region	varchar(20)	регион поиска соискателей
schedule	integer	график работы
experience	integer	требуемый опыт работы
min_hours_per_day	integer	минимальное количество рабочих часов в неделю
max_hours_per_day	integer	максимальное количество рабочих часов в неделю
description	varchar(1000)	описание вакансии

Таблица RESPONSES хранит информацию об откликах на вакансии. Описание её столбцов представлено в таблице 2.5.

Таблица 2.5 – Описание таблицы RESPONSES

Название столбца	Тип данных	Описание
id	integer	идентификатор отклика, первичный ключ
cv	integer	идентификатор резюме, внешний ключ
vacancy	integer	идентификатор вакансии, внешний ключ
status	char(1)	показатель статуса отклика

Таблица REVIEWS хранит информацию об отзывах пользователей друг на друга. Описание её столбцов представлено в таблице 2.6.

Таблица 2.6 – Описание таблицы REVIEWS

Название столбца	Тип данных	Описание
id	integer	идентификатор отзыва, первичный ключ
p_subject	integer	идентификатор соискателя, оставившего отзыв, внешний ключ
p_object	integer	идентификатор соискателя, о котором был оставлен отзыв, внешний ключ
b_subject	integer	идентификатор работодателя, оставившего отзыв, внешний ключ
b_object	integer	идентификатор работодателя, о котором был оставлен отзыв, внешний ключ
text	varchar(100)	текст отзыва
rating	integer	оценка
reported	char(1)	показатель того, была ли оставлена жалоба на отзыв

Таблица PROMOTION_REQUESTS хранит информацию о запросах на подтверждение компании. Описание её столбцов представлено в таблице 2.7.

Таблица 2.7 – Описание таблицы PROMOTION_REQUESTS

Название столбца	Тип данных	Описание
id	integer	идентификатор запроса, первичный ключ
company_id	int	идентификатор компании, внешний ключ
proof	varchar(125000)	текст предоставленного доказательства

Таблица ACCOUNT_DROP_REQUESTS хранит информацию о запросах на удаление учётных записей. Описание её столбцов представлено в таблице 2.8. В данной таблице b_subject ссылается на столбец id таблицы BOURGEOISIE, а p_subject – на столбец id таблицы PROLETARIAT. Данное решение было принято для исключения вспомогательного столбца is_company.

Таблица 2.8 – Описание таблицы ACCOUNT_DROP_REQUESTS

Название столбца	Тип данных	Описание
id	integer	идентификатор запроса, первичный ключ
p_subject	integer	идентификатор соискателя, внешний ключ
b_subject	integer	идентификатор компании, внешний ключ
commentary	varchar(255)	комментарий к запросу

Таблица TOKENS хранит информацию о refresh-токенах пользователей. Описание её столбцов представлено в таблице 2.9.

Таблица 2.9 – Описание таблицы TOKENS

Название столбца	Тип данных	Описание
id	integer	идентификатор токена, первичный ключ
owner_p	integer	идентификатор соискателя, внешний ключ
owner_b	integer	идентификатор компании, внешний ключ
value	varchar(256)	значение токена

Описание столбцов таблицы BLACK_LIST представлено в таблице 2.10.

Таблица 2.10 – Описание таблицы BLACK_LIST

Название столбца	Тип данных	Описание
id	integer	идентификатор записи, первичный ключ
p_subject	integer	идентификатор соискателя, которому запрещается оставлять отзыв, внешний ключ
p_object	integer	идентификатор соискателя, о котором запрещено оставлять отзыв, внешний ключ
b_subject	integer	идентификатор работодателя, которому запрещается оставлять отзыв, внешний ключ
b_object	integer	идентификатор работодателя, о котором запрещено оставлять отзыв, внешний ключ

Данная таблица хранит информацию о пользователях, которым запрещено оставлять отзывы о некоторых других пользователях.

2.4 Проектирование сервера web-приложения

Для обработки запросов применяется четыре роутера, каждый из которых обрабатывает запросы к определённым адресам. Так, masterRouter обрабатывает все запросы, для которых не нужна авторизация, такие как получение списка вакансий, жалоба на отзыв и так далее. Список представляемых обработчиков представлен в таблице 2.11.

Таблица 2.11 – Список представляемых обработчиков роутера masterRouter

Адрес	Метод	Описание
/public-vacancies	GET	Возвращает список публичных вакансий с заданным смещением относительно начала таблицы. Если в строке запроса находятся параметры из списка фильтров, их значения применяются для фильтрации вакансий
/public-companies	GET	Возвращает список названий и идентификаторов всех компаний. Если не получен параметр поисковой строки skipRating, также вычисляется рейтинг каждой компании
/company-reviews	GET	Возвращает список отзывов о компании, её рейтинг и вспомогательные данные об отзывах о компании
/report-review	PUT	Создаёт жалобу на отзыв. Если жалоба уже существовала, возвращает сообщение об этом. Если жалобы не существовало, возвращает сообщение об успешном создании жалобы
/logout	GET	Производит очистку cookie клиента от идентификатора пользователя, его типа пользователя, access-токена и refresh-токена

Роутер proletariatRouter предназначен для обработки запросов соискателей. Все запросы к нему должны начинаться с /prol. Список представляемых обработчиков представлен в таблице 2.12.

Таблица 2.12 – Список представляемых обработчиков роутера proletariatRouter

Адрес	Требуется авторизации	Метод	Описание
/login	Нет	GET	Производит авторизацию, генерирует пару токенов и устанавливает необходимые данные в cookie пользователя

Продолжение таблицы 2.12

/register	Нет	PUT	Осуществляет проверку на существование пользователя с предоставленным логином и создаёт нового пользователя
/personal	Да	GET	Возвращает идентификатор пользователя, его имя, объект, содержащий сведения о его образовании, объект, содержащий сведения о его опыте работы, адрес его электронной почты и показатель того, был выполнен запрос на удаление учётной записи или нет
/personal	Да	POST	Обновляет персональные данные пользователя. Притом, если исходное значение электронной почты было правильным, а новое не проходит проверку регулярным выражением, то возвращается сообщение об ошибке
/password	Да	PATCH	Устанавливает хеш нового пароля в базу данных и возвращает сообщение об успешной смене пароля
/review	Да	PUT	Создаёт отзыв о компании по её идентификатору
/review	Да	DELETE	Проверяет, существует ли отзыв с данным идентификатором от данного пользователя. Если такой отзыв существует, удаляет его. В противном случае возвращает сообщение об ошибке
/cv	Да	GET	Если строка запроса не содержит параметров, возвращает список всех резюме соискателя. В случае, если установлен параметр id, возвращает резюме по его идентификатору. В противном случае возвращает сообщение о том, что резюме не найдено. В случае, если установлен параметр vacancy, но не установлен id, возвращает список резюме, которыми можно откликнуться на данную вакансию

Окончание таблицы 2.12

/cv	Да	POST	Обновляет данные резюме по его идентификатору и возвращает сообщение об успешном выполнении операции
/cv	Да	PUT	Создаёт новое резюме с заданными данными и возвращает сообщение об успешном создании резюме
/cv	Да	DELETE	Получает идентификатор пользователя из cookie-файла. Проверяет, существует ли резюме с указанным в теле запроса названием и принадлежащее данному соискателю. Если резюме с указанным названием существует, то удаляет его и возвращает сообщение об успешном удалении
/responses	Да	GET	Возвращает список всех откликов данного соискателя и общее число откликов данного соискателя
/responses	Да	PUT	Создаёт отклик по идентификатору резюме и вакансии
/responses	Да	DELETE	Удаляет отклик по его идентификатору
/drop-requests	Да	PUT	Проверяет, был ли запрос на удаление учётной записи уже отправлен текущим соискателем. Если такой запрос уже существует, возвращает сообщение об ошибке. В противном случае создаёт запрос и возвращает сообщение об его успешном создании

Роутер bourgeoisieRouter предназначен для обработки запросов работодателей и запросов на получение публичной информации о работодателе. Список представляемых обработчиков представлен в таблице 2.13.

Таблица 2.13 список представляемых обработчиков роутера bourgeoisieRouter

Адрес	Требуется авторизация	Метод	Описание
/login	Нет	GET	Если пользователь уже авторизован, возвращает сообщение об ошибке. Производит авторизацию, генерирует пару токенов и устанавливает нужные данные в cookie пользователя. В случае ошибки производит выход пользователя из учётной записи

Продолжение таблицы 2.13

/register	Нет	PUT	Осуществляет проверку на существование пользователя с предоставленным логином. В случае, если логин занят, возвращает сообщение об ошибке
/personal	Да	GET	Возвращает идентификатор работодателя, его имя, описание, адрес его электронной почты показатель того, был выполнен запрос на удаление учётной записи, показатель того, была ли компания подтверждена и показатель того, был ли отправлен
/personal	Да	POST	Обновляет персональные данные пользователя
/password	Да	PATCH	Устанавливает хеш нового пароля в базу данных и возвращает сообщение об успешной смене пароля
/review	Да	GET	Предназначен для получения отзывов о соискателе по его идентификатору
/review	Да	PUT	Проверяет, имеет ли право соискатель оставлять отзыв о данной компании, проверяя наличие нужной записи в чёрном списке. Создает отзыв и возвращает сообщение об успешном его создании
/review	Да	DELETE	Проверяет, существует ли отзыв с данным идентификатором от данного пользователя. Если такой отзыв существует, удаляет его. В противном случае возвращает сообщение об ошибке
/applicants-list	Да	GET	Возвращает список идентификаторов и имён соискателей, которые отправили отклик на вакансию текущего работодателя и чьи отклики были одобрены, для отображения в списке доступных для создания отзыва соискателей
/icon	Да	PUT	Записывает содержимое тела запроса в изображение текущего работодателя на стороне сервера. Возвращает сообщение об успешной замене изображения

Окончание таблицы 2.13

/info	Нет	GET	Возвращает публичные данные работодателя: идентификатор, название, описание и адрес электронной почты работодателя по его идентификатору
/responses	Да	GET	Возвращает список откликов текущего работодателя и количество всех откликов
/responses	Да	POST	Обновляет статус отклика по идентификатору и возвращает сообщение об успешном обновлении данных
/vacancy	Да	GET	Возвращает список всех вакансий текущего работодателя по его идентификатору или вакансию работодателя по её идентификатору
/vacancy	Да	PUT	Проверяет, занято ли данное имя вакансии другой вакансией данного работодателя, производит валидацию полей вакансии, создаёт её и возвращает сообщение об успешном создании
/vacancy	Да	POST	Обновляет данные о вакансии
/vacancy	Да	DELETE	Удаляет вакансию по её идентификатору, генерирует событие changed
/promotion-request	Да	PUT	Проверяет существование запроса на подтверждение текущей компании. Создает запрос на подтверждение компании с идентификатором текущего работодателя
/drop-request	Да	PUT	Проверяет существование запроса на удаление учётной записи текущей компании. Создает запрос на удаление учётной записи с идентификатором текущего работодателя и комментарием, полученным из тела запроса

Роутер adminRouter предназначен для обработки запросов администратора. Для доступа ко всем обработчикам данного роутера требуется авторизация и роль администратора. Список представляемых обработчиков представлен в таблице 2.14.

Таблица 2.14 Список представляемых обработчиков роутера adminRouter

Адрес	Метод	Описание
/promotion-requests	GET	Возвращает список запросов на подтверждение компаний

Окончание таблицы 2.14

/drop-requests	GET	Возвращает список запросов на удаление учётных записей, и общее количество запросов
/promote	PATCH	Устанавливает статус компании в подтверждённое значение и удаляет запрос на подтверждение. Если у компании установлен адрес электронной почты, отправляет на него письмо с сообщением о том, что запрос на подтверждение был удовлетворён
/promote	DELETE	Удаляет запрос о повышении по его идентификатору. Если у компании установлен адрес электронной почты, отправляет на него письмо с сообщением о том, что запрос на подтверждение был отклонён
/drop-user	DELETE	Удаляет учётную запись по её идентификатору. Возвращает сообщение об успешном удалении пользователя
/drop-user	POST	Удаляет запрос на удаление учётной записи по его идентификатору. Возвращает сообщение об успешном отклонении запроса
/reported-reviews	GET	Возвращает список отзывов, на которые поступила жалоба и общее число отзывов, на которые была оставлена жалоба
/review	POST	Обновляет значение столбца reported строки таблицы REVIEWS по идентификатору и возвращает сообщение об успешном удалении жалобы
/review	DELETE	Удаляет отзыв по его идентификатору и возвращает сообщение об успешном удалении жалобы
/ban	GET	Возвращает список записей в чёрном списке
/ban	POST	Создаёт запись в чёрном списке по идентификаторам блокируемого пользователя и пользователя, у которого пользователь блокируется
/ban	DELETE	Удаляет запись из чёрного списка по идентификаторам пользователей

Для проверки авторизации используется middleware, которое проверяет, находится ли запрашиваемый ресурс в списке защищаемых, и в случае, если для доступа к данному ресурсу необходима роль соискателя, работодателя или администратора, производит авторизацию и аутентификацию.

Все запросы, для которых не был зарегистрирован обработчик, сначала обрабатываются middleware, обрабатывающим запросы на статические файлы. Если и это middleware не вернуло ответ, возвращается страница React-приложения.

3 Разработка web-приложения

3.1 Разработка бэкэнда

Для разработки бэкэнда был использован фреймворк h3. Согласно [2], в нём для обработки запросов могут применяться простые обработчики запросов. Обработчики запросов, добавляющие к ответу заголовки CORS и возвращающие html-страницу на любой запрос, представлены в листинге 3.1.

```
app.use(defineEventHandler(event => {
  appendHeaders(event, {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Expose-Headers': '*'
  })
}));
app.use(defineEventHandler(async event => {
  setResponseHeader(event, 'Content-Type', 'text/html');
  return fs.readFileSync('./views/react-front/dist/index.html');
}));
```

Листинг 3.1 – Простые обработчики запросов

Также для обработки запросов могут использоваться более продвинутые роутеры. Фрагмент роутера masterRouter представлен в листинге 3.2.

```
const masterRouter = createRouter()
.get('/public-companies', defineEventHandler(async event => {
  const query = getQuery(event);
  const companies = await BOURGEOISIE.findAndCountAll({
attributes: ['id', 'name'] })
  companies.rows = companies.rows.map(e => e.dataValues);
  if (!query.skipRating) {
    for (let company of companies.rows) {
      company.rating = await GetRating('C',
company.id);
    }
  }
  return { companies: companies.rows, totalElements:
companies.count };
}))
.get('/logout', defineEventHandler(async event => {
  await logout(event);
  return { message: 'what kind of message did you expect?'
};
}));
```

Листинг 3.2 – Фрагмент роутера masterRouter

Объекты запроса и ответа объединены в объект event, над которым можно производить различные операции, такие как получение и установка cookie, чтение тела запроса, установка тела ответа при помощи оператора return и так далее.

Для запуска сервера необходимо создать объект приложения, определить

обработчики запросов и роутеры, преобразовать объект приложения к слушателю событий Node.js и запустить сервер при помощи метода `listen`. Код запуска сервера представлен в листинге 3.3. Поддержка HTTPS реализована согласно [3].

```
import { createServer } from "node:https";
import { createApp, toNodeListener } from "h3";
import { masterRouter } from '../routers/master.mjs';
export const app = createApp();
app.use(masterRouter);
const nodeApp = toNodeListener(app);
const credentials = {
  key: fs.readFileSync('../server.key'),
  cert: fs.readFileSync('../server.crt')
}
const httpsServer = createServer(credentials, nodeApp);
httpsServer.listen(process.env.PORT || 3000);
```

Листинг 3.3 – Код запуска сервера

Для обработки запросов на WebSocket-соединение используется специальный адаптер для Node.js от Crossws. Функция обработки WebSocker-соединений представлена в листинге 3.4.

```
import wsAdapter from "crossws/adapters/node";
const { handleUpgrade } = wsAdapter({
  hooks: {
    async open(peer) {
      console.log("[ws] open", peer);
      vacancyEmitter.on('changed', vacancy => {
peer.send(vacancy); console.log('sending') });
    },

    message(peer, message) {
      console.log("[ws] message", peer, message);
      if (message.text().includes("ping")) {
        peer.send("pong");
      }
    },
    close(peer, event) {console.log("[ws] close", peer, event);
  },
  error(peer, error) {console.log("[ws] error", peer, error);
  },
},
});
```

Листинг 3.4 – Функция обработки WebSocker-соединений

Файлы, требующиеся для работы фронтэнда, такие как таблицы стилей и скрипты, запрашиваются браузером и должны обрабатываться отдельно. Для этого предусмотрена функция `handleStatic`, которая проверяет запрашиваемый ресурс на соответствие шаблону и в случае, если путь к ресурсу оканчивается на `.css`, `.js` или `.jpg`, ищет нужный файл в специальной папке на сервере. Код данной функции

представлен в листинге 3.5.

```
export function handleStatic(event) {
  const regex = /(css|js|jpg)$/;
  if (!regex.test(event.path)) return;
  return serveStatic(event, {
    getContents: async id => {
      return await readFile(join(publicDir, id))
    },
    getMeta: async id => {
      const stats = await stat(join(publicDir, id)).catch(() => {});
      if (!stats || !stats.isFile()) {
        return;
      }
      const extension = id.split('.').pop();
      return {
        type: extension === 'js' ? 'text/javascript'
        : extension === 'css' ? 'text/css'
        : extension === 'html' ? 'text/html'
        : 'image/jpeg',
        size: stats.size,
        mtime: stats.mtimeMs
      };
    }
  });
}
```

Листинг 3.5 – Функция обработки запросов на статические файлы

Для получения данных из базы данных использовалась ORM Sequelize. Применялся подход model-first, при котором в первую очередь разрабатывались модели данных, а затем в соответствии с ними создавались таблицы и обработчики запросов. Были определены модели для соискателей, резюме, работодателей, вакансий, откликов, отзывов, запросов на подтверждение, запросов на удаление учётных записей, токенов и записей чёрного списка. Объявление модели запросов на подтверждение представлено в листинге 3.6.

```
class PROMOTION_REQUESTS extends Model {}
PROMOTION_REQUESTS.init({
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  company_id: { type: DataTypes.INTEGER, allowNull: false,
  references: { model: 'BOURGEOISIE', key: 'id' } },
  proof: { type: DataTypes.STRING(125000) },
  sequelize,
  timestamps: false,
  modelName: 'PROMOTION_REQUESTS',
  tableName: 'PROMOTION_REQUESTS'
});
```

Листинг 3.6 – Функция обработки запросов на статические файлы

Для поиска, создания, обновления и удаления записей в базе данных применялись методы моделей findOne (findByPk, findAndCountAll, findAll), create, update и destroy соответственно.

Для отправки почты использовался пакет nodemailer и сервис mail.ru. Код

функции для отправки почты представлен в листинге 3.7.

```
export function sendMail(to, subject, text) {
    let options = {
        service: 'mail.ru',
        auth: {
            user: config.mailUsername,
            pass: config.mailPassword
        }
    }
    nodemailer.createTransport(smtpTransport(options)).sendMail({
        from: config.mailUsername, to, subject, text
    }, function(error){
        if (error) {
            log('email sending error: ' + JSON.stringify(error));
        }
    });
}
```

Листинг 3.7 – Функция отправки электронных писем

Данные для авторизации в почтовом сервисе, а также секретные строки для access-токена и refresh-токена и соль для хеширования паролей хранятся в конфигурационном файле в формате JSON.

3.2 Разработка фронтэнда

Для разработки сайта использовалась библиотека react в качестве базы, пакет react-router-dom для создания нескольких страниц с разными URI и библиотека mui-material с готовыми компонентами для упрощения разработки [4]. Фрагмент роутера фронтэнд-приложения представлен в листинге 3.8.

```
const router = createBrowserRouter([
  {
    path: '/',
    element: <Root />,
    errorElement: <ErrorPage/>,
    children: [
      { index: true, element: <Index />, loader: indexLoader,
      }, { path: 'cv',
        element: <CVs/>,
        loader: cvsLoader,
      },
    ],
  },
],
```

Листинг 3.8 – Фрагмент роутера фронтэнд-приложения

За каждую страницу отвечает свой компонент, который запрашивает данные с сервера и отображает их на странице. Для запроса данных с сервера была разработана специальная функция-декоратор, которая запрашивает данные с определённого адреса и в зависимости от возвращённого статуса ответа либо возвращает данные, либо перенаправляет пользователя на главную страницу или на

страницу выхода из учётной записи в случае ошибки. Код данной функции представлен в листинге 3.9.

```
export async function fetchForLoader(path) {
  return fetch(path).then(r => {
    if (r.ok) return r.json();
    else throw r.json();
  })
  .catch(async err => {
    err = await err;
    console.log(err);
    if (err.code === 401) {
      location.href = '/signout';
    } else if (err.code === 403) {
      location.href = '/';
    }
  })
  .then(d => {
    return d;
  });
}
```

Листинг 3.9 – Функция запроса данных с сервера для загрузчика

Для отправки данных на сервер была разработана другая функция-декоратор `fetchWithResult`. Её код представлен в листинге 3.10.

```
export async function fetchWithResult(path, options, showAlert,
onSuccess, onError) {
  fetch(path, options)
  .then(r => {
    if (r.ok) return r.json();
    else throw r.json();
  })
  .then(d => {
    showAlert(d.message, 'success');
    if (onSuccess) onSuccess(d);
  })
  .catch(async err => {
    err = await err;
    console.log(err);
    showAlert(err.message, 'error');
    if (onError) onError(err);
  });
}
```

Листинг 3.10 – Функция для отправки данных на сервер

Данная функция дополнительно принимает параметры `showAlert` (функция отображения на странице всплывающего сообщения) и функции `onSuccess` и `onError`, вызываемые при успешном и неудачном получении данных с сервера соответственно. Сообщение, показываемое при помощи `showAlert`, отображает сообщение, которое вернул сервер. Также оно цветом отображает, был ли запрос

выполнен успешно, или произошла ошибка.

Также для отображения номеров страниц в случаях, когда содержимое не помещается на одну страницу, применяется функция CustomPaging. Она использует компонент Paging из библиотеки mui-material. Её код представлен в листинге 3.11.

```
export function CustomPaging(
  query,
  totalElements,
  callback) {
  return (<Box sx={{
    display: 'flex',
    justifyContent: 'center',
    width: '100%'}}>
    <Paging count={Math.ceil(totalElements / 20 || 1)}
      page={Math.floor((query.offset ?? 0) / 20) + 1}
      onChange={callback}/>
  </Box>);
}
```

Листинг 3.11 – Функция для отправки данных на сервер

В качестве параметров данная функция принимает параметры строки запроса текущей страницы, общее количество элементов и функцию обратного вызова, которая вызывается при изменении состояния блока Paging. Так, при выборе любого номера страницы будет произведён переход на данную страницу, но с параметром offset, равным произведению разности номера страницы и единицы и двадцати. Сервер, получив запрос с параметром URI offset, запросит из базы данных строки с заданным смещением.

4 Тестирование web-приложения

Для тестирования web-приложения использовалось ручное тестирование. Обработчики запросов были проверены на возможность неавторизованного и неаутентифицированного доступа. Пример страницы, показывающей сообщение об ошибке о несуществующей странице, представлен на рисунке 4.1.

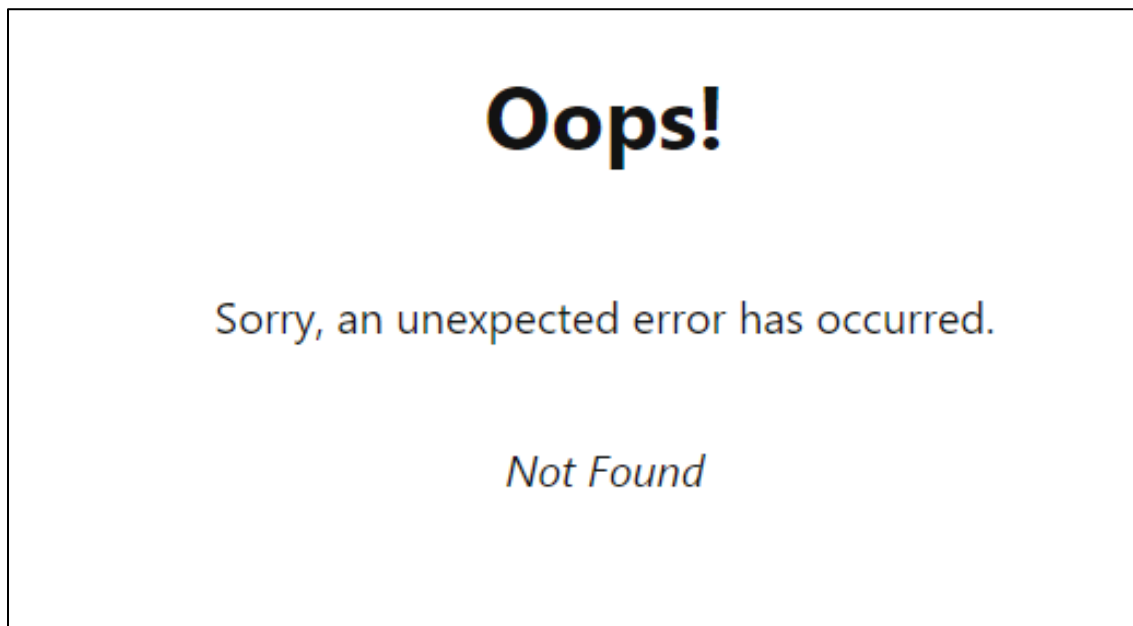


Рисунок 4.1 – Страница сообщения об ошибке не найденной страницы

Также обработчики запросов были проверены на правильность возвращаемых данных. На рисунке 4.2 представлена страница об ошибке, возникающей при получении от сервера неправильных данных.

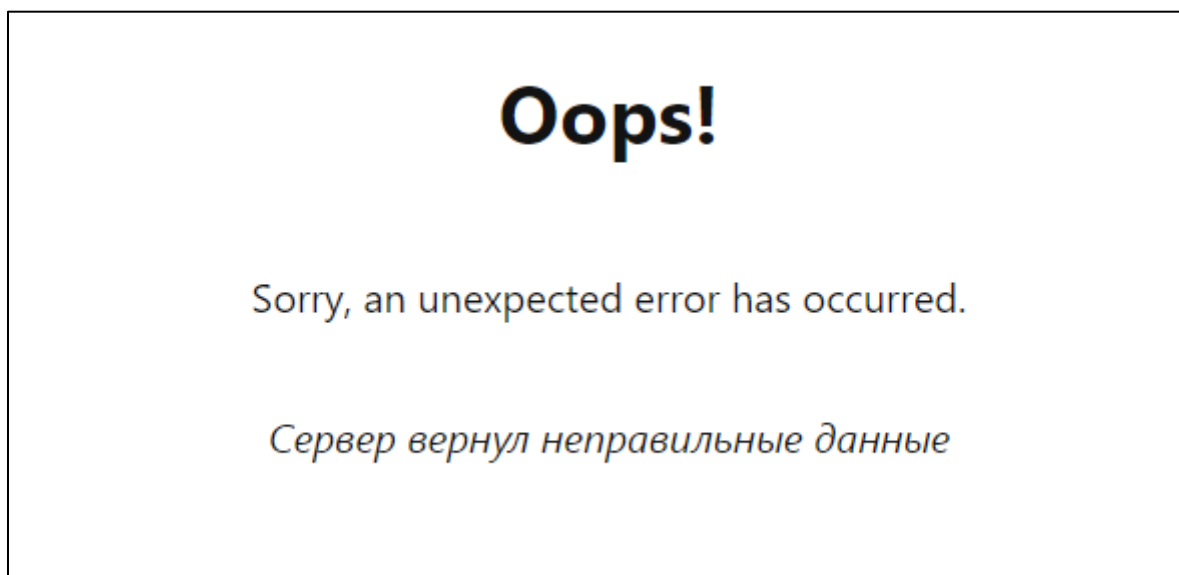


Рисунок 4.2 – Страница сообщения об ошибке неверный данных

Также обработчики запросов были проверены при помощи ПО Postman.

Пример запроса представлен на рисунке 4.3.

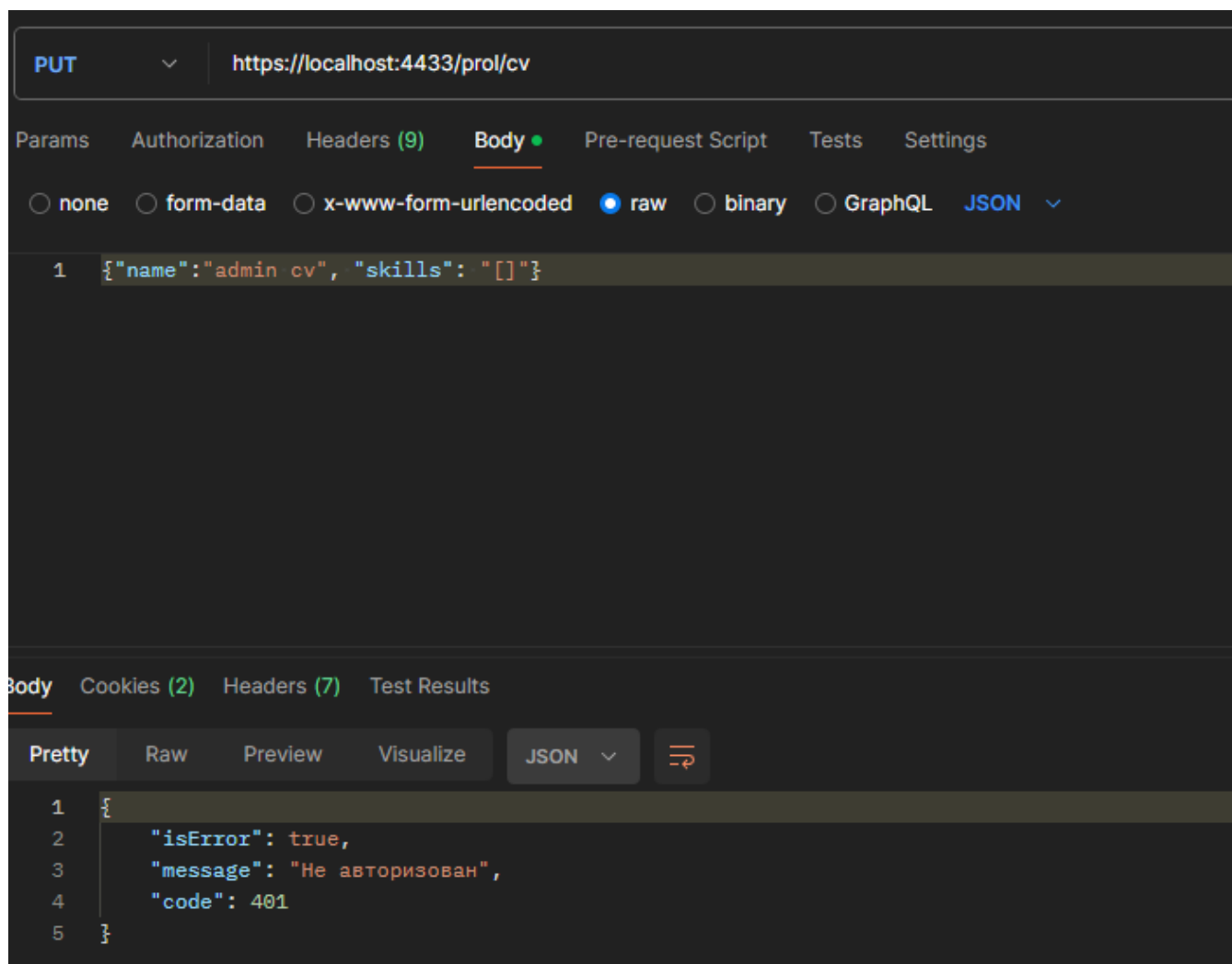


Рисунок 4.3 – Пример результата теста в Postman

В данном тесте был выполнен запрос к обработчику, который предусматривает проверку авторизации. На клиентской стороне были установлены cookie токенов, но не были установлены cookie идентификатора и типа пользователя. Обработчик обнаружил это и вернул сообщение об ошибке.

Схожие сообщения возвращаются в случае, если администратор пытается обратиться к обработчикам запросов, к которым не должен иметь доступа. К таким обработчикам относятся все обработчики роутера `bourgeoisieRouter` и обработчики `personal`, `review`, `cv`, `password`, `responses` и `drop-request` роутера `proletariatRouter`. В таких случаях пользователю возвращается ответ с кодом 403 и сообщением о том, что данный обработчик не предназначен для обработки запросов администратора.

Также было применено автоматическое тестирование. Для этого было создано вспомогательное приложение, выполняющее запросы ко всем обработчикам запросов с заданными параметрами при помощи функции `fetch`, вызываемую со всеми методами для каждого адреса. Были проверены все обработчики запросов всеми HTTP-методами с различными параметрами. Фрагмент приложения для тестирования представлен в листинге 4.1.

```

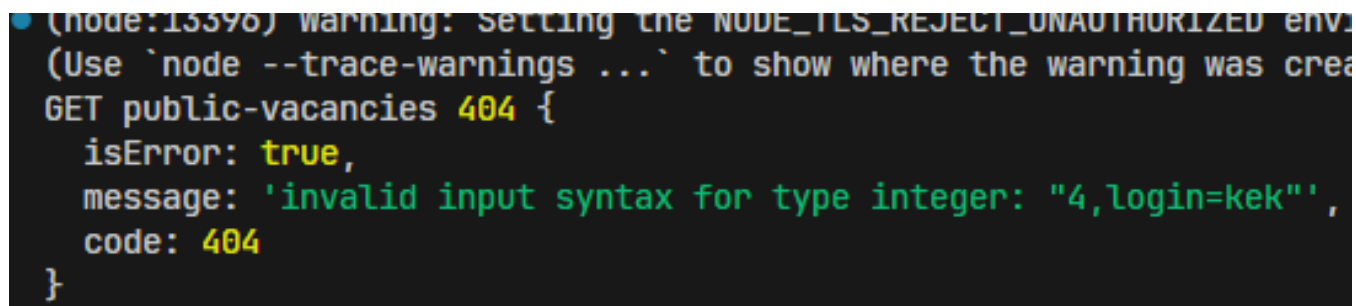
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";

(async () => {
  for (let method of ['GET', 'PUT', 'POST', 'DELETE']) {
    for (let uri of uris) {
      for (let param of params) {
        const response = await
fetch(`https://localhost:4433/${uri}?${params}`, { method });
        const contentType = response.headers.get('content-type');
        let responseBody;
        if (contentType === 'application/json') {
          responseBody = await response.json();
        } else {
          responseBody = await response.text();
        }
        console.log(method, uri, response.status, responseBody);
      }
    }
  }
}) ()

```

Листинг 4.1 – Фрагмент приложения для тестирования

Фрагмент вывода данного приложения представлен на рисунке 4.4.



```

(node:13396) warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED env.
(Use `node --trace-warnings ...` to show where the warning was crea
GET public-vacancies 404 {
  isError: true,
  message: 'invalid input syntax for type integer: "4,login=kek"',
  code: 404
}

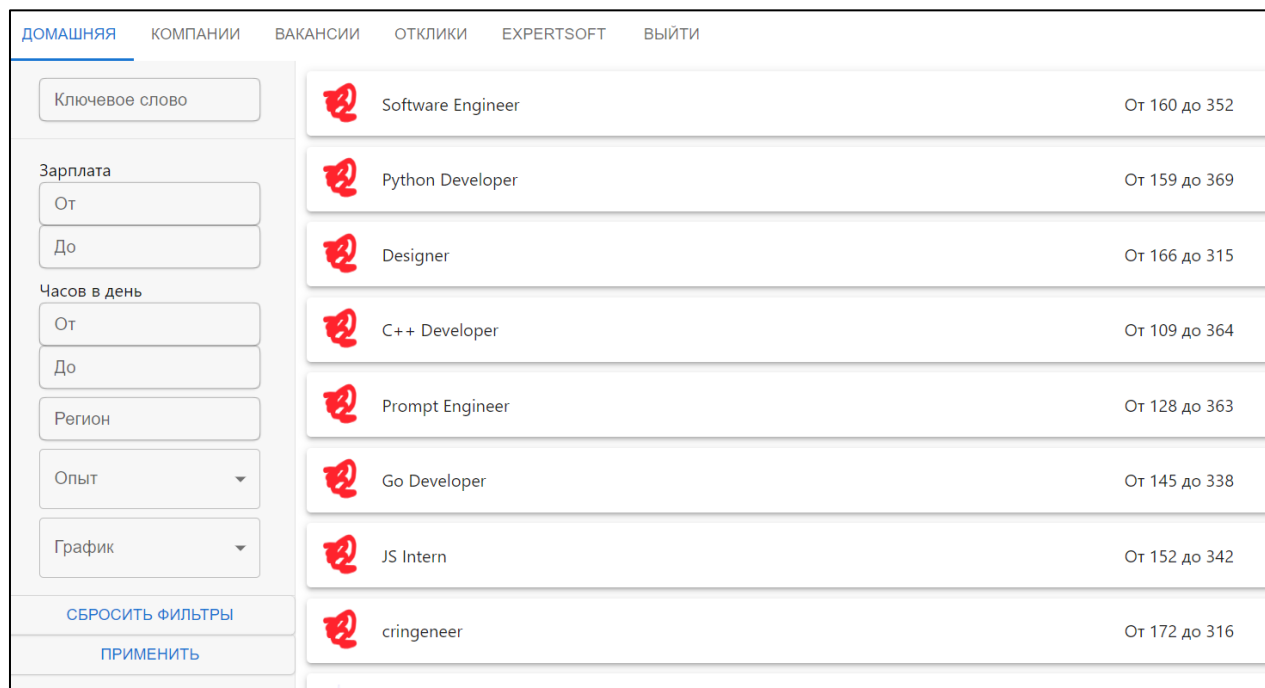
```

Рисунок 4.4 – Фрагмент вывода приложения для тестирования

В ходе тестирования были выявлены и исправлены несоответствия запрашиваемых клиентом и возвращаемых сервером данных, а также ошибки в исходном коде.

5 Руководство пользователя

При первом открытии сайта пользователь видит страницу вакансий, представленную на рисунке 5.1.

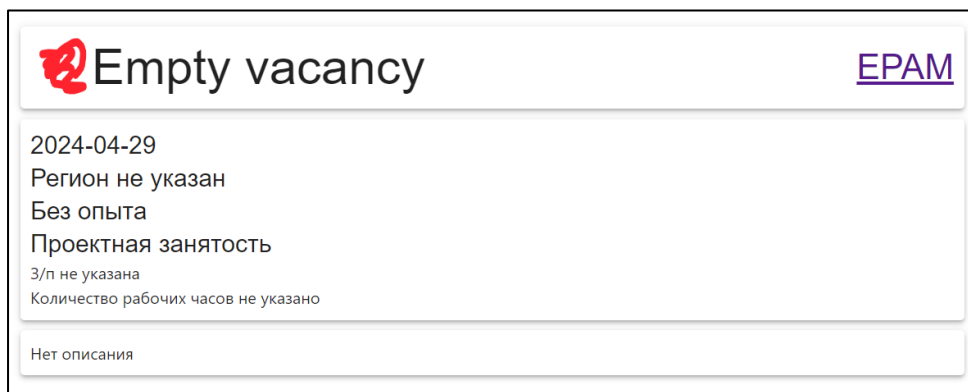



Ключевое слово	Вакансия	Зарплата
	Software Engineer	От 160 до 352
	Python Developer	От 159 до 369
	Designer	От 166 до 315
	C++ Developer	От 109 до 364
	Prompt Engineer	От 128 до 363
	Go Developer	От 145 до 338
	JS Intern	От 152 до 342
	cringeneer	От 172 до 316

Рисунок 5.1 – Страница публичных вакансий

Чтобы перейти на страницу нужно вакансии, пользователь должен нажать на её карточку. Чтобы отфильтровать вакансии по нужным признакам, пользователь должен ввести желаемые значения в нужные поля блока фильтрации в левой части страницы и нажать на кнопку «Применить». Для сброса фильтров пользователь может нажать на соответствующую кнопку.

На странице вакансии, представленной на рисунке 5.2, пользователь может ознакомиться со всеми данными вакансии и перейти на страницу описания работодателя.



 Empty vacancy [EPAM](#)

2024-04-29

Регион не указан

Без опыта

Проектная занятость

З/п не указана

Количество рабочих часов не указано

Нет описания

Рисунок 5.2 – Страница вакансии

На странице работодателя пользователь может ознакомиться с его описанием.

Для ознакомления с отзывами о работодателе пользователь должен перейти на страницу со списком всех компаний, представленную на рисунке 5.3.

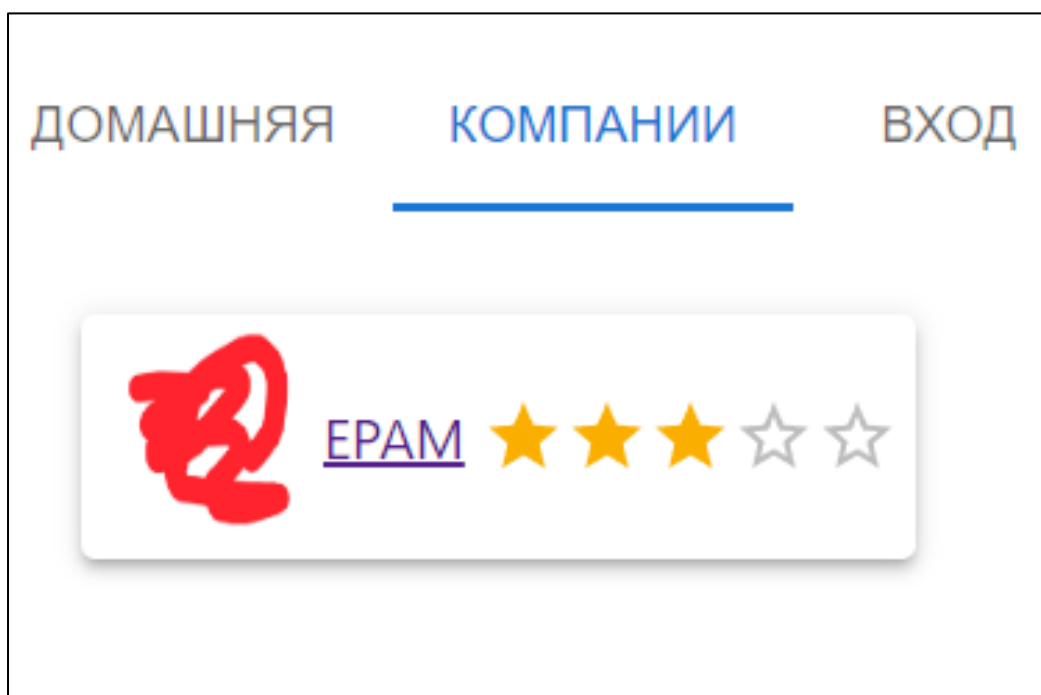


Рисунок 5.3 – Страница всех компаний

По ссылке в карточке пользователь может перейти на страницу с отзывами о компании, представленную на рисунке 5.4.

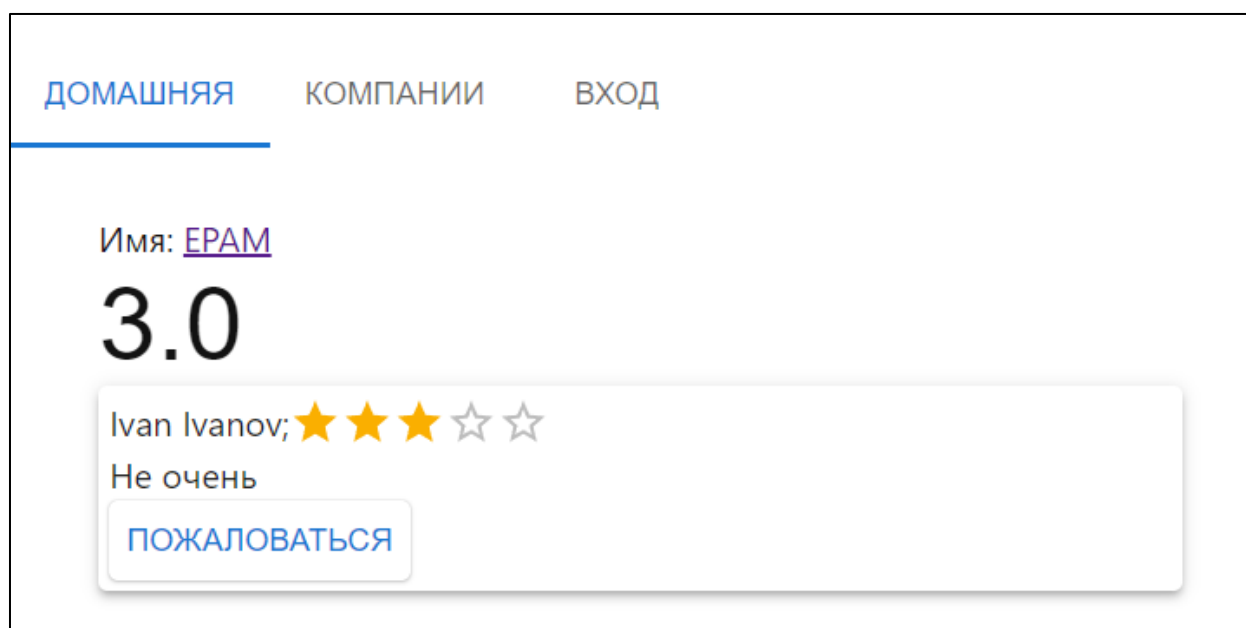


Рисунок 5.4 – Страница отзывов о компании

С данной страницы пользователь может попасть на страницу описания компании и пожаловаться на отзыв нажатием на соответствующую кнопку в карточке отзыва.

Для входа в учётную запись или регистрации пользователь должен перейти на

страницу «Вход», представленную на рисунке 5.5.

Рисунок 5.5 – Страница входа и регистрации

Пользователь должен нажатием на соответствующие блоки выбрать, что он хочет сделать: войти в учётную запись или зарегистрироваться – и в качестве кого: соискателя или работодателя – и нажать на кнопку «Подтвердить» для совершения действия или нажать на кнопку «Отмена» для перехода на главную страницу.

Работодателю доступны дополнительные страницы. Например, страница вакансий работодателя, представленная на рисунке 5.6.

Рисунок 5.6 – Страница вакансий компании

На данной странице работодатель может просмотреть список своих вакансий, создать новую нажатием на соответствующую кнопку и перейти к изменению нужной вакансии.

Также работодателю доступна страница откликов на его вакансии,

представленная на рисунке 5.7, на которой работодатель может просмотреть список откликов.

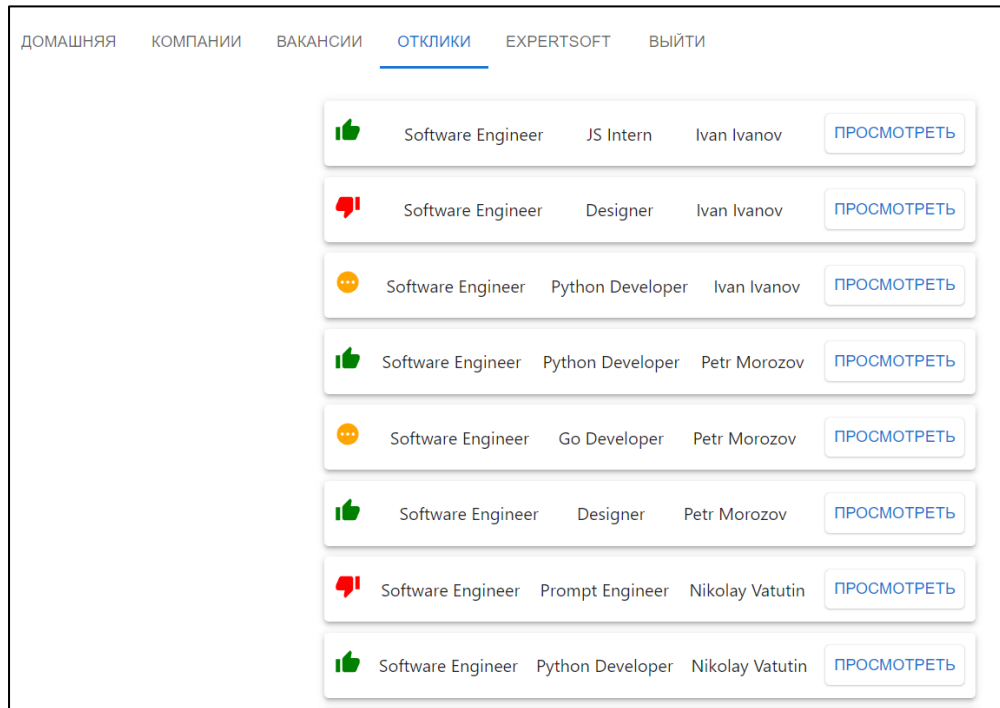


Рисунок 5.7 – Страница откликов компании

Пользователь может вызвать всплывающее окно с подробной информацией о каждом отклике по нажатию на кнопку «Просмотреть», пример которого представлен на рисунке 5.8.

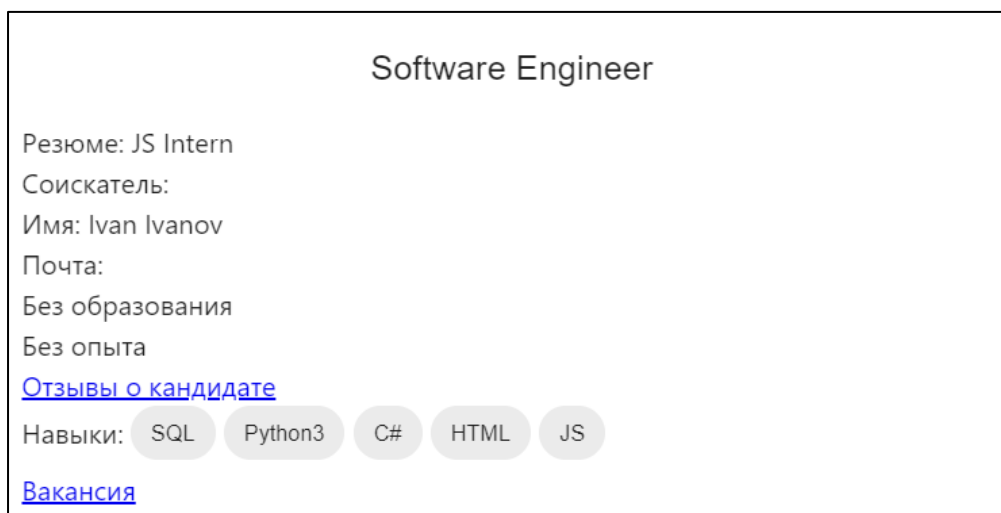


Рисунок 5.8 – Страница откликов компании

Из окна подробностей пользователь может перейти на страницу отзывов о кандидате и на страницу вакансии, на которую был отправлен отклик. Также, если отклик ещё ожидает рассмотрения, работодатель может принять или отклонить его нажатием на соответствующую кнопку.

Соискатель имеет доступ к другим дополнительным страницам. Одной из них

является резюме соискателя. На ней пользователь может ознакомиться со списком своих резюме, создать новое и перейти к редактированию нужного резюме по нажатию на ссылку нужного резюме. Пример данной страницы представлен на рисунке 5.9.

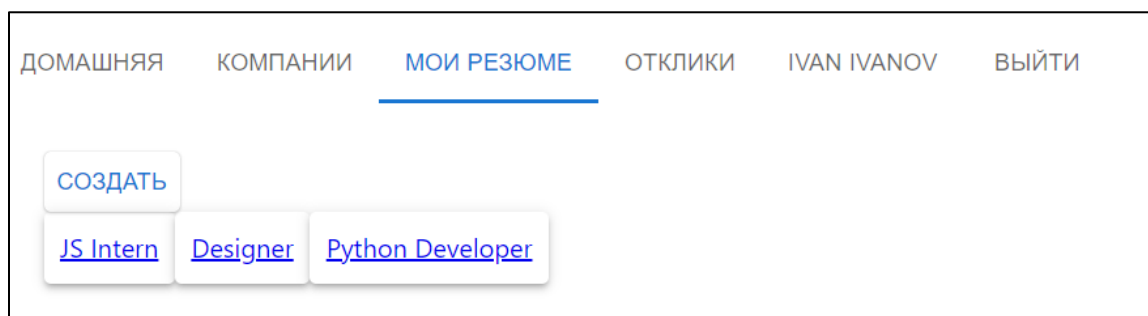


Рисунок 5.9 – Страница резюме соискателя

Страница редактирования резюме содержит поле ввода названия, область отображения ключевых навыков, поле ввода названия нового навыка, кнопку добавления навыка в список и кнопки сохранения и удаления резюме. В случае, если пользователь введёт название уже присутствующего в резюме навыка, появится уведомляющее об этом сообщение.

Пользователь также может просматривать список своих откликов на странице Отклики. В карточке каждого отклика находится кнопка «Отозвать», нажатие на которую отправляет запрос на удаление отклика. Пример данной страницы представлен на рисунке 5.10.

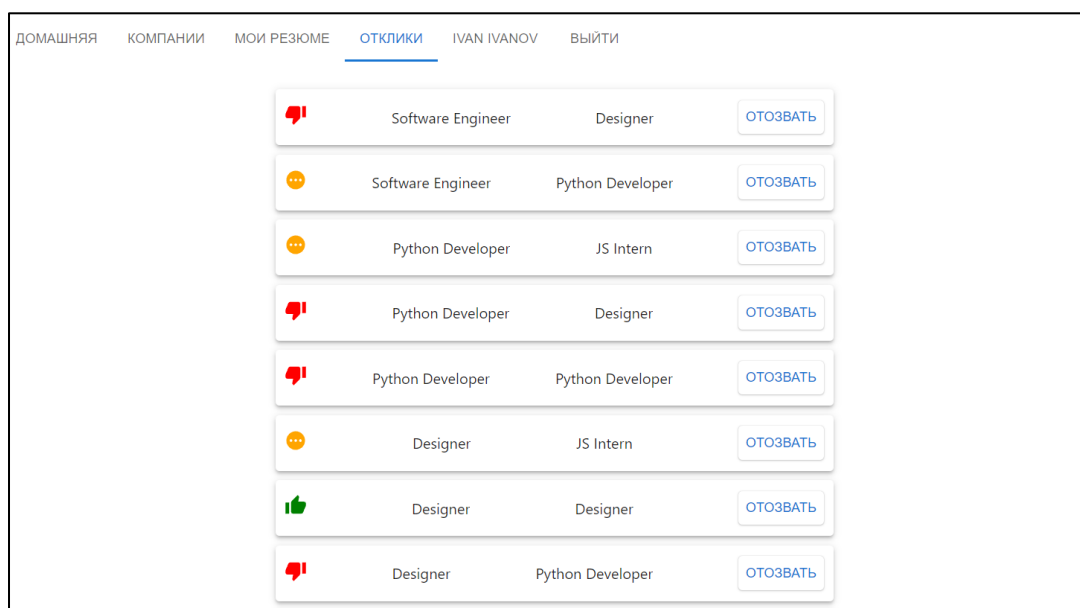


Рисунок 5.10 – Страница откликов соискателя

Соискателям и работодателям доступна страница создания отзывов друг о друге. На данной странице расположен выпадающий список доступных кандидатов для отзыва, поле ввода комментария, поле ввода оценки и кнопка отправки отзыва.

Другой доступной для всех авторизованных пользователей страницей является страница персональных данных. На ней пользователи могут изменить своё имя и адрес электронной почты, соискатели дополнительно могут изменить данные о своём образовании и опыте работы, а работодатели могут изменить описание и иконку и отправить запрос о подтверждении компании, по желанию приложив текстовый файл со своим комментарием. Также соискателям и работодателям доступна отправка запроса на удаление учётной записи.

Администратору доступен другой набор страниц. Он может просматривать список запросов на подтверждение компаний, получать комментарий по каждому из них и отклонять либо принимать их нажатиями на соответствующие кнопки. Точно так же администратор может взаимодействовать со списком запросов на удаление учётных записей. Пример страницы запросов на подтверждение компаний представлен на рисунке 5.11.

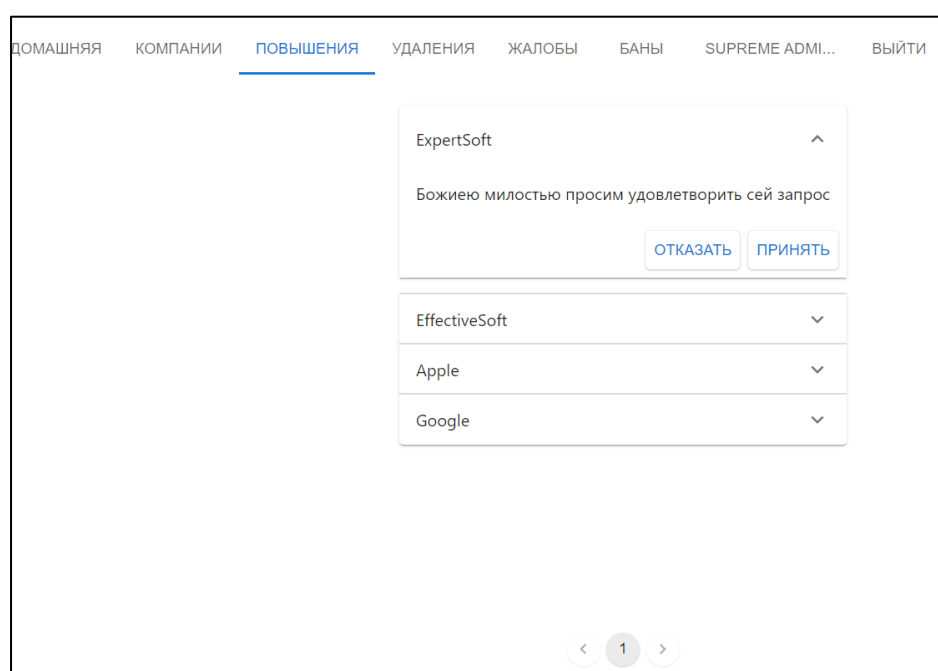


Рисунок 5.11 – Страница запросов на подтверждение компаний

Ещё одной доступной только администратору страницей является страница жалоб на отзывы. На ней расположены карточки жалоб, содержащие тип пользователя, оставившего отзыв, его имя, имя того, на кого был оставлен отзыв, и текст отзыва. Администратор может удалить комментарий, заблокировать возможность автору отзыва впредь оставлять отзывы о данном пользователе и удалить жалобу.

Последней доступной только администратору страницей является страница чёрного списка, на которой отображаются записи о заблокированных пользователях. Администратор может снять блокировку возможности нужному пользователю оставлять отзывы о другом указанном пользователе, нажав на кнопку «Разблокировать» в карточке нужной записи чёрного списка.

Заключение

При выполнении курсового проекта было создано приложение поиска и предложения работы. Сервер был создан при помощи платформы Node.js, языка программирования JavaScript и фреймворка h3. Web-сайт был реализован при помощи библиотеки React. База данных была реализована в СУБД PostgreSQL. Были реализованы все функциональные требования, а именно:

- обеспечение возможности регистрации и авторизации;
- поддержка ролей гостя, соискателя, работодателя и администратора;
- обеспечение возможности изменять образование, опыт работы и список ключевых навыков соискателя;
- предоставление возможности работодателю принимать и отклонять отклики;
- обеспечение возможности оставлять отзывы об исполнителе и работодателе;
- предоставление возможности отслеживать статус предложения о работе в реальном времени;
- обеспечение возможности соискателю откликаться на предложения о работе;
- предоставление возможности создавать и удалять учётные записи;
- обеспечение возможности фильтровать предложения по критериям (тип, необходимый опыт, оплата и так далее);
- предоставление возможности размещать и удалять вакансии и заказы на выполнение работ.

Также были реализованы WebSocket-сервер и поддержка протокола HTTPS. Приложение было протестировано на наличие ошибок с использованием ручного и автоматического тестирования. Для ручного тестирования использовался браузер и Postman. Для автоматического тестирования использовалось вспомогательное приложение, использующее функцию fetch.

По итогам тестирования были исправлены следующие ошибки в приложении: неверный формат отправляемых данных, отсутствие необходимых данных в ответе, ошибки в проверке авторизации.

Также было создано иллюстрированное руководство пользователя, в котором были описаны способы взаимодействия с приложением и наглядно продемонстрированы web-страницы приложения.

Список используемых источников

- 1 PostgreSQL Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/docs/>.
- 2 h3 – The Web Framework for Modern JavaScript Era [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://h3.unjs.io>.
- 3 Enabling HTTPS on express.js [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://stackoverflow.com/questions/11744975/enabling-https-on-express-js>.
- 4 Material UI components [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://mui.com/material-ui/all-components/>.

Приложение А

Листинг определения моделей для ORM Sequelize

```
import { Sequelize, Model, DataTypes } from "sequelize"
const sequelize = new Sequelize('xd', 'postgres', mysecretpassword',
{
    host: 'ugabuntu',
    dialect: 'postgres',
    pool: {
        max: 10,
        min: 0
    }
})

class PROLETARIAT extends Model {}
PROLETARIAT.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    login: {
        type: DataTypes.STRING(20),
        allowNull: false,
        unique: true
    },
    name: {
        type: DataTypes.STRING(70),
        allowNull: false,
    },
    password_hash: {
        type: DataTypes.STRING(60),
        allowNull: false,
    },
    is_admin: {
        type: DataTypes.CHAR(1),
        allowNull: false,
        get() {
            return
this.getDataValue('is_admin') === 'Y'
        },
        set(value) {
            this.setDataValue('is_admin',
value ? 'Y' : 'N')
        }
    },
    education_json: {
        allowNull: false,
        type: DataTypes.STRING(200),
        get() {
            return
JSON.parse(this.getDataValue('education_json'))
        },
    },
})
```

```

        set(value) {
            this.setDataValue('education_json',
JSON.stringify(value))
        },
        experience_json: {
            allowNull: false,
            type: DataTypes.STRING(500),
            get() {
                return
JSON.parse(this.getDataValue('experience_json'))
            },
            set(value) {
                this.setDataValue('experience_json',
JSON.stringify(value))
            },
        },
        email: {
            type: DataTypes.STRING(30)
        },
    }, {
        sequelize,
        timestamps: false,
        modelName: 'PROLETARIAT',
        tableName: 'PROLETARIAT',
    });

class CVS extends Model {}
CVS.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    name: {
        type: DataTypes.STRING(30),
        allowNull: false,
    },
    applicant: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'PROLETARIAT',
            key: 'id',
        },
    },
    skills_json: {
        allowNull: false,
        type: DataTypes.STRING(100),
        get() {
            return
JSON.parse(this.getDataValue('skills_json'))

```

```

    },
    set(value) {
        this.setDataValue('skills_json',
JSON.stringify(value))
    }
}
}, {
    sequelize,
    timestamps: false,
    modelName: 'CVS',
    tableName: 'CVS',
});

PROLETARIAT.hasMany(CVS, { foreignKey: 'applicant', sourceKey: 'id',
onDelete: 'cascade' });
CVS.belongsTo(PROLETARIAT, { foreignKey: 'applicant', targetKey:
'id', onDelete: 'cascade' });

class BOURGEOISIE extends Model {}
BOURGEOISIE.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    login: {
        type: DataTypes.STRING(20),
        allowNull: false,
        unique: true
    },
    name: {
        type: DataTypes.STRING(70),
        allowNull: false,
        unique: true,
    },
    password_hash: {
        type: DataTypes.STRING(60),
        allowNull: false
    },
    approved: {
        type: DataTypes.CHAR(1),
        allowNull: false,
        get() {
            return
this.getDataValue('approved') === 'Y'
        },
        set(value) {
            this.setDataValue('approved',
value ? 'Y' : 'N')
        }
    },
    description: {
        type: DataTypes.STRING(2000)
    },

```

```

        email: {
            type: DataTypes.STRING(30),
        }, {
            sequelize,
            timestamps: false,
            modelName: 'BOURGEOISIE',
            tableName: 'BOURGEOISIE',
        })

class VACANCIES extends Model {}
VACANCIES.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    name: {
        type: DataTypes.STRING(30),
        allowNull: false,
    },
    release_date: {
        type: DataTypes.DATEONLY,
        allowNull: false,
    },
    company: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'BOURGEOISIE',
            key: 'id',
        },
    },
    active: {
        type: DataTypes.CHAR(1),
        allowNull: false,
        get() {
            return
this.getDataValue('active') === 'Y';
        },
        set(value) {
            this.setDataValue('active', value
? 'Y' : 'N');
        },
    },
    min_salary: {
        type: DataTypes.INTEGER,
        validate: {
            min: 0,
        },
    },
    max_salary: {
        type: DataTypes.INTEGER,
        validate: {

```

```

        min: 0,
      },
    },
    region: {
      type: DataTypes.STRING(20),
    },
    schedule: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        min: 1,
        max: 5
      }
    },
    experience: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        min: 1,
        max: 4
      }
    },
    min_hours_per_day: {
      type: DataTypes.INTEGER,
      validate: {
        min: 1,
      }
    },
    max_hours_per_day: {
      type: DataTypes.INTEGER,
      validate: {
        min: 1,
      }
    },
    description: {
      type: DataTypes.STRING(1000),
      allowNull: false,
    },
  }, {
    sequelize,
    timestamps: false,
    modelName: 'VACANCIES',
    tableName: 'VACANCIES',
  });

BOURGEOISIE.hasMany(VACANCIES, { foreignKey: 'company', sourceKey:
'id', onDelete: 'cascade' });
VACANCIES.belongsTo(BOURGEOISIE, { foreignKey: 'company', targetKey:
'id', onDelete: 'cascade' });

class RESPONSES extends Model {}
RESPONSES.init({
  id: {
    type: DataTypes.INTEGER,

```

```

        primaryKey: true,
        autoIncrement: true
    },
    cv: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'CVS',
            key: 'id',
        }
    },
    vacancy: {
        type: DataTypes.INTEGER,
        allowNull: false,
        references: {
            model: 'VACANCIES',
            key: 'id',
        }
    },
    status: {
        type: DataTypes.CHAR(1),
        allowNull: false,
        validate: {
            isIn: [['W', 'X', 'Y']]//wait,
no, yes
        }
    }
}, {
    sequelize,
    timestamps: false,
    modelName: 'RESPONSES',
    tableName: 'RESPONSES'
});

VACANCIES.hasOne(RESPONSES, { foreignKey: 'vacancy', sourceKey:
'id', onDelete: 'cascade' });
CVS.hasOne(RESPONSES, { foreignKey: 'cv', sourceKey: 'id', onDelete:
'cascade' });
RESPONSES.belongsTo(CVS, { foreignKey: 'cv', sourceKey: 'id'});
RESPONSES.belongsTo(VACANCIES, { foreignKey: 'vacancy', sourceKey:
'id'});

class REVIEWS extends Model {}
REVIEWS.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    p_subject: {
        type: DataTypes.INTEGER,
        references: {
            model: 'PROLETARIAT',
            key: 'id',

```



```

    },
    b_subject: {
      type: DataTypes.INTEGER,
      references: {
        model: 'BOURGEOISIE',
        key: 'id',
      }
    },
    b_object: {
      type: DataTypes.INTEGER,
      references: {
        model: 'BOURGEOISIE',
        key: 'id',
      }
    },
    p_object: {
      type: DataTypes.INTEGER,
      references: {
        model: 'PROLETARIAT',
        key: 'id',
      }
    },
    text: {
      type: DataTypes.STRING(100),
    },
    rating: {
      type: DataTypes.INTEGER,
      allowNull: false,
      validate: {
        min: 1,
        max: 5
      }
    },
    reported: {
      type: DataTypes.CHAR(1),
      allowNull: false,
      defaultValue: 'N',
      validate: { isIn: [['Y', 'N']] }
    }
  }, {
    sequelize,
    timestamps: false,
    modelName: 'REVIEWS',
    tableName: 'REVIEWS'
  })

REVIEWS.belongsTo(PROLETARIAT, { foreignKey: 'p_object', onDelete: 'cascade' });
REVIEWS.belongsTo(PROLETARIAT, { foreignKey: 'p_subject', onDelete: 'cascade' });
REVIEWS.belongsTo(BOURGEOISIE, { foreignKey: 'b_object', onDelete: 'cascade' });
REVIEWS.belongsTo(BOURGEOISIE, { foreignKey: 'b_subject', onDelete:

```

```

'cascade' });

class PROMOTION_REQUESTS extends Model {}
PROMOTION_REQUESTS.init({
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  company_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: 'BOURGEOISIE',
      key: 'id'
    }
  },
  proof: {
    type: DataTypes.STRING(125000)
  }
}, {
  sequelize,
  timestamps: false,
  modelName: 'PROMOTION_REQUESTS',
  tableName: 'PROMOTION_REQUESTS'
});

BOURGEOISIE.hasOne(PROMOTION_REQUESTS, { foreignKey:
'company_id', sourceKey: 'id', onDelete: 'cascade' });
PROMOTION_REQUESTS.belongsTo(BOURGEOISIE, { foreignKey:
'company_id', sourceKey: 'id' });

class ACCOUNT_DROP_REQUESTS extends Model {}
ACCOUNT_DROP_REQUESTS.init({
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  p_subject: {
    type: DataTypes.INTEGER,
    references: {
      model: 'PROLETARIAT',
      key: 'id',
    }
  },
  b_subject: {
    type: DataTypes.INTEGER,
    references: {
      model: 'BOURGEOISIE',
      key: 'id',
    }
  }
},

```

```

        commentary: {
            type: DataTypes.STRING()
        }
    }, {
        sequelize,
        timestamps: false,
        modelName: 'ACCOUNT_DROP_REQUESTS',
        tableName: 'ACCOUNT_DROP_REQUESTS'
    });
PROLETARIAT.hasOne(ACCOUNT_DROP_REQUESTS, { foreignKey: 'p_subject',
sourceKey: 'id', onDelete: 'cascade' });
BOURGEOISIE.hasOne(ACCOUNT_DROP_REQUESTS, { foreignKey: 'b_subject',
sourceKey: 'id', onDelete: 'cascade' });

class TOKENS extends Model {}
TOKENS.init({
    type: {
        type: DataTypes.CHAR(1),
        allowNull: false,
        validate: {
            isIn: [['A', 'R']]
        }
    },
    owner_p: {
        type: DataTypes.INTEGER,
        references: {
            model: 'PROLETARIAT',
            key: 'id',
        }
    },
    owner_b: {
        type: DataTypes.INTEGER,
        references: {
            model: 'BOURGEOISIE',
            key: 'id',
        }
    },
    value: {
        type: DataTypes.STRING(256),
        allowNull: false
    }
}, {
    sequelize,
    timestamps: false,
    modelName: 'TOKENS',
    tableName: 'TOKENS'
});
PROLETARIAT.hasOne(TOKENS, { foreignKey: 'owner_p', sourceKey: 'id',
onDelete: 'cascade' });
BOURGEOISIE.hasOne(TOKENS, { foreignKey: 'owner_b', sourceKey: 'id',
onDelete: 'cascade' });

async function GetRating(userType, userId) {
    const result = (await

```

```

sequelize.query('select GetAverageRating(:userType, :userId);', {
  replacements: { userType, userId } })
    [0][0].getaveragerating;
    return result;
}

class BLACK_LIST extends Model {};
BLACK_LIST.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    p_subject: {
        type: DataTypes.INTEGER,
        references: {
            model: 'PROLETARIAT',
            key: 'id',
        }
    },
    p_object: {
        type: DataTypes.INTEGER,
        references: {
            model: 'PROLETARIAT',
            key: 'id',
        }
    },
    b_subject: {
        type: DataTypes.INTEGER,
        references: {
            model: 'BOURGEOISIE',
            key: 'id',
        }
    },
    b_object: {
        type: DataTypes.INTEGER,
        references: {
            model: 'BOURGEOISIE',
            key: 'id',
        }
    }
}, {
    timestamps: false,
    sequelize,
    modelName: 'BLACK_LIST',
    tableName: 'BLACK_LIST'
});
PROLETARIAT.hasMany(BLACK_LIST, { foreignKey: 'p_subject',
sourceKey: 'id', onDelete: 'cascade' });
BOURGEOISIE.hasMany(BLACK_LIST, { foreignKey: 'b_subject',
sourceKey: 'id', onDelete: 'cascade' });
PROLETARIAT.hasMany(BLACK_LIST, { foreignKey: 'p_object', sourceKey:
'id', onDelete: 'cascade' });
BOURGEOISIE.hasMany(BLACK_LIST, { foreignKey: 'b_object', sourceKey:

```

```
'id', onDelete: 'cascade' });

sequelize.sync({
    // alter: true,
    // force: true
});
export {
    PROLETARIAT,
    CVS,
    BOURGEOISIE,
    VACANCIES,
    RESPONSES,
    REVIEWS,
    PROMOTION_REQUESTS,
    ACCOUNT_DROP_REQUESTS,
    TOKENS,
    BLACK_LIST,
    GetRating,
    sequelize
}
```