

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

К защите допустить:
И.о. заведующего кафедрой
информатики
_____ С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
дипломного проекта
на тему

**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ПОИСКА РАБОТЫ МОЛОДЫМ
СПЕЦИАЛИСТАМ**

БГУИР ДП 1-40 04 01 079 ПЗ

Студент
Руководитель
Консультанты:
от кафедры
по технико-экономическому
обоснованию
Нормоконтролер
Рецензент

К.А. Пригожий
А.В. Давыдчик

А.В. Давыдчик

С.Ф. Куган
Н.Н. Бабенко

Минск 2025

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ПОИСКА РАБОТЫ МОЛОДЫМ СПЕЦИАЛИСТАМ: дипломный проект / К. А. Пригожий – Минск: БГУИР, 2025, – п.з. – 79 с., чертежей – 3 л. формата А4, плакатов – 3 л. формата А4.

Целью данного дипломного проекта является разработка программного средства в виде веб-приложения для поиска работы, специально ориентированного на молодых специалистов. Приложение призвано обеспечить эффективное и удобное взаимодействие между молодыми соискателями и работодателями, предлагая инструменты для поиска вакансий, размещения резюме и осуществления коммуникации. Важным аспектом является адаптация к потребностям различных категорий пользователей (соискателей и работодателей).

Объектом дипломного проекта является интерактивная платформа, предназначенная для содействия трудоустройству молодых специалистов путем предоставления системы поиска вакансий, инструментов для создания и управления резюме, а также средств для взаимодействия между соискателями и работодателями. Платформа разработана с учетом потребностей молодых специалистов, делающих первые шаги в карьере, и работодателей, заинтересованных в привлечении талантливых молодых кадров.

В ходе работы был проведен анализ существующих платформ для поиска работы, выявлены их ограничения и определены ключевые требования к функционалу разрабатываемой системы. Особое внимание уделено созданию интуитивно понятного пользовательского интерфейса, обеспечению масштабируемости для обработки растущего числа пользователей и вакансий, а также интеграции необходимых функций для эффективного взаимодействия между соискателями и работодателями.

Произведен выбор технологий и программных средств, оптимальных для реализации поставленных задач. Разработана архитектура приложения, спроектирована база данных и определены ключевые модули системы.

Разработанное в ходе дипломного проекта веб-приложение представляет собой законченный продукт, готовый к внедрению. Благодаря модульной структуре и продуманной архитектуре, система может быть легко расширена новыми функциями и адаптирована под различные страны и категории работников.

Дипломный проект является завершенным, поставленная задача решена в полной мере. Планируется дальнейшее развитие и совершенствование программно-аппаратного средства. Проект выполнен самостоятельно, проведен анализ оригинальности. В системе «Антиплагиат», процент оригинальности которой указан в приложении А составляет 90%.

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

УТВЕРЖДАЮ

И.о. _____ заведующего
кафедрой информатики
_____ С.И. Сиротко
« ____ » _____ 2025 г.

**ЗАДАНИЕ
на дипломный проект**

Обучающемуся Пригожему Кириллу Александровичу
(фамилия, собственное имя, отчество (если таковое имеется))

Курс 4 Учебная группа 153504

Специальность 1-40 04 01 «Информатика и технологии программирования»

Тема дипломного проекта

Программное средство для поиска работы молодым специалистам
(наименование темы)

Утверждена приказом ректора 13.03.2025 № 630-с

Исходные данные к дипломному проекту

Тип операционной системы – Windows 10; Язык программирования – Kotlin, JavaScript.

Перечень выполняемых функций: создание и управление резюме.

Авторизация пользователя.

Взаимодействие с вакансиями работодателей.

Перечень подлежащих разработке вопросов или краткое содержание
расчетно-пояснительной записки

1 Анализ литературных источников и программных решений по теме дипломного проекта

2 Моделирование предметной области и разработка требований к
программному средству

3 Проектирование и разработка программного средства

4 Тестирование программного средства

5 Методика использования разработанного программного средства

6 Техничко-экономическое обоснование разработки и реализации на рынке
программного средства

Заключение

Список использованных источников

Приложение А. Отчет о проверке на антиплагиат

Приложение Б. Исходный код

СОДЕРЖАНИЕ

Введение.....	7
1 Анализ литературных источников и программных решений по теме дипломного проекта.....	9
1.1 Описание и анализ предметной области.....	9
1.2 Обзор существующих аналогов.....	10
1.3 Спецификация требований к разрабатываемому программному средств.....	16
1.4 Обоснование выбора языка и сред разработки.....	19
2 Моделирование предметной области и разработка требований к программному средству.....	23
2.1 Спецификация требований.....	23
2.2 Разработка инфологической модели базы данных.....	27
2.3 Разработка физической модели базы данных.....	28
3 Проектирование и разработка программного средства.....	32
3.1 Разработка архитектуры программного средства.....	32
3.2 Разработка базы данных.....	33
3.3 Разработка серверной части приложения.....	38
4 Тестирование программного средства.....	44
5 Методика использования разработанного программного средства.....	50
6 Техничко-экономическое обоснование разработки и реализации на рынке программного средства.....	57
6.1 Характеристика разрабатываемого продукта.....	57
6.2 Расчет инвестиций в разработку программного средства для его реализации на рынке.....	58
6.3 Расчет экономического эффекта от реализации программного средства на рынке.....	60
6.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке.....	61
Заключение.....	63
Список использованной литературы.....	64
Приложение А (обязательное) Отчет о проверке на антиплагиат.....	66
Приложение Б (обязательное) Исходный код.....	67

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

Авторизация – предоставление определенному лицу или группе лиц прав на выполнение определенных действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аутентификация – проверка подлинности предъявленного пользователем идентификатора.

База данных – совокупность взаимосвязанных данных, организованных в соответствии со схемой базы данных таким образом, чтобы с ними мог работать пользователь.

Кроссплатформенность – способность программного обеспечения работать более чем на одной аппаратной платформе и (или) операционной системе.

Программное средство (ПС) – компьютерная программа, либо совокупность связанных программ, предназначенная для автоматизации определенной области профессиональной деятельности

Спецификация – документ, который желательно полно, точно и верифицируемо определяет требования, дизайн, поведение или другие характеристики компонента или системы, и, часто, инструкции для контроля выполнения этих требований

БД – база данных.

ООП – объектно-ориентированное программирование.

ПО – программное обеспечение.

СУБД – система управления базами данных.

API – программный интерфейс приложения.

JSON – текстовый формат обмена данными, основанный на языке программирования JavaScript.

ВВЕДЕНИЕ

Актуальность темы дипломного проекта обусловлена возрастающей потребностью молодых специалистов в эффективных инструментах поиска работы, соответствующих современным требованиям цифровой экономики. В условиях высокой конкуренции на рынке труда выпускникам учебных заведений необходимо профессионально презентовать свои навыки и оперативно находить подходящие вакансии, в то время как работодатели сталкиваются с проблемой идентификации действительно перспективных кандидатов среди большого потока соискателей.

Объектом исследования в данной работе выступают современные платформы и подходы к поиску работы и трудоустройству. Предметом исследования является процесс проектирования и разработки специализированного программного обеспечения, ориентированного на специфические потребности молодых специалистов, делающих первые шаги в своей карьере.

Целью настоящего дипломного проекта является создание интерактивной онлайн-платформы, которая существенно упростит процесс поиска работы для начинающих специалистов за счет автоматизации ключевых этапов, включая создание и управление резюме, интеллектуальный подбор вакансий на основе профиля соискателя и налаживание эффективной коммуникации с потенциальными работодателями. Для достижения поставленной цели необходимо последовательно решить ряд взаимосвязанных задач:

- провести всесторонний анализ существующих решений и тенденций на рынке онлайн-рекрутинга, выявив их сильные и слабые стороны, а также неудовлетворенные потребности молодых специалистов;
- на основе проведенного анализа сформулировать детальные функциональные и нефункциональные требования к разрабатываемой системе, учитывая специфику целевой аудитории;
- разработать логическую и физическую архитектуру программного обеспечения, определив основные компоненты системы, их взаимодействие и используемые технологии;
- реализовать ключевые функциональные модули платформы, включая подсистему управления профилями соискателей и работодателей, модуль создания и редактирования резюме, систему поиска и подбора вакансий, а также инструменты для коммуникации между пользователями;
- провести тестирование разработанной системы для выявления и устранения возможных ошибок и несоответствий требованиям;
- выполнить экономическое обоснование целесообразности и эффективности разработки предложенного программного средства.

Для реализации задуманного функционала разрабатываемой платформы поиска работы для молодых специалистов будет задействован комплекс современных и зарекомендовавших себя технологий веб-разработки. В

качестве основного языка программирования для серверной части приложения выбран Kotlin, благодаря его лаконичности, безопасности, производительности и отличной совместимости с Java-экосистемой [1].

Для построения backend-приложения будет использован легковесный и асинхронный фреймворк Ktor, который оптимально подходит для создания масштабируемых веб-сервисов [2]. В качестве системы управления базами данных выбрана PostgreSQL, обеспечивающая надежное, транзакционное и структурированное хранение данных, а также обладающая широкими возможностями для оптимизации запросов [3]. Доступ к базе данных из Kotlin-кода будет осуществляться с использованием библиотеки JOOQ, генерирующей типобезопасный API на основе схемы базы данных, что повышает надежность и упрощает разработку [4]. Для обеспечения неблокирующего взаимодействия с PostgreSQL будет использован реактивный драйвер R2DBC.

На клиентской стороне приложения для создания интерактивного и отзывчивого пользовательского интерфейса будет использован современный JavaScript-фреймворк React. React позволяет разрабатывать сложные UI как набор переиспользуемых компонентов, эффективно управлять состоянием приложения и динамически обновлять отображение на основе пользовательских действий и данных, полученных с сервера [5].

Взаимодействие между клиентской и серверной частями приложения будет осуществляться посредством REST API, что обеспечит простоту, гибкость и масштабируемость обмена данными в формате JSON [6].

Контейнеризация приложения с использованием Docker обеспечит удобство развертывания и масштабирования в различных окружениях [7]. Выбор данных технологий обусловлен их зрелостью, активным сообществом, богатой экосистемой библиотек и фреймворков, а также способностью обеспечить высокую производительность, надежность и удобство разработки и поддержки создаваемого программного средства.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности соответствует норме, установленной кафедрой информатики. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников». Процент оригинальности работы представлен в приложении А.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ПРОГРАММНЫХ РЕШЕНИЙ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТА

1.1 Описание и анализ предметной области

Программное средство для поиска работы молодым специалистам ориентировано на пользователей, которые только начинают свою карьеру и сталкиваются с проблемами трудоустройства. Основные трудности, с которыми сталкиваются молодые специалисты, включают недостаток опыта, отсутствие знаний о процессе трудоустройства, сложности в создании конкурентоспособного резюме и нехватку навыков прохождения собеседований.

Большинство существующих решений на рынке либо ориентированы на широкий круг соискателей, либо требуют значительного уровня подготовки в вопросах составления резюме и поиска вакансий. Это создает дополнительный барьер для молодых специалистов, которым необходимы не только актуальные предложения о работе, но и инструменты для повышения конкурентоспособности на рынке труда.

Одним из ключевых факторов успешного трудоустройства является персонализация процесса подбора вакансий и возможность получения обратной связи от работодателей. Многие платформы предлагают фильтры по ключевым параметрам вакансий, но не учитывают уровень подготовки соискателя, что может приводить к неподходящим рекомендациям. В результате молодой специалист тратит много времени на отклик на неподходящие вакансии, что снижает эффективность поиска работы и может привести к разочарованию в процессе.

Одной из важных задач является минимизация когнитивной нагрузки на пользователя. В отличие от профессионалов, которые знакомы с процессом трудоустройства, молодые специалисты часто испытывают трудности при анализе требований вакансий и адаптации своего резюме под конкретную позицию. Введение автоматизированных подсказок, основанных на лучших практиках, позволит сократить время на подготовку и повысить вероятность успешного трудоустройства.

Другим важным направлением является автоматизация процесса отклика на вакансии. Молодые специалисты часто отправляют резюме в большое количество компаний, но не получают обратной связи. Возможность отслеживания статуса откликов, получения автоматизированных уведомлений о просмотре резюме и получения шаблонов сопроводительных писем позволит сделать процесс более прозрачным и эффективным.

Безопасность данных соискателей также играет важную роль. Хранение персональной информации, такого как резюме, контакты и история откликов, требует высокой степени защиты. Использование шифрования данных и многофакторной аутентификации позволит снизить риски утечки информации.

Таким образом, программное средство для поиска работы молодым специалистам возможно станет не просто платформой для публикации вакансий, а полноценным инструментом, включающим интеллектуальный подбор, обучение, автоматизацию процесса подачи заявок и защиту персональных данных. Это позволит сделать поиск работы более эффективным и доступным для начинающих специалистов, повысив их шансы на успешное трудоустройство.

1.2 Обзор существующих аналогов

Разработка программного обеспечения начинается с тщательного анализа существующих решений на рынке. Этот анализ помогает выявить как преимущества, так и недостатки конкурирующих продуктов. Определение сильных и слабых сторон является ключевым этапом для создания более эффективного и востребованного программного обеспечения. В процессе разработки также важно учитывать потребности потенциальных пользователей и тенденции рынка. Для специалистов в области разработки программного обеспечения существует множество платформ и сервисов для поиска работы. Эти ресурсы предлагают широкий спектр вакансий от различных компаний. Ниже будет представлен обзор наиболее популярных и эффективных из них. Этот обзор поможет соискателям сориентироваться на рынке труда и найти подходящие карьерные возможности.

1.2.1 Rabota.by

Rabota.by является ведущей онлайн-платформой в Беларуси, специализирующейся на поиске работы [8]. Этот сервис предлагает обширную базу данных вакансий, охватывающую различные отрасли и уровни квалификации. Для работодателей Rabota.by предоставляет эффективные инструменты для поиска и найма подходящих кандидатов. В свою очередь, соискатели могут воспользоваться удобным интерфейсом для поиска интересующих их предложений. Ключевая задача платформы заключается в том, чтобы сделать процесс трудоустройства максимально простым и эффективным для обеих сторон. Благодаря удобной системе поиска и коммуникации, Rabota.by способствует быстрому нахождению работы и сотрудников. Платформа активно развивается, предлагая новые функции и возможности для своих пользователей. Главная страница сайта показана на рисунке 1.1.

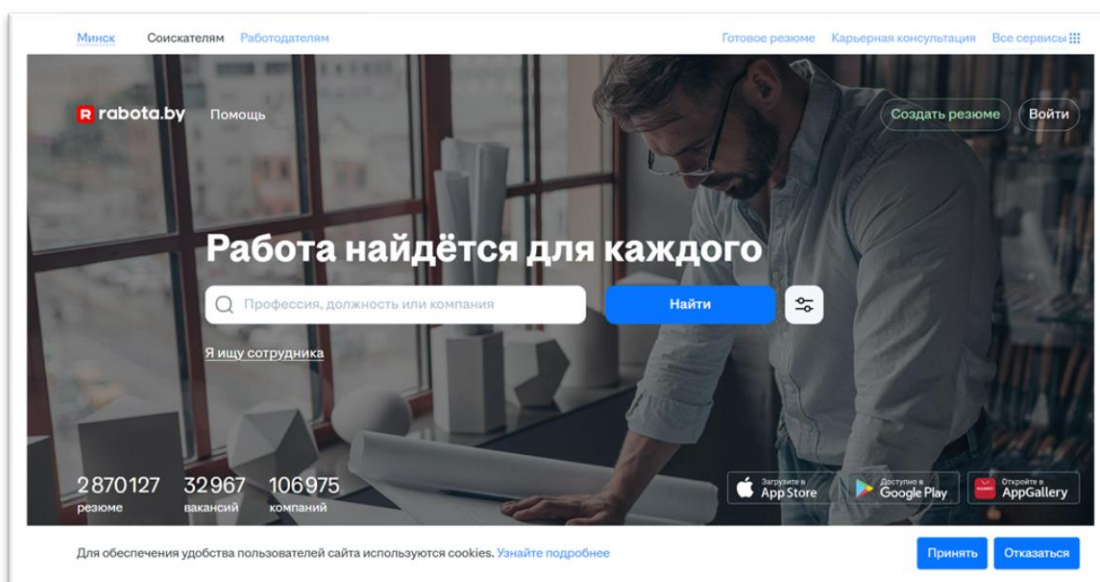


Рисунок 1.1 – Главная страница сайта Rabota.by

Одним из ключевых достоинств Rabota.by является его интуитивно понятный интерфейс, что делает навигацию по сайту простой и удобной для всех пользователей. Это можно увидеть на рисунке 1.2, где представлена страница с поиском вакансий. На странице есть возможность ввести ключевые слова для поиска, а также воспользоваться фильтрами. Помимо этого имеются рекомендованные вакансии на основе профиля и резюме пользователя. Платформа располагает внушительной и регулярно обновляемой базой вакансий от надежных и проверенных работодателей.

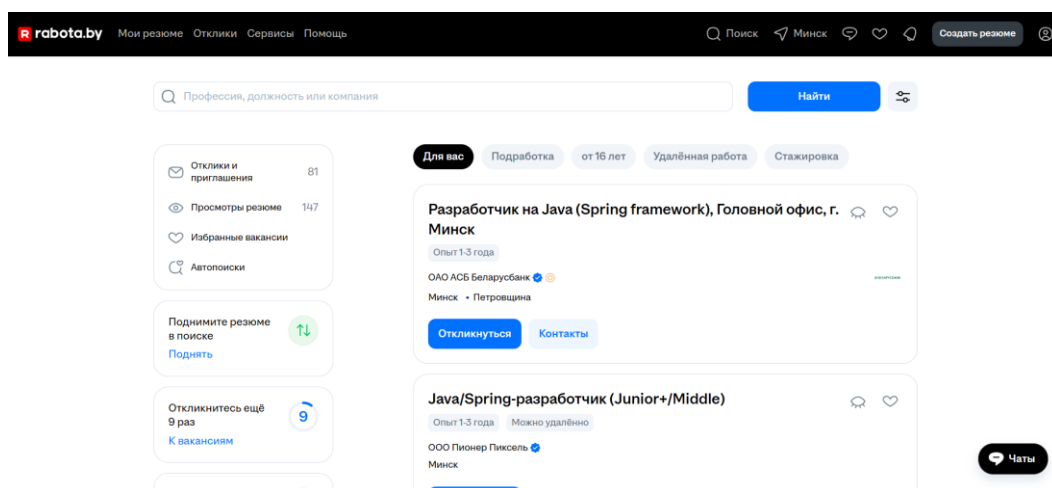


Рисунок 1.2 – Страница с поиском вакансий на Rabota.by

Кроме того, пользователи могут настроить систему уведомлений, чтобы оперативно получать информацию о новых, соответствующих их критериям предложениях. Rabota.by предоставляет возможность создавать и редактировать резюме непосредственно на сайте, а также применять

различные фильтры для поиска вакансий по заданным параметрам, что существенно облегчает процесс отбора.

На странице с резюме пользователя отображенном на рисунке 1.3 имеется возможность создать новые резюме, поднять существующие резюме выше в рекомендациях за дополнительную плату, а также воспользоваться услугами составления резюме и карьерной консультации. Однако не каждый начинающий специалист может себе позволить данные услуги, так как цена на них начинается от 75 белорусских рублей.

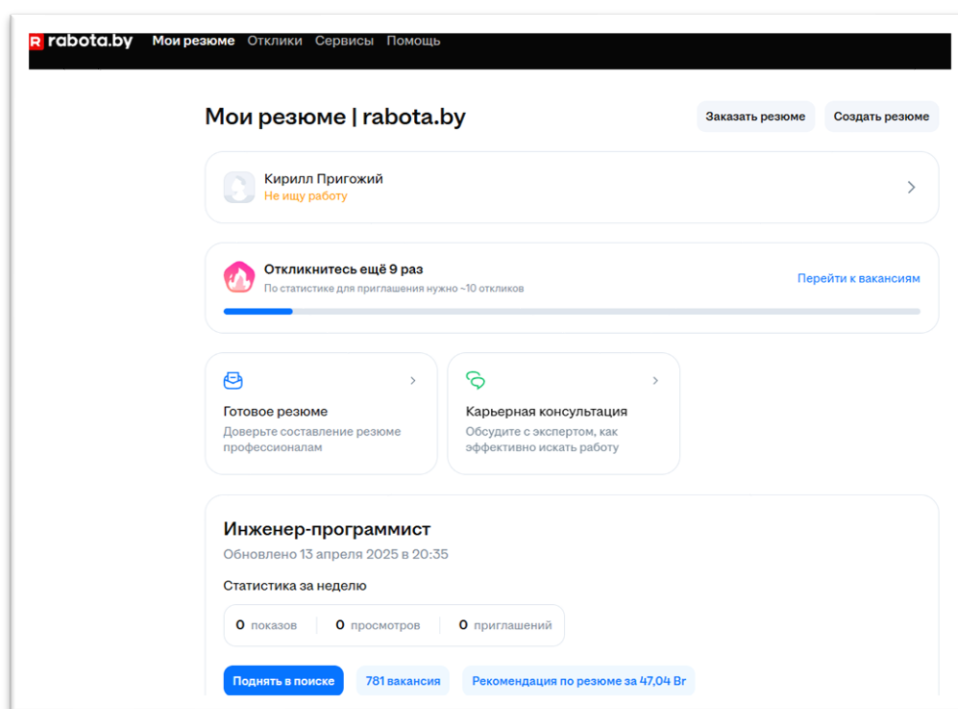


Рисунок 1.3 – Страница с резюме на Rabota.by

Тем не менее, для молодых специалистов, только начинающих свою карьеру, сервис имеет некоторые ограничения. В частности, платформа не предоставляет бесплатных инструментов, которые могли бы помочь в улучшении резюме или проведении детального анализа полученных откликов, что могло бы оказаться полезным для начинающих соискателей в их подготовке к трудоустройству.

Похожий интерфейс на rabota.by виден работодателю. В списке вакансий отображаются созданные работодателем вакансии, а так же их статус как показано на рисунке 1.4.

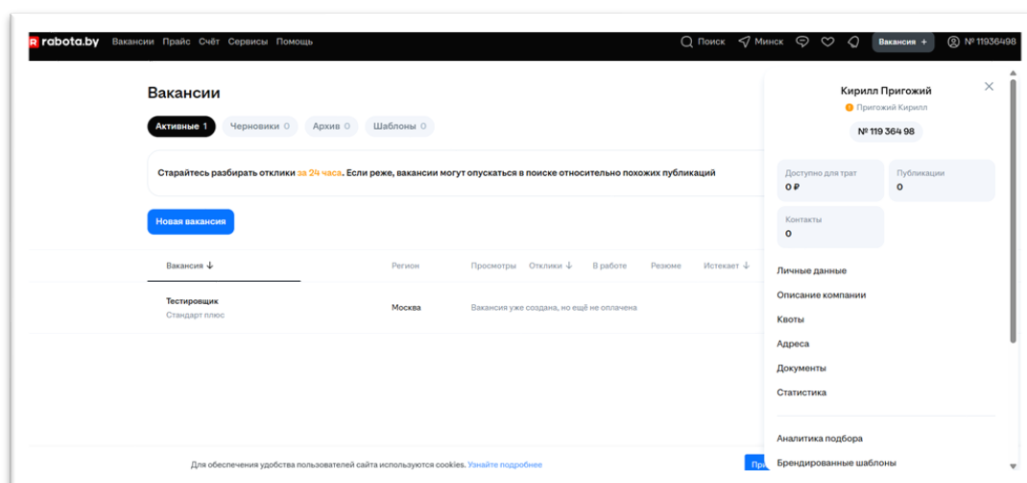


Рисунок 1.4 – Страница работодателя на Rabota.by

1.2.2 LinkedIn

LinkedIn представляет собой одну из самых масштабных профессиональных сетей, которая эффективно используется как для поиска работы, так и для налаживания ценных деловых связей [9]. Главная страница представлена на рисунке 1.5.

Среди ключевых преимуществ этой платформы следует отметить ее обширную аудиторию работодателей, представляющих самые разные отрасли. Пользователи имеют возможность создать подробный и информативный профиль, который зачастую служит полноценной заменой традиционному резюме. Кроме того, LinkedIn обладает развитой системой рекомендаций, позволяющей коллегам и руководителям подтверждать профессиональные навыки и достижения. Активное ведение профиля и публикация экспертного контента способствуют формированию и продвижению личного профессионального бренда пользователя в онлайн-пространстве.

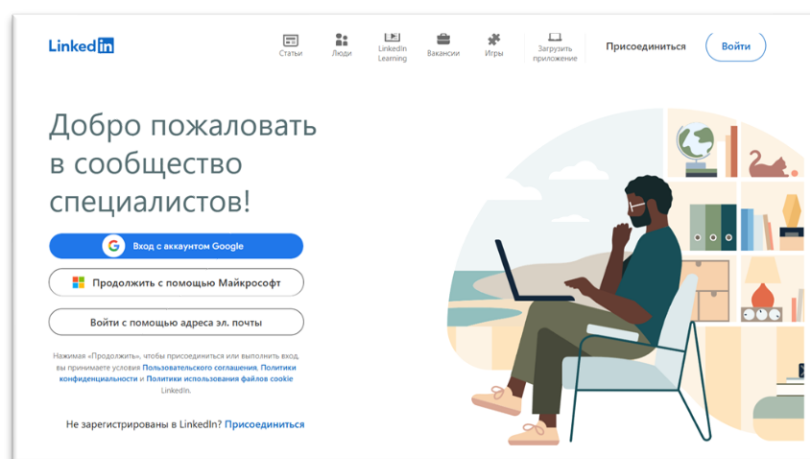


Рисунок 1.5 – Главная страница сайта LinkedIn

Хотя LinkedIn предоставляет ценные ресурсы для молодых специалистов, включая образовательные материалы и возможности для нетворкинга, которые могут способствовать их карьерному росту, стоит отметить некоторые особенности платформы. Во-первых, алгоритм рекомендаций вакансий зачастую в большей степени ориентирован на профессионалов с опытом и не всегда предлагает релевантные варианты для начинающих специалистов в Беларуси. Во-вторых, для получения максимальной отдачи от LinkedIn требуется активное участие в профессиональном сообществе и регулярное обновление профиля, что может представлять определенную сложность для новичков. Кроме того, многие полезные функции, такие как расширенная аналитика профиля и доступ к продвинутым инструментам поиска, доступны только по платной подписке, что может стать финансовым барьером для молодых специалистов, особенно при поиске работы в Беларуси.

Таким образом, несмотря на глобальный охват и множество возможностей, LinkedIn в большей степени ориентирован на международный рынок труда, и его эффективность для поиска первой работы в Беларуси может быть ниже по сравнению с локальными платформами.

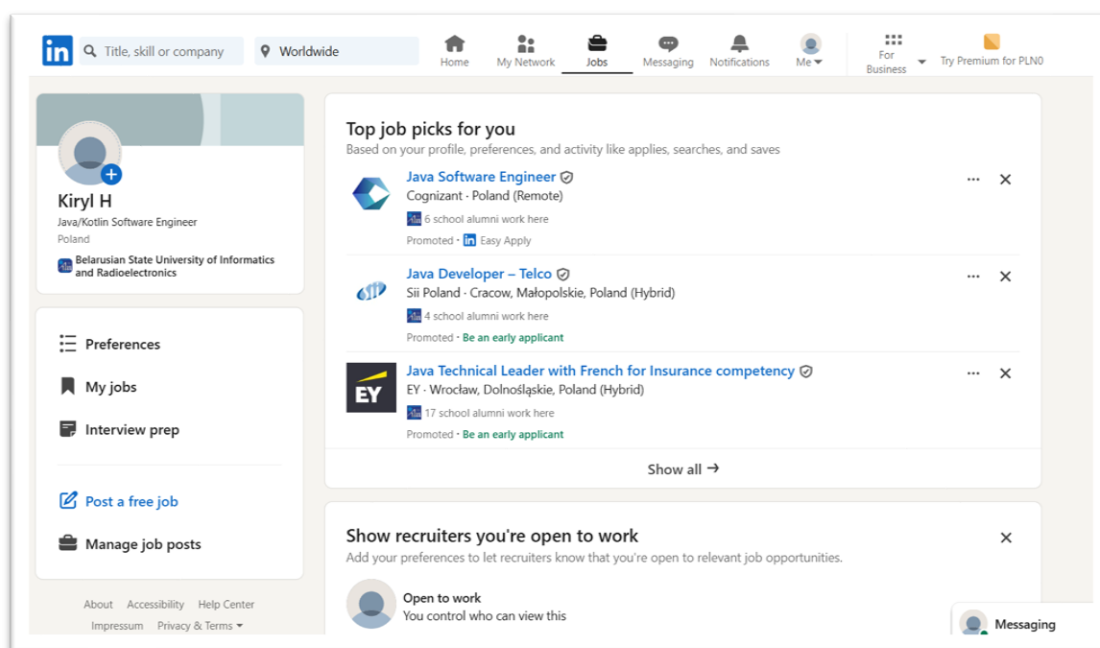


Рисунок 1.6 – Страница с поиском вакансий на LinkedIn

1.2.3 Praca.by

Praca.by представляет собой белорусский веб-сайт, специализирующийся на размещении вакансий от работодателей и помощи соискателям в поиске подходящей работы на территории Беларуси [10]. Интерфейс сайта представлен на рисунках 1.7 и 1.8. Эта платформа является частью крупной международной рекрутинговой сети под названием The

Network, что обеспечивает ей доступ к передовым технологиям и опыту в сфере трудоустройства.

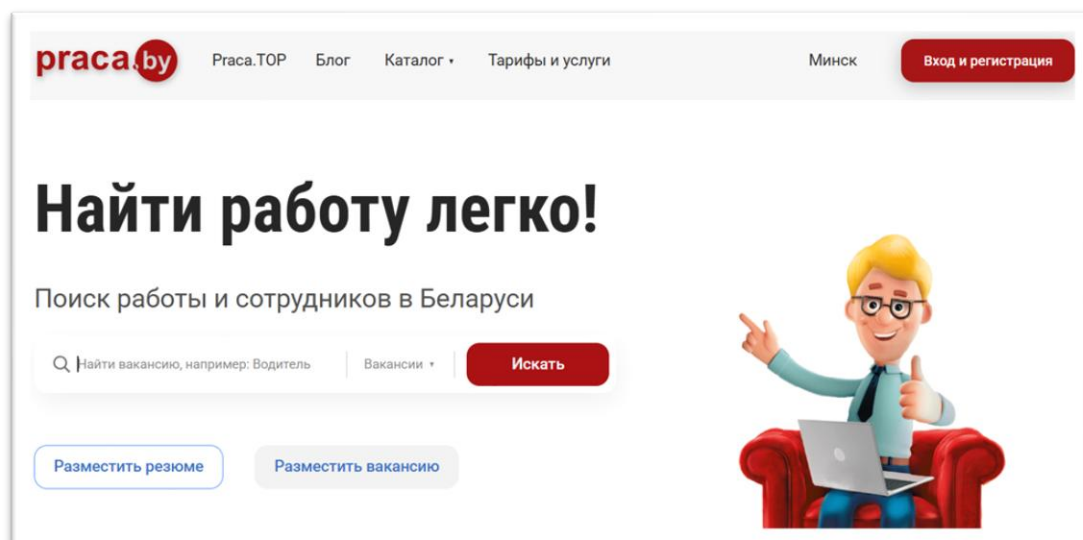


Рисунок 1.7 – Главная страница сайта Praca.by

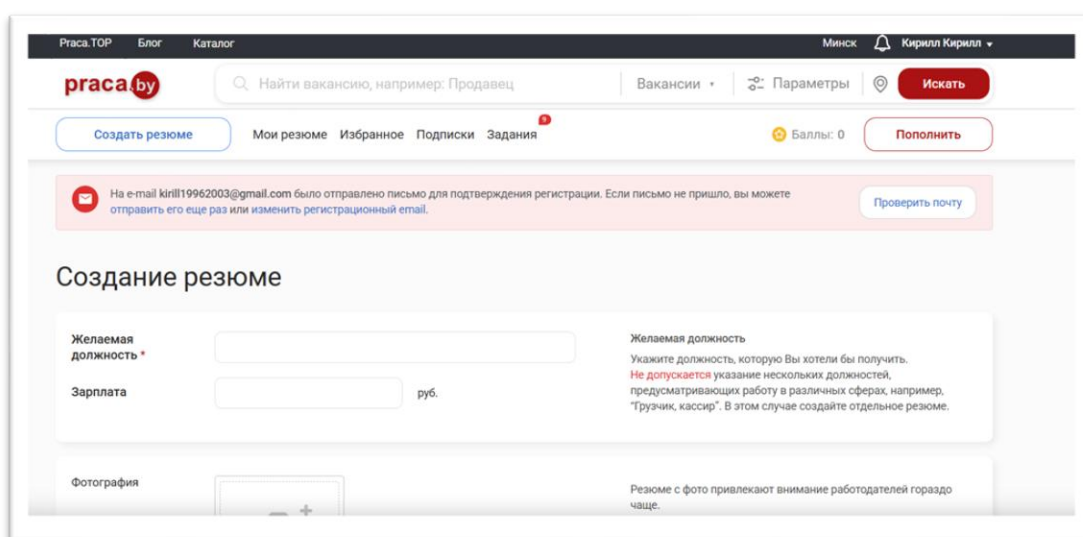


Рисунок 1.8 – Страница для создания резюме Praca.by

Одним из заметных минусов Praca.by является меньшее количество размещенных вакансий, если сравнивать его с более крупной платформой Rabota.by, что объясняется тем, что не все белорусские работодатели активно используют данный ресурс для поиска сотрудников.

Следует отметить, что внешний вид и структура Praca.by оставляют желать лучшего с точки зрения современного веб-дизайна. Интерфейс платформы выглядит несколько архаично и может показаться менее интуитивно понятным по сравнению с более современными аналогами, что

потенциально может затруднять навигацию и общее восприятие сайта пользователями.

Анализ ключевых онлайн-платформ для поиска работы, таких как широко известный в Беларуси Rabota.by, международная сеть профессиональных контактов LinkedIn и белорусский ресурс Praca.by, выявляет доминирование крупных игроков на рынке. Эти платформы предлагают обширные базы данных вакансий, охватывающие сферы деятельности и уровни квалификации, стремясь удовлетворить потребности максимально широкой аудитории соискателей.

Однако, при ближайшем рассмотрении становится очевидным, что универсальность этих сервисов имеет и обратную сторону. Ориентируясь на массового пользователя, они не всегда способны в полной мере учесть специфические потребности и карьерные устремления молодых специалистов, находящихся на начальном этапе своего профессионального развития. Молодые люди, только начинающие свой трудовой путь, часто сталкиваются с уникальными вызовами при поиске первой работы, включая ограниченный опыт, необходимость развития базовых профессиональных навыков и потребность в карьерном ориентировании.

В то время как Rabota.by предлагает обширную базу вакансий в Беларуси и инструменты для взаимодействия с работодателями, его фокус остается на широком рынке труда, и специализированные ресурсы для поддержки начинающих карьеру могут быть ограничены. LinkedIn, будучи мощной международной платформой для установления деловых связей и поиска работы, в большей степени ориентирован на опытных профессионалов и международный рынок, что может снижать его релевантность для поиска первой работы в Беларуси. Praca.by, хотя и является белорусским ресурсом, уступает по объему вакансий Rabota.by и может иметь менее современный интерфейс, что также не всегда отвечает потребностям молодых, технологически подкованных специалистов.

Таким образом, проведенный анализ подчеркивает существующий разрыв между возможностями, предоставляемыми крупными платформами для поиска работы, и особыми потребностями молодых специалистов. Существует потребность в более специализированных ресурсах или дополнительных инструментах в рамках существующих платформ, которые могли бы более эффективно поддерживать молодых людей в их первом трудоустройстве, предлагая, например, возможности для стажировок, программы развития для начинающих специалистов, консультации по составлению резюме и подготовке к собеседованиям с учетом их ограниченного опыта.

1.3 Спецификация требований к разрабатываемому программному средству

Опираясь на проведенный анализ существующих платформ для поиска работы, таких как Rabota.by, LinkedIn и Praca.by, и выявленные в них

недостатки, особенно в контексте потребностей молодых специалистов, формулируются ключевые требования к разрабатываемому программному средству. Это новое решение призвано стать специализированным инструментом, ориентированным на оказание всесторонней помощи молодым людям в их первом трудоустройстве.

Разрабатываемая платформа для поиска работы молодыми специалистами представляет собой комплексное веб-приложение, спроектированное с целью автоматизации и значительной оптимизации процесса трудоустройства для этой целевой аудитории. Система нацелена на эффективное решение ряда ключевых проблем, с которыми наиболее часто сталкиваются начинающие специалисты на рынке труда. Среди этих проблем выделяются отсутствие значительного релевантного опыта работы, трудности в создании эффективного резюме, которое бы выгодно подчеркивало их потенциал, неэффективность традиционных методов поиска вакансий, которые часто ориентированы на более опытных кандидатов, а также недостаток своевременной и конструктивной обратной связи от работодателей, что затрудняет понимание требований рынка и совершенствование навыков. Разрабатываемая платформа стремится предоставить молодым специалистам инструменты и ресурсы, которые помогут им преодолеть эти барьеры и успешно начать свою карьеру.

Разрабатываемая платформа для молодых специалистов будет обладать широким спектром функциональных возможностей, призванных обеспечить удобный и эффективный процесс поиска работы.

Во-первых, процесс взаимодействия с платформой начнется с регистрации и авторизации с использованием электронной почты.

После успешной авторизации каждый молодой специалист сможет создать подробный и информативный профиль, который станет его цифровым резюме. Профиль будет включать основные личные данные, детальную информацию об образовании, перечень ключевых навыков, описание имеющегося опыта работы, а также возможность загрузки портфолио с примерами выполненных проектов или учебных работ, что позволит наглядно продемонстрировать свои способности.

Важным аспектом станет настройка конфиденциальности, предоставляющая пользователям полный контроль над видимостью их резюме для потенциальных работодателей. Молодые специалисты смогут самостоятельно определять, кто сможет просматривать их профиль, что обеспечит им чувство безопасности и приватности.

Центральным элементом платформы станет персонализированная лента вакансий. Интеллектуальная система будет анализировать указанные в профиле навыки, полученное образование и карьерные предпочтения пользователя, чтобы предлагать наиболее релевантные и интересные вакансии, экономя время и усилия на самостоятельном поиске.

Для дальнейшей оптимизации поиска будут реализованы фильтрация и сортировка вакансий по множеству параметров, включая сферу деятельности, желаемый уровень заработной платы, предпочтительный тип занятости. Это

позволит молодым специалистам быстро находить именно те предложения, которые соответствуют их критериям.

Процесс подачи заявки на понравившуюся вакансию будет максимально упрощен благодаря функции отправки откликов в один клик с автоматическим прикреплением резюме и возможностью добавления сопроводительного письма, что позволит быстро реагировать на интересные предложения.

Для обеспечения эффективной коммуникации между молодыми специалистами и потенциальными работодателями будет реализован встроенный чат. Этот инструмент позволит оперативно задавать вопросы, уточнять детали вакансий и договариваться о дальнейших шагах, таких как собеседования.

Система уведомлений о статусе отклика будет держать соискателей в курсе событий, информируя их о том, был ли просмотрен их отклик, получили ли они приглашение на собеседование или, к сожалению, получили отказ. Это позволит им отслеживать свой прогресс и своевременно корректировать свою стратегию поиска.

Для предоставления молодым специалистам актуальной информации о рынке труда будут доступны различные статистические данные, отражающие текущее положение дел, среднее количество откликов на вакансии в интересующих их сферах и другие полезные метрики.

Платформа также будет включать рекомендации по составлению эффективного резюме с учетом специфики первого трудоустройства и советы по успешному прохождению собеседований, помогая молодым специалистам подготовиться к ключевым этапам процесса найма.

Безопасность пользователей будет обеспечена за счет защиты от несанкционированного доступа к их личным данным и внедрения механизмов для выявления и блокировки мошеннических вакансий, что создаст доверительную и безопасную среду для поиска работы.

Наконец, для обеспечения максимальной приватности будет реализована настройка приватности, позволяющая пользователям скрывать свое резюме от определенных компаний, например, от текущего работодателя, если они находятся в активном поиске новой работы.

Помимо обширного набора функциональных возможностей, разрабатываемая платформа для молодых специалистов должна соответствовать ряду критически важных нефункциональных требований, которые обеспечат ее стабильность, удобство использования, безопасность и возможность дальнейшего развития.

Во-первых, архитектура приложения должна быть тщательно спроектирована с особым вниманием к простоте поддержки и дальнейшего развития функционала. Для достижения этой цели необходимо применить модульную структуру кода с четким разделением компонентов и ответственностей. Такой подход позволит изолировать различные части системы, что существенно упростит внесение изменений, исправление ошибок и добавление новых функций без возникновения риска дестабилизации работы всей платформы. Четкая модульность также облегчит работу команды

разработчиков и снизит затраты на сопровождение проекта в долгосрочной перспективе.

Во-вторых, интерфейс платформы должен быть разработан с учетом того, что многие молодые специалисты впервые сталкиваются с процессом поиска работы. Следовательно, ключевым требованием является максимальная простота и удобство использования. Для этого необходимо руководствоваться современными UX/UI-практиками, включая минималистичный и интуитивно понятный дизайн, использование четких и однозначных призывов к действию, а также внедрение пошаговых подсказок и руководств в процессе заполнения профиля и использования основных функций платформы. Дружественный и понятный интерфейс снизит порог входа для молодых специалистов и повысит их удовлетворенность работой с платформой.

В-третьих, безопасность данных пользователей является абсолютным приоритетом. Все персональные данные, включая резюме, контактную информацию и переписку с работодателями, должны передаваться по зашифрованным каналам с использованием современных протоколов безопасности таких как HTTPS [11]. Пароли пользователей должны храниться в хешированном виде с применением надежных и криптостойких алгоритмов, что предотвратит их несанкционированный доступ даже в случае утечки данных. Кроме того, необходимо внедрить комплексные механизмы защиты от распространенных веб-уязвимостей, таких как SQL-инъекции [12] и межсайтовый скриптинг [13], чтобы обеспечить целостность и конфиденциальность пользовательских данных.

Предложенный сбалансированный набор как функциональных, так и нефункциональных требований закладывает прочную основу для создания эффективной и востребованной платформы, которая не только существенно упрощает процесс поиска работы для молодых специалистов, но и оказывает им всестороннюю помощь в развитии их профессиональных навыков и уверенном старте карьеры. Интеграция элементов автоматизации рутинных задач, персонализации контента на основе индивидуальных потребностей и аналитики рынка труда и активности пользователя делает эту систему незаменимым и удобным инструментом для молодых людей, стремящихся успешно начать свою профессиональную деятельность.

1.4 Обоснование выбора языка и сред разработки

Для успешной реализации амбициозной платформы поиска работы, ориентированной на молодых специалистов, был тщательно отобран комплекс передовых технологий, полностью отвечающий специфическим требованиям проекта и соответствующий современным стандартам разработки программного обеспечения.

В качестве основного языка разработки для серверной части приложения был выбран Kotlin. Этот современный язык программирования отличается своей лаконичностью и выразительностью синтаксиса, что

способствует написанию более чистого и поддерживаемого кода. Кроме того, Kotlin обладает высоким уровнем безопасности за счет встроенной системы обработки `NullPointerException` и полной совместимостью с обширной JVM (Java Virtual Machine) экосистемой, что открывает доступ к множеству зрелых библиотек и инструментов [14].

Для создания высокопроизводительного и масштабируемого серверного API был выбран фреймворк Ktor. Этот асинхронный фреймворк идеально подходит для разработки легковесных RESTful сервисов благодаря своей модульной архитектуре, позволяющей включать только необходимые компоненты, и встроенной поддержке корутин для организации асинхронного кода [15]. Использование Ktor позволяет создавать быстрые и эффективные серверные приложения с минимальными накладными расходами, что критически важно для обработки большого количества запросов от пользователей.

В качестве надежного хранилища данных была выбрана реляционная СУБД PostgreSQL. Этот зрелый и проверенный временем инструмент обеспечивает ACID-совместимую реляционную модель данных, гарантирующую целостность и надежность хранимой информации. Поддержка сложных SQL-запросов является ключевым фактором для эффективной работы с данными о вакансиях, резюме пользователей и их взаимодействиях [16]. Кроме того, ценной особенностью PostgreSQL является поддержка JSONB, позволяющая гибко хранить полуструктурированные данные, такие как дополнительные характеристики вакансий или навыки пользователей. Встроенная полнотекстовая поисковая система значительно упростит реализацию быстрого и релевантного поиска по вакансиям, что является одной из ключевых функций платформы.

В качестве основной интегрированной среды разработки (IDE) для написания кода на Java и Kotlin был выбран признанный лидер в этой области – IntelliJ IDEA [17]. Эта мощная IDE предлагает первоклассную поддержку Kotlin, включая интеллектуальное автодополнение кода, значительно ускоряющее процесс написания, мощные инструменты для рефакторинга, позволяющие безопасно изменять структуру кода, и передовые средства анализа кода, помогающие выявлять потенциальные ошибки и проблемы на ранних этапах разработки. Глубокая интеграция IntelliJ IDEA с экосистемой JVM, включая бесшовную поддержку современных фреймворков, таких как Ktor, существенно повышает продуктивность разработчиков. Особенно ценными для данного проекта являются такие возможности IntelliJ IDEA, как контекстно-зависимое автодополнение, автоматизированный рефакторинг с гарантией сохранения функциональности, а также статические анализаторы кода, которые помогают предотвратить возникновение ошибок еще до этапа тестирования. Встроенные инструменты для работы с базами данных и Docker-контейнерами делают эту IDE универсальным инструментом, позволяющим эффективно взаимодействовать со всеми компонентами разрабатываемой системы в едином окружении.

Для обеспечения единообразной и воспроизводимой среды выполнения всех компонентов системы будет активно использоваться технология Docker. Этот инструмент позволяет упаковывать серверные приложения, базы данных и другие необходимые сервисы в изолированные контейнеры, что решает ряд ключевых задач, связанных с развертыванием и масштабированием приложения [18].

Для данного проекта использование Docker имеет особое значение, поскольку он:

- обеспечивает легкую интеграцию сторонних сервисов путем их включения в отдельные контейнеры и организации взаимодействия между ними;
- гарантирует воспроизводимость сборок, так как контейнеры содержат все необходимые зависимости и настройки, исключая проблемы, связанные с различиями в окружении разработчиков или серверов;
- значительно минимизирует накладные расходы по сравнению с традиционными виртуальными машинами, обеспечивая более эффективное использование системных ресурсов.

Выбранный технологический стек представляет собой современное и надежное решение, которое позволит команде разработчиков создать высокопроизводительную, масштабируемую и удобную в поддержке платформу для помощи молодым специалистам в поиске работы.

Для разработки клиентской части веб-приложения, отвечающей за взаимодействие с пользователем, был выбран JavaScript-фреймворк React. Это решение является критически важным для обеспечения высокого уровня интерактивности и отзывчивости интерфейса, что напрямую влияет на удобство использования платформы молодыми специалистами. Компонентная архитектура React позволяет структурировать пользовательский интерфейс как набор независимых и многократно используемых компонентов [19]. Такой подход является особенно ценным при разработке сложных UI-элементов, таких как многошаговые формы редактирования резюме, гибкие фильтры для поиска вакансий и интерактивные панели для предварительного просмотра информации. Богатая и зрелая экосистема React, включающая такие популярные библиотеки, как Redux для эффективного управления состоянием приложения на клиентской стороне, React Router для организации плавной и удобной навигации между страницами, а также обширный выбор готовых UI-библиотек, значительно ускоряет процесс разработки визуально привлекательного и функционального пользовательского интерфейса [20]. Все эти ключевые факторы делают React оптимальным выбором для создания динамичного, высокопроизводительного и легко масштабируемого пользовательского интерфейса платформы поиска работы для молодых специалистов.

Выбранный технологический стек представляет собой тщательно сбалансированное сочетание производительности серверной части, скорости разработки клиентской части и общей надежности системы, что в полной мере соответствует амбициозным требованиям проекта. Все отобранные

технологии активно поддерживаются крупным и развитым сообществом разработчиков, обладают обширной и качественной документацией и широко используются в современной индустрии разработки программного обеспечения. Это гарантирует не только доступность квалифицированных специалистов для разработки и поддержки платформы, но и долгосрочную жизнеспособность и возможность ее дальнейшего развития и масштабирования.

В заключение следует особо подчеркнуть, что разработанная модель данных платформы полностью охватывает все ключевые аспекты взаимодействия между различными участниками процесса поиска работы. Предложенная структура базы данных обеспечивает четкое разделение ролей пользователей (молодые специалисты и работодатели), позволяет вести детализированный учет профессиональных качеств и навыков соискателей, а также предоставляет возможность для полного и структурированного описания вакансий, предлагаемых работодателями. Интегрированная система коммуникации и уведомлений создает удобную и прозрачную среду для взаимодействия между сторонами, способствуя оперативному обмену информацией и организации дальнейших шагов, таких как собеседования. Наконец, гибкие настройки профилей как для соискателей, так и для работодателей, позволяют адаптировать функционал платформы под индивидуальные потребности и предпочтения каждого пользователя. Все эти элементы в совокупности формируют надежную и масштабируемую основу для разработки эффективной, удобной и востребованной платформы трудоустройства, которая будет способствовать успешному старту карьеры молодых специалистов.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

Планируемая к разработке платформа для поиска работы будет включать систему регистрации и аутентификации. Для всех пользователей предусмотрена стандартная регистрация по email.

Для соискателей будет разработан удобный конструктор резюме с шаблонами и рекомендациями по заполнению. На основе данных профиля система будет формировать подборки соответствующих вакансий. Планируется создание системы мониторинга откликов и инструментов для коммуникации с работодателями.

Работодателям будут предоставлены инструменты для создания и публикации вакансий с возможностью указания требований к кандидатам. Система будет включать функции поиска и фильтрации резюме по заданным параметрам, а также инструменты для оценки соответствия кандидатов.

Административная часть платформы будет включать модули для модерации контента, мониторинга активности пользователей, обработки жалоб и технической поддержки. Гостевой доступ позволит ознакомиться с базовым функционалом системы и просматривать обезличенные вакансии.

2.1 Спецификация требований

Разрабатываемое программное средство, в соответствии с требованиями, сформулированными в подразделе 1.3, будет реализовано с применением системы ролей, что предполагает четкое разделение пользователей на различные категории и, как следствие, предоставление уникального набора прав и доступа к функциональности для каждой из этих категорий. Данный подход позволяет обеспечить безопасность, разграничить ответственность и оптимизировать взаимодействие пользователей с платформой в зависимости от их целей и задач.

На платформе будут определены следующие основные роли:

1 Гость. Эта роль присваивается неавторизованным пользователям, которые впервые посещают платформу. Их возможности будут ограничены ознакомительным функционалом, позволяющим получить общее представление о возможностях системы.

2 Соискатель. Эта роль предназначена для молодых специалистов, находящихся в поиске работы. Пользователи с этой ролью получают доступ к инструментам для создания и управления резюме, поиска и просмотра вакансий, отклика на интересующие предложения и взаимодействия с работодателями.

3 Работодатель. Эта роль предназначена для представителей компаний, размещающих вакансии и осуществляющих поиск молодых специалистов. Пользователи с этой ролью получают доступ к функциям создания и

управления вакансиями, поиска и просмотра резюме соискателей, а также взаимодействия с ними.

4 Администратор. Эта роль предназначена для персонала, ответственного за управление и обслуживание платформы. Администраторы обладают полным доступом ко всем функциям системы, включая управление пользователями, контентом, настройками безопасности и технической поддержкой.

Визуализация возможностей анонимного (гостя) и авторизированных пользователей (соискателя, работодателя, администратора) представлена на диаграмме вариантов использования (рисунок 2.1), разработанной с использованием стандарта UML 2.1. Эта диаграмма наглядно демонстрирует, какие действия могут выполнять пользователи, находящиеся в различных ролях, и как они взаимодействуют с ключевыми функциями платформы. Анализ данной диаграммы позволяет получить четкое представление о функциональном наполнении системы для каждой категории пользователей и служит основой для дальнейшего проектирования интерфейса и логики работы приложения.

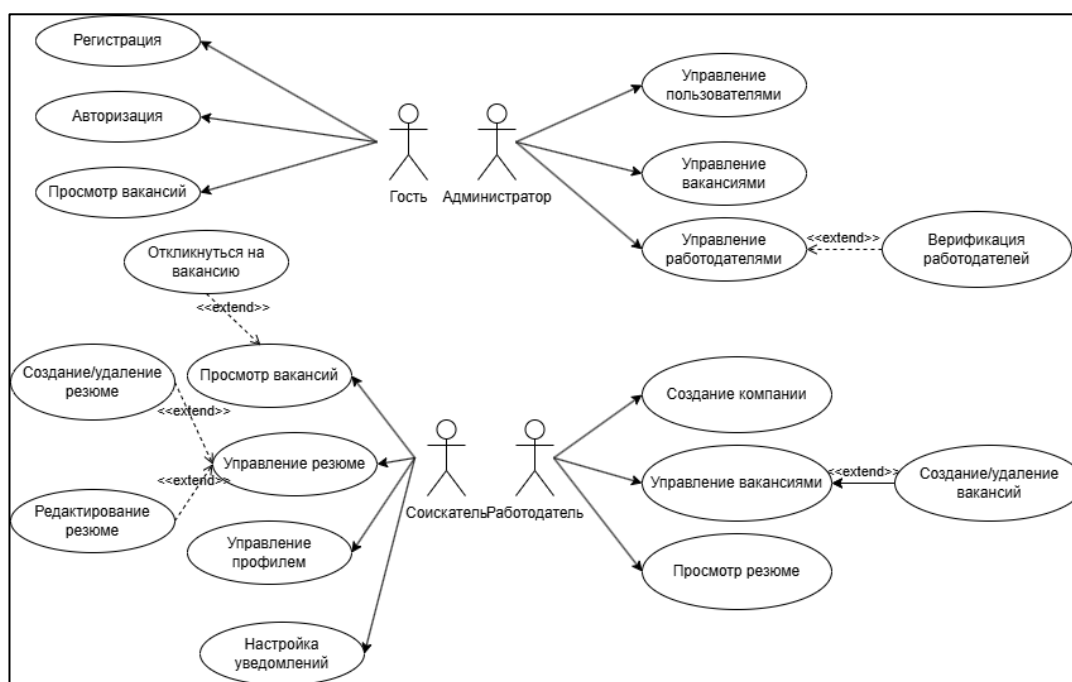


Рисунок 2.1 – Диаграмма использования программного средства

Рассмотрим подробно представленные на рисунке прецеденты.

Регистрация, аутентификация и авторизация – функции, которые доступны для роли «Гость» (пользователь, не зарегистрированный в системе). В приложении планируется вход с различных платформ.

После успешной регистрации на платформе, каждому пользователю назначается соответствующая роль администратором системы. Для молодых специалистов предусмотрена роль «Соискатель», а для представителей компаний, размещающих вакансии, – роль «Работодатель». Гости имеют

ограниченный доступ к функционалу до прохождения регистрации. Далее рассмотрены подробнее функциональные возможности, доступные для каждой из этих ролей.

Функции управления резюме включают в себя:

1 Создание резюме. Данная функция должна предоставлять каждому соискателю возможность создать одно или несколько резюме через интуитивно понятный интерфейс, включающий указание опыта работы с названием компании, должностью, периодом и описанием обязанностей, информации об образовании с названием учебного заведения, периодом обучения и полученной квалификацией, присвоение названия каждому резюме для удобства управления, добавление общей информации, такой как личные данные и профессиональное резюме, а также настройку видимости резюме для работодателей и отображение даты создания и последнего изменения для отслеживания актуальности.

2 Редактирование и удаление резюме. Функция дает соискателям возможность вносить изменения в ранее созданные резюме, поддерживая актуальность и релевантность представленной информации. Каждый пользователь может получить доступ к списку своих сохраненных резюме и выбрать любое из них для редактирования. После выбора резюме открывается интерфейс, аналогичный процессу создания, с уже заполненными полями. Соискатель может свободно изменять, добавлять или удалять информацию в любом разделе резюме, включая опыт работы (корректировка дат, обязанностей, достижений), образование (добавление новых учебных заведений, курсов), название резюме, общую информацию (обновление контактных данных, карьерной цели, навыков), а также изменять настройки видимости резюме. Сохранение изменений приводит к обновлению даты последнего изменения резюме, что позволяет как соискателю, так и работодателям видеть, насколько актуальна представленная информация.

Функции управления вакансиями включают в себя:

1 Поиск и просмотр вакансий с фильтрами. Пользователи имеют возможность осуществлять поиск по ключевым словам, названиям вакансий или компаний, а также просматривать весь список доступных вакансий. Для более точной фильтрации результатов поиска предусмотрен ряд параметров, включая желаемый уровень заработной платы (с возможностью указания диапазона), требуемый опыт работы, тематические тэги, а также предпочтительный формат работы. При выборе конкретной вакансии соискателю открывается подробная информация о ней, включающая описание обязанностей, требования к кандидату, условия работы, название компании, местоположение и контактные данные работодателя. Данная функция обеспечивает эффективный и целенаправленный поиск, позволяя молодым специалистам быстро находить вакансии, соответствующие их квалификации и карьерным целям.

2 Отклик на вакансию. Рассматриваемая функция предоставляет соискателям возможность выразить свою заинтересованность в конкретной вакансии и направить свою кандидатуру работодателю. После просмотра

детальной информации о вакансии, пользователь имеет возможность инициировать процесс отклика. Система позволяет прикрепить к отклику только одно из ранее созданных резюме, предоставляя соискателю выбор наиболее релевантного документа для данной вакансии. Помимо резюме, соискатель также может добавить сопроводительное письмо, в котором он может кратко рассказать о своей заинтересованности в данной позиции, подчеркнуть ключевые навыки и опыт, соответствующие требованиям вакансии, и выразить готовность к дальнейшим шагам. После отправки отклика, информация о нем фиксируется в системе, и соискатель получает уведомление о том, что его заявка отправлена работодателю. Статус отклика впоследствии может обновляться работодателем, что отображается в личном кабинете соискателя. Данная функция обеспечивает простой и централизованный способ подачи заявок на интересующие вакансии.

3 Создание и удаление вакансии. Эта функция позволяет работодателям размещать подробные объявления о работе, указывая название, описание, требования к кандидатам, условия работы, сферу деятельности, ключевые навыки, местоположение, контактную информацию и срок публикации, с возможностью сохранения черновиков перед публикацией, в то время как функция удаления вакансии дает работодателям возможность удалять неактуальные объявления из списка доступных вакансий на платформе.

Функция "Управление пользователями" в административной панели предоставляет администраторам возможность контролировать доступ к платформе, позволяя блокировать учетные записи пользователей (соискателей или работодателей) в случае нарушения правил использования или возникновения подозрительной активности, тем самым временно или постоянно ограничивая их возможность входа в систему и использования ее функционала, а также восстанавливать доступ к учетным записям путем их разблокировки, если причины блокировки были устранены или произошла ошибка.

Административная панель предоставляет администраторам расширенные возможности по контролю над контентом платформы, включая управление размещенными вакансиями. Функция "Управление вакансиями" позволяет администраторам просматривать список всех активных и неактивных вакансий, размещенных работодателями. В случае обнаружения нарушений правил публикации, получения жалоб от пользователей или по другим причинам, администратор имеет возможность скрыть вакансию от общего доступа. Скрытая вакансия перестает отображаться в результатах поиска для соискателей, однако информация о ней сохраняется в системе для внутреннего учета и возможного дальнейшего анализа.

Помимо временного скрытия, администратор также обладает правом удалить вакансию из системы. Данное действие является необратимым и приводит к полному удалению всей связанной с вакансией информации из базы данных. Удаление вакансий может потребоваться в случаях серьезных нарушений правил, размещения недостоверной информации или по истечении определенного срока хранения.

2.2 Разработка инфологической модели базы данных

Представленная схема описывает инфологическую модель базы данных разрабатываемой платформы поиска работы. Модель состоит из нескольких взаимосвязанных таблиц, каждая из которых хранит определенный набор данных о ключевых сущностях системы.

Основные сущности и их атрибуты представлены в таблице 2.1.

Сущность	Атрибуты
Пользователь	<ul style="list-style-type: none">- уникальный идентификатор пользователя (первичный ключ);- адрес электронной почты пользователя (уникальный);- хэш пароля пользователя для безопасности, роль пользователя в системе;- временная метка регистрации пользователя, текущий статус пользователя;- имя пользователя;- фамилия пользователя;- отчество пользователя, дата рождения пользователя.
Компания	<ul style="list-style-type: none">- уникальный идентификатор компании (первичный ключ);- название компании (уникальное);- количество сотрудников в компании;- описание компании;- адрес компании;- ссылка на веб-сайт компании.
Вакансия	<ul style="list-style-type: none">- уникальный идентификатор вакансии (первичный ключ);- название вакансии, минимальная предлагаемая зарплата;- максимальная предлагаемая зарплата;- валюта заработной платы,;- тип занятости;- местоположение вакансии;- описание вакансии;- требуемый уровень опыта;- количество просмотров вакансии;- идентификатор компании-работодателя (внешний ключ);- временная метка создания вакансии;- временная метка последнего редактирования вакансии.

Продолжение таблицы 2.1

Сущность	Атрибуты
Резюме	<ul style="list-style-type: none"> - уникальный идентификатор резюме (первичный ключ); - идентификатор пользователя-соискателя (внешний ключ); - название резюме; - краткое описание резюме; - временная метка создания резюме; - временная метка последнего редактирования резюме; - флаг активности резюме - опыт работы (структурированные данные).

Данная инфологическая модель представляет собой основу для физического проектирования базы данных и обеспечивает хранение и управление всеми необходимыми данными для функционирования платформы поиска работы.

2.3 Разработка физической модели базы данных

Представленная на рисунке 2.2 SQL схема базы данных платформы поиска работы обладает рядом ключевых особенностей, направленных на эффективное хранение и управление информацией о пользователях, компаниях, вакансиях, резюме и их взаимосвязях. Одной из отличительных черт является использование UUID (Universally Unique Identifier) в качестве первичных ключей для большинства основных сущностей (user, company, vacancy, resume) [21]. Это обеспечивает глобальную уникальность записей и упрощает масштабирование системы, а также интеграцию с другими сервисами.

Схема предусматривает четкое разделение ролей пользователей через поле role в таблице user, что позволяет реализовать различные уровни доступа и функциональности для соискателей, работодателей и администраторов. Для обеспечения безопасности используется поле password_hash для хранения хэшированных паролей.

Также внимание было уделено связям между сущностями. Отношение "один-ко-многим" между компанией и вакансиями (company -> vacancy по employer_id) позволяет одной компании размещать множество вакансий. Аналогично, отношение "один-ко-многим" между пользователем и резюме (user -> resume по user_id) дает возможность соискателю создавать несколько резюме.

Для реализации многозначных связей, таких как связь между вакансиями и метками, а также резюме и метками, используются связующие

таблицы (vacancy_label, resume_label). Это позволяет гибко классифицировать контент и упрощает поиск по интересам.

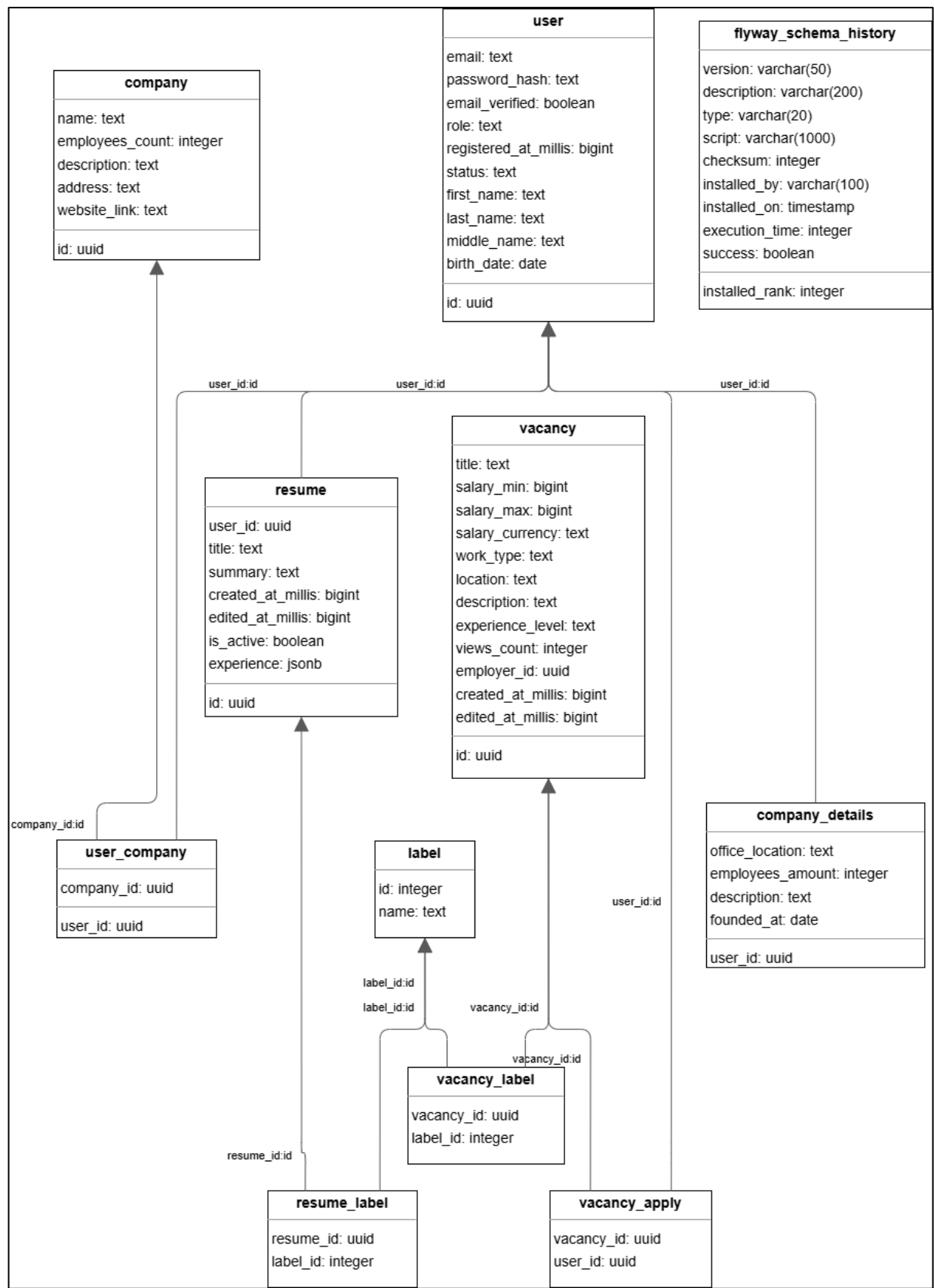


Рисунок 2.2 – Схема базы данных

При разработке платформы будут учитываться ключевые функциональные требования. Базовый функционал будет включать систему регистрации с верификацией пользователей, ролевую модель доступа и персонализированные рабочие пространства.

Для соискателей планируется создать конструктор резюме с шаблонами, систему поиска вакансий по различным параметрам, а также инструменты для отслеживания откликов. Работодатели получают возможность создавать вакансии, искать и фильтровать резюме, оценивать кандидатов.

Административные функции будут включать мониторинг активности пользователей, обработку нарушений и генерацию отчетов. Особое внимание будет уделено производительности системы - время отклика не должно превышать несколько секунд.

Надежность работы будет обеспечиваться регулярным резервным копированием данных и механизмами восстановления после сбоев. Безопасность данных будет реализована с использованием современных методов шифрования и защиты от несанкционированного доступа.

Предметная область программного средства включает ключевые сущности и соответствующие им атрибуты, которые отражают структуру и характеристики данных, используемых в системе.

Каждый пользователь платформы обладает уникальным идентификатором, который позволяет однозначно отличать его от других участников системы. При регистрации пользователь указывает имя, электронную почту и пароль, который хранится в захешированном виде для обеспечения безопасности. Система фиксирует дату регистрации и время последнего входа, что помогает отслеживать активность. Пользователи разделяются на три основные роли: соискатели, работодатели и администраторы, каждая из которых имеет свои права и возможности. Дополнительно сохраняется фотография профиля, контактный телефон и часовой пояс для корректного отображения времени. Статус аккаунта показывает, активен ли пользователь или временно заблокирован, а специальный флаг подтверждает верификацию электронной почты.

Со стороны соискателей система хранит расширенные данные, включая имя, фамилию и дату рождения. Учитывается пол, город проживания и готовность к переезду, что помогает в поиске подходящих вакансий. Соискатель указывает ожидаемый уровень зарплаты и предпочитаемый тип занятости: полная, частичная или удаленная работа. Видимость профиля может быть публичной или скрытой в зависимости от предпочтений пользователя. Важными характеристиками являются уровень образования, желаемая должность и общий опыт работы в годах, которые используются для автоматического подбора вакансий.

Работодатели предоставляют информацию о своей компании: официальное название, описание деятельности и веб-сайт. Учитывается размер компании по количеству сотрудников и основная отрасль деятельности. Логотип компании загружается по ссылке, а для связи

указывается контактное лицо с указанием его должности. Эти данные помогают соискателям оценивать потенциальных работодателей.

Каждое резюме содержит уникальный идентификатор и привязывается к конкретному соискателю. Основные элементы включают название резюме, профессиональное описание и желаемую должность. Указывается общий опыт работы, даты создания и последнего обновления. Резюме может быть основным или дополнительным, с настройками видимости для работодателей. Используемый шаблон определяет визуальное оформление, а ключевые навыки и дополнительная информация помогают выделиться среди других кандидатов.

Раздел опыта работы включает перечень компаний с указанием периодов работы, должностей и основных обязанностей. Особое внимание уделяется профессиональным достижениям и сфере деятельности работодателя. Образовательная часть содержит информацию об учебных заведениях, полученных степенях и специальностях. Указываются годы обучения и форма (очная, заочная), а также текущий статус для тех, кто еще продолжает обучение.

Работодатели создают вакансии с подробным описанием требований и обязанностей. Указывается тип занятости, зарплатный диапазон в определенной валюте и локация работы с возможностью удаленного формата. Система фиксирует даты публикации и закрытия, а также текущий статус вакансии. Соискатели отправляют отклики, которые проходят несколько статусов от отправки до финального решения. В процессе коммуникации сохраняются комментарии работодателей, даты собеседований и персональные сообщения.

3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного средства

Принимая во внимание требования к платформе поиска работы для молодых специалистов, включающие необходимость обеспечения отзывчивого пользовательского интерфейса, эффективной обработки данных и возможности масштабирования по мере роста пользовательской базы, выбор был остановлен на классической трехуровневой клиент-серверной архитектуре [22]. Визуальное представление данной архитектуры приведено на рисунке 3.1.

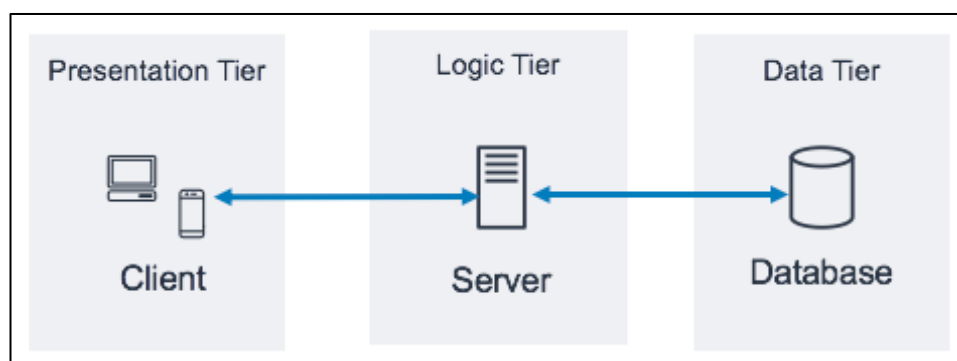


Рисунок 3.1 – Клиент-серверная архитектура

Эта архитектурная модель предполагает четкое разделение приложения на три логически обособленных, но взаимодействующих уровня, каждый из которых выполняет свою специфическую роль:

1 Клиентский уровень. Этот уровень непосредственно взаимодействует с конечным пользователем и представлен веб-браузером. Его основной задачей является отображение пользовательского интерфейса – всех элементов, с которыми пользователь взаимодействует (кнопки, формы, списки вакансий, профили и т.д.). Кроме того, клиентский уровень отвечает за первоначальную обработку действий пользователя (например, ввод данных в формы, клики) и отображение полученных от сервера результатов. Современные frontend-технологии, такие как React позволяют создавать богатые и интерактивные пользовательские интерфейсы, способные динамически обновляться без полной перезагрузки страницы, обеспечивая плавный и отзывчивый пользовательский опыт.

2 Серверный уровень. Этот уровень является центральным звеном архитектуры и отвечает за выполнение основной бизнес-логики приложения. Он принимает запросы от клиентского уровня, обрабатывает их в соответствии с правилами работы платформы (например, поиск вакансий по заданным критериям, обработка откликов, аутентификация пользователей), взаимодействует с уровнем базы данных для получения или сохранения

информации и формирует ответы, которые затем отправляются обратно клиенту. На сервере также реализуются механизмы безопасности и контроля доступа к данным. Для реализации серверной части был выбран Kotlin и фреймворк Ktor, которые обеспечивают производительность, масштабируемость и удобство разработки.

3 Уровень базы данных. Этот уровень отвечает за хранение, организацию и управление всеми постоянными данными, необходимыми для функционирования платформы. Сюда относятся данные о пользователях, компаниях, размещенных вакансиях, резюме соискателей, их откликах и другая информация. В качестве СУБД будет использована PostgreSQL, которая обеспечивает надежное, транзакционное и структурированное хранение данных, а также предоставляет мощные средства для запросов и управления информацией. Серверный уровень взаимодействует с уровнем базы данных для получения и сохранения необходимых данных, абстрагируясь от деталей физического хранения.

Выбор именно этой трехуровневой архитектуры продиктован стремлением к четкому разделению ответственности между различными аспектами приложения. Это позволяет каждой команде разработчиков специализироваться на своем уровне, упрощает процесс разработки, тестирования и поддержки. Кроме того, такая архитектура способствует лучшей масштабируемости, поскольку отдельные уровни могут быть масштабированы независимо друг от друга в зависимости от нагрузки. Например, при увеличении числа пользователей можно увеличить количество серверов приложений, не затрагивая базу данных или клиентскую часть. Наконец, разделение на уровни повышает безопасность системы, ограничивая прямой доступ клиента к базе данных и централизуя контроль доступа на серверном уровне.

3.2 Разработка базы данных

3.2.1 Производительность базы данных

Хотя система управления базами данных PostgreSQL изначально демонстрирует высокие стандарты производительности, для дальнейшей оптимизации работы платформы и обеспечения быстрого отклика на часто используемые запросы необходимо создание дополнительных индексов. Эти индексы позволят базе данных более эффективно находить требуемые записи, минимизируя необходимость полного сканирования таблиц, что особенно важно для таких интенсивно используемых сущностей, как вакансии.

Для начала будет рассмотрена таблица `vacansy`, которая является ключевым элементом для поиска работы соискателями. Для повышения эффективности операций поиска и фильтрации вакансий необходимо определить и создать соответствующие индексы для этой таблицы.

Для обеспечения высокой производительности при работе с таблицей `vacansy`, особенно при выполнении операций поиска и фильтрации, мы применим стратегию индексирования ключевых полей. Индексы

представляют собой специальные структуры, которые позволяют базе данных быстро находить нужные записи без полного сканирования всей таблицы, что существенно ускоряет выполнение запросов [23].

Одним из важнейших аспектов поиска вакансий является их название. Для оптимизации полнотекстового поиска по полю `title` будет создан инвертированный индекс с использованием механизма текстового поиска PostgreSQL. Это позволит эффективно находить вакансии, содержащие определенные ключевые слова в своем названии. Соответствующий SQL-код для создания данного индекса представлен в листинге 3.1.

```
CREATE INDEX idx_vacancy_title_fts ON vacancy USING GIN
(to_tsvector('russian', title));
```

Листинг 3.1 – Запрос создания индекса для названия

Не менее важным критерием поиска является местоположение вакансии. Для ускорения фильтрации вакансий по определенному городу или региону будет создан стандартный B-tree индекс по полю `location`: `CREATE INDEX idx_vacancy_location ON vacancy (location);`. Аналогично, для оптимизации поиска по типу занятости (полная, частичная, удаленная и т.д.), который также является распространенным фильтром, будет создан B-tree индекс по полю `work_type` с помощью запроса в листинге 3.2.

```
CREATE INDEX idx_vacancy_work_type ON vacancy (work_type);
```

Листинг 3.2 – Запрос создания индекса типа работы

Кроме того, часто соискатели комбинируют критерии поиска по уровню опыта и дате публикации вакансии. Для оптимизации таких составных запросов будет создан комбинированный B-tree индекс по полям `experience_level` и `created_at_millis`. Важно отметить, что сортировка по дате создания будет осуществляться в обратном порядке (DESC), чтобы наиболее свежие вакансии отображались первыми. Запрос приведен в листинге 3.3.

```
CREATE INDEX idx_vacancy_experience_created_at ON vacancy
(experience_level, created_at_millis DESC);
```

Листинг 3.3 – Запрос создания индекса по дате создания

Наконец, для быстрого извлечения всех вакансий, размещенных конкретным работодателем, что необходимо при отображении профиля компании, будет создан B-tree индекс по внешнему ключу `employer_id`, связывающему вакансию с таблицей компаний. Запрос создания данного индекса приведен в листинге 3.4.

```
CREATE INDEX idx_vacancy_employer_id ON vacancy
(employer_id);
```

Листинг 3.4 – Запрос создания индекса по идентификатору работодателя

При внедрении данной стратегии индексирования необходимо учитывать компромисс между ускорением операций чтения данных и потенциальным замедлением операций записи (добавление, обновление, удаление данных), так как при каждом изменении индексированных полей требуется обновление соответствующих индексов. Оптимальный набор индексов будет уточнен на этапе тестирования и эксплуатации платформы на основе анализа реальных запросов к базе данных.

3.2.2 Доступ к данным с помощью JOOQ

Для обеспечения эффективного и типобезопасного взаимодействия с базой данных PostgreSQL в серверной части нашего приложения, разработанной на языке Java с использованием фреймворка Ktor, мы выбрали библиотеку jOOQ (Java Object Oriented Querying). jOOQ представляет собой мощный инструмент, который позволяет писать SQL-запросы на языке Java, используя объектно-ориентированный подход. Одной из ключевых особенностей jOOQ является наличие codegen (генератора кода), который на основе существующей схемы базы данных (описанной в предыдущем разделе и представленной на рисунке 2.2) генерирует Java-классы, представляющие таблицы, поля, типы данных и связи [24]. Этот codegen обеспечивает ряд значительных преимуществ:

1 Типобезопасность. Все обращения к таблицам и полям базы данных осуществляются через сгенерированные Java-классы, что позволяет компилятору выявлять многие потенциальные ошибки на этапе разработки, предотвращая runtime-исключения, связанные с неправильными именами таблиц или типов данных.

2 IntelliSense и автодополнение. Использование сгенерированных классов в IDE обеспечивает интеллектуальное автодополнение кода, значительно ускоряя процесс написания запросов и снижая вероятность опечаток.

3 Удобство работы с SQL. jOOQ предоставляет богатый API, который позволяет строить сложные SQL-запросы декларативным и интуитивно понятным способом, минимизируя необходимость написания чистого SQL-кода в виде строк.

4 Производительность. jOOQ генерирует оптимальный SQL-код, который напрямую выполняется базой данных, обеспечивая высокую производительность доступа к данным.

Для иллюстрации использования JOOQ ниже рассмотрен пример запроса на получение списка вакансий с применением фильтрации. Для удобства доступа к данным с помощью JOOQ был реализован класс

JooqR2dbcContextFactory. Представленный в листинге 3.5 метод позволяет доставать данные используя пагинацию.

```
fun <T> fetchPageAndAwait(
    cursor: Cursor,
    mapper: RecordMapper<T>,
    table: Table<*>,
    fields: Array<Field<*>>,
    whereConditions: List<Condition> = emptyList(),
): Page<T> {
    val dslContext = DSL.using(connectionFactory)

    val select = dslContext
        .select(*fields)
        .from(table)
        .where(whereConditions)
        .offset(cursor.getOffset())
        .limit(cursor.getLimit())

    val data = Flux.from(select)
        .map { mapper(it) }
        .collectList()

    val totalCountRecord =
        Mono.from(
            dslContext
                .select(count().`as`("total_count"))
                .from(table)
                .where(whereConditions)
        ).block()
    val totalCount = totalCountRecord?.get("total_count", Int::class.java) ?: 0

    return Page(
        data = data.block(),
        pageInfo = PageInfo(
            totalPages = computeTotalPages(totalElements
= totalCount, pageSize = cursor.pageSize),
            pageSize = cursor.pageSize,
            pageNumber = cursor.pageNumber,
        )
    )
}
```

Листинг 3.5 – Запрос на получение данных с пагинацией

Используя сгенерированные jOOQ-классы, представляющие схему базы данных проекта, такой запрос на языке Java с использованием jOOQ и вспомогательного класса приведен в листинге 3.6.

```

jooqR2dbcContextFactory.fetchPageAndAwait(
    cursor = cursor,
    mapper = vacancyMapper,
    table = VACANCY,
    fields = vacancyFields,
    whereConditions = listOfNotNull(
        filters.currency?.let { VACANCY.SALARY_CUR-
RENCY.eq(it.name) },
        filters.workType?.let { VA-
CANCY.WORK_TYPE.eq(it.name) },
        filters.experience?.let { VACANCY.EXPERI-
ENCE_LEVEL.eq(it.name) },
        filters.salaryFrom?.let { VACANCY.SAL-
ARY_MAX.isNull.or(VACANCY.SALARY_MAX.ge(it.toLong())) },

        // todo: implement full text search
        filters.search?.let {
            VACANCY.DESCRPTION.likeIgnore-
Case("%$it%")
            .or(VACANCY.TITLE.likeIgnore-
Case("%$it%"))
        },
    )
)

```

Листинг 3.6 – Запрос создания индекса

Этот пример наглядно демонстрирует, как JOOQ с использованием сгенерированных классов позволяет строить типобезопасные, читаемые и производительные запросы к базе данных PostgreSQL на языке Java, значительно упрощая процесс доступа к данным в серверной части нашей платформы. Аналогичным образом могут быть реализованы и другие операции, такие как вставка, обновление и удаление данных.

Стоит также особо отметить, что для обеспечения неблокирующего и реактивного взаимодействия с базой данных PostgreSQL в рамках нашего backend-приложения, разработанного с использованием Kotlin и Ktor, был выбран R2DBC (Reactive Relational Database Connectivity) драйвер. В отличие от традиционных JDBC-драйверов, которые являются блокирующими и могут привести к неэффективному использованию ресурсов при большом количестве одновременных запросов, R2DBC предоставляет асинхронный API, основанный на реактивных потоках.

Использование R2DBC позволяет приложению обрабатывать множество параллельных запросов без блокировки потоков, что существенно повышает масштабируемость и отзывчивость серверной части. В контексте JOOQ, хотя сам JOOQ является синхронной библиотекой, существуют адаптеры и подходы для его интеграции с реактивными драйверами, такими как R2DBC, позволяя использовать преимущества типобезопасного построения запросов JOOQ в неблокирующем окружении. Это обеспечивает

оптимальное сочетание удобства разработки и высокой производительности при взаимодействии с базой данных PostgreSQL.

3.3 Разработка серверной части приложения

3.3.1 CQRS

Для организации серверной части нашего приложения, разработанной на Kotlin с использованием фреймворка Ktor, был выбран архитектурный паттерн CQRS (Command Query Responsibility Segregation). CQRS представляет собой шаблон проектирования, который разделяет операции по изменению состояния системы (команды) и операции по чтению состояния системы (запросы) на два отдельных интерфейса, его схема представлена на рисунке 3.2. В традиционных архитектурах часто используется одна и та же модель данных и один и тот же API для обеих этих целей. CQRS же предлагает их явное разделение, что позволяет оптимизировать каждую операцию независимо и решить ряд проблем, возникающих в сложных системах с высокой нагрузкой [25].

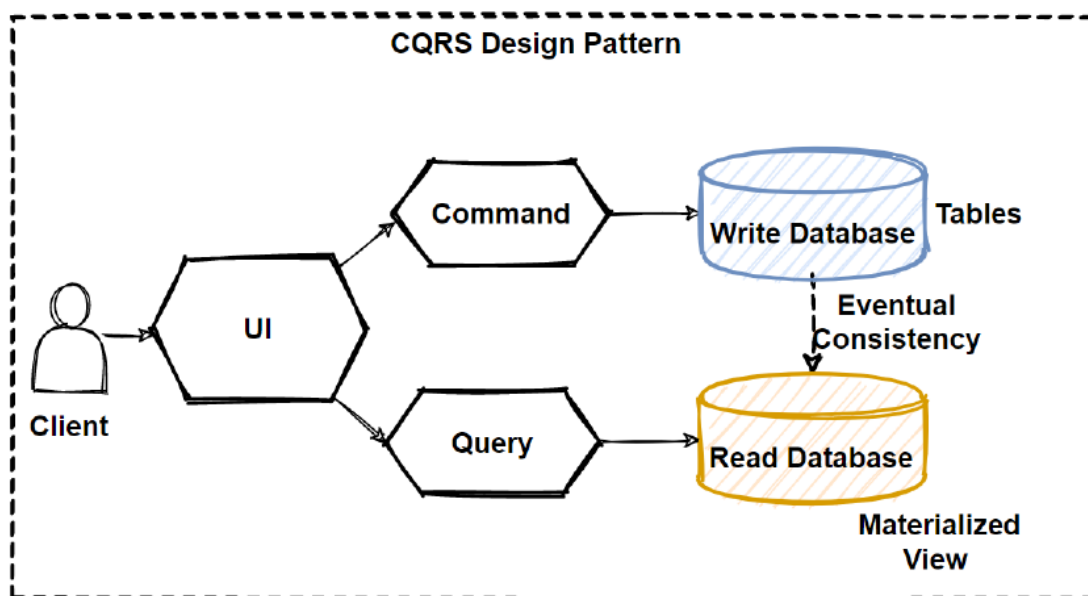


Рисунок 3.2 – Схема CQRS паттерна

Основные принципы паттерна CQRS:

1 Разделение команд и запросов. Существуют отдельные модели и интерфейсы для обработки команд (операций, которые изменяют состояние системы, например, создание вакансии, отклик на вакансию) и запросов (операций, которые извлекают информацию из системы, например, получение списка вакансий, просмотр профиля пользователя).

2 Раздельные модели данных (опционально). Хотя это не является строгим требованием CQRS, часто для команд и запросов используются различные модели данных. Модель для записи может быть оптимизирована для операций изменения состояния, а модель для чтения может быть

денормализована для обеспечения высокой производительности при выборке данных.

3 Раздельные хранилища данных (опционально). В более сложных реализациях CQRS могут использоваться различные хранилища данных, оптимизированные для записи и чтения. Например, для записи может использоваться реляционная база данных с обеспечением ACID-транзакций, а для чтения – NoSQL база данных или специализированное хранилище для быстрого поиска и агрегации. В нашем проекте на начальном этапе мы можем использовать одну и ту же базу данных PostgreSQL, но с оптимизированными моделями и запросами для чтения и записи.

Преимущества применения CQRS в разрабатываемом проекте:

1 Оптимизация производительности. Разделение операций чтения и записи позволяет оптимизировать модели данных и механизмы хранения для каждой цели. Запросы на чтение, которые обычно составляют большую часть нагрузки в приложениях с интенсивным использованием данных, могут быть оптимизированы для скорости выборки, возможно, за счет денормализации данных. Операции записи могут быть сфокусированы на обеспечении целостности и надежности.

2 Масштабируемость. Раздельные модели чтения и записи могут масштабироваться независимо друг от друга в зависимости от нагрузки. Если количество запросов на чтение значительно превышает количество операций записи, мы можем масштабировать подсистему чтения независимо.

3 Гибкость и сложность модели. CQRS позволяет использовать различные модели данных, лучше соответствующие требованиям операций чтения и записи. Модель записи может быть более нормализованной и отражать предметную область, в то время как модель чтения может быть адаптирована для конкретных сценариев отображения данных, что упрощает сложные запросы на чтение.

4 Разделение ответственности. CQRS способствует более четкому разделению ответственности между различными частями кодовой базы. Обработчики команд отвечают за изменение состояния, а обработчики запросов – за извлечение информации. Это упрощает разработку, тестирование и поддержку.

В серверной части будет реализован CQRS следующим образом:

1 Команды (Commands). Представляют собой намерения пользователя изменить состояние системы (например, CreateVacancyCommand, ApplyToVacancyCommand). Команды будут обрабатываться специальными обработчиками команд (Command Handlers), которые будут выполнять необходимую бизнес-логику и изменять состояние системы, взаимодействуя с моделью записи и уровнем доступа к данным (с использованием JOOQ и R2DBC).

1 Запросы (Queries). Представляют собой запросы пользователя на получение информации из системы. Запросы будут обрабатываться обработчиками запросов (Query Handlers), которые будут извлекать данные из модели чтения (которая может быть той же или оптимизированной версией

модели записи) с использованием JOOQ и R2DBC и возвращать их клиенту. В каждой команде и запросе указывается их идентификатор, а также можно указать, пользователи с какой ролью могут их вызывать. Команда для получения вакансий с пагинацией приведена в листинге 3.7.

```
class GetVacanciesWithCursorQuery(  
    val filters: GetVacanciesFilters,  
    override val cursor: Cursor,  
    ) : AbstractCommand<Page<VacancyDto>>(), WithCursor {  
  
    companion object : UriAware, WithRoles {  
        override fun uri(): String = "com.job.vacancy.get_vacancies_with_cursor"  
  
        override fun roles(): Set<UserRole> = EnumSet.of(UserRole.ANONYMOUS, UserRole.USER, UserRole.EMPLOYER)  
    }  
}
```

Листинг 3.7 – Команда на получение вакансий с пагинацией

В качестве примера можно рассмотреть сценарий просмотра списка вакансий. Клиентское приложение отправляет запрос (Query) `GetVacancyListQuery` с определенными параметрами фильтрации (местоположение, ключевые слова и т.д.). Этот запрос поступает на шину запросов, которая направляет его к обработчику `GetVacancyListQueryHandler`. Обработчик выполняет необходимый запрос к базе данных (используя оптимизированную модель чтения и JOOQ), получает список вакансий и возвращает его клиенту.

При создании новой вакансии клиентское приложение отправляет команду `CreateVacancyCommand` с данными новой вакансии. Эта команда поступает на шину команд, которая направляет ее к обработчику `CreateVacancyCommandHandler`. Обработчик выполняет валидацию данных, сохраняет новую вакансию в базе данных (используя модель записи и JOOQ) и может инициировать другие действия (например, отправку уведомлений). Команда для создания вакансии приведена в листинге 3.8.

```
data class CreateVacancyCommand(  
    val title: String,  
    val salaryMin: Money?,  
    val salaryMax: Money?,  
    val workType: WorkType?,  
    val location: String?,  
    val description: String?,  
    val experienceLevel: ExperienceLevel?,  
    val createdAtMillis: Long,  
    val editedAtMillis: Long,  
    ) : UnitCommand() {
```

```

companion object : UriAware, WithRoles {
    override fun uri(): String = "com.job.vacancy.create"

    override fun roles(): Set<UserRole> = Enum-
Set.of(UserRole.EMPLOYER, UserRole.USER)
}
}

```

Листинг 3.8 – Команда для создания вакансии

Таким образом, применение паттерна CQRS позволит создать более масштабируемую, производительную и гибкую серверную часть нашего приложения, лучше соответствующую требованиям сложной системы с большим объемом операций чтения и записи.

3.3.2 Цепочка ответственности

Для обеспечения надежной обработки входящих HTTP-запросов к серверной части нашего приложения, разработанной на Kotlin с использованием фреймворка Ktor, был реализован паттерн Chain of Responsibility в рамках механизма middleware. Схема паттерна представлена на рисунке 3.3. Паттерн Chain of Responsibility позволяет организовать последовательность таких middleware-компонентов, где каждый компонент (handler в цепочке) выполняет определенную задачу и затем передает управление следующему компоненту в цепочке, либо завершает обработку запроса [26].

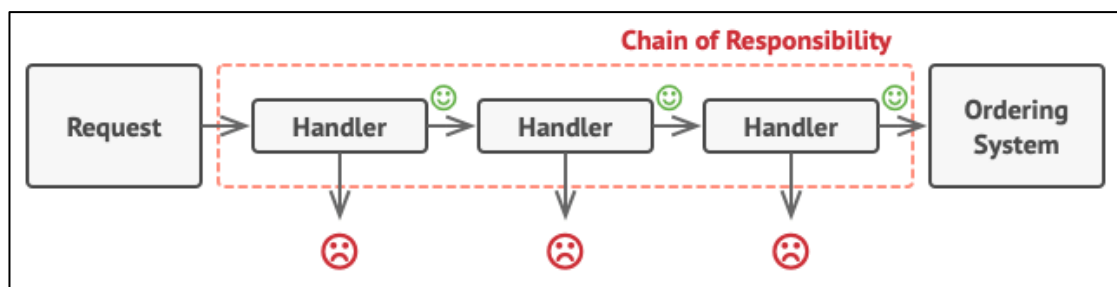


Рисунок 3.3 – Схема паттерна цепочка ответственности

Применение паттерна Chain of Responsibility в middleware обеспечивает следующие ключевые преимущества для нашего проекта:

Разделение ответственности: Каждый middleware-компонент отвечает за выполнение строго определенной задачи, такой как аутентификация, авторизация, логирование, обработка ошибок, CORS и т.д. Это способствует чистоте кода, упрощает тестирование и поддержку отдельных аспектов обработки запросов.

1 Гибкость и расширяемость. Добавление новой функциональности обработки запросов не требует изменения существующих компонентов. Достаточно создать новый middleware-компонент и включить его в цепочку в

нужном порядке. Это обеспечивает высокую гибкость и упрощает расширение функциональности приложения в будущем.

2 Конфигурируемость. Порядок следования middleware-компонентов в цепочке определяет порядок их выполнения. Это позволяет легко настраивать процесс обработки запросов в зависимости от требований безопасности, логирования и других аспектов. Например, логирование может выполняться до или после аутентификации, в зависимости от того, какие данные необходимо зафиксировать.

3 Уменьшение связности. Каждый middleware-компонент слабо связан со следующим в цепочке. Он знает только об интерфейсе следующего обработчика, что снижает зависимость между компонентами и упрощает их независимую разработку и тестирование.

В проекте паттерн Chain of Responsibility в middleware используется для организации таких критически важных аспектов обработки запросов, как безопасность и логирование.

Для обеспечения безопасности доступа к ресурсам нашего backend-приложения, разработанного на Kotlin с использованием фреймворка Ktor, мы применяем паттерн Chain of Responsibility в рамках механизма middleware, объединяя функциональность аутентификации и авторизации в единый компонент Auth Middleware.

Когда на сервер поступает HTTP-запрос, первым шагом в цепочке middleware является Auth Middleware. Его основной задачей является установление подлинности пользователя (аутентификация) и проверка его прав на выполнение запрошенного действия (авторизация).

Процесс аутентификации в Auth Middleware происходит следующим образом. На этом этапе middleware проверяет наличие в запросе учетных данных пользователя. Это может включать анализ токенов.

Проверка наличия и валидности токенов, таких как JWT (JSON Web Tokens), которые могут передаваться в заголовках запроса (например, Authorization: Bearer <token>) или в cookie. Middleware проверяет подпись токена, срок его действия и другие валидационные параметры.

В случае успешной аутентификации, информация об идентифицированном пользователе (например, его уникальный идентификатор и присвоенные роли) сохраняется в контексте запроса. Эта информация становится доступной для последующих обработчиков запросов, которым не нужно повторно аутентифицировать пользователя.

После успешной аутентификации, Auth Middleware переходит к этапу авторизации. На этом этапе определяется, имеет ли аутентифицированный пользователь необходимые права доступа для выполнения запрошенного ресурса и HTTP-метода. Процесс авторизации может основываться на следующих факторах:

Если Auth Middleware определяет, что аутентифицированный пользователь обладает достаточными правами доступа, он передает управление следующему middleware-компоненту в цепочке (если такой есть) или конечному обработчику запроса.

В случае, если аутентификация не удалась (например, неверный токен или отсутствие учетных данных) или авторизация была отклонена (пользователь не имеет прав на запрашиваемый ресурс), Auth Middleware немедленно прерывает дальнейшую обработку запроса. Клиенту возвращается HTTP-ответ с соответствующим кодом состояния ошибки (например, 401 Unauthorized для проблем с аутентификацией, 403 Forbidden для проблем с авторизацией) и, возможно, дополнительным сообщением, объясняющим причину отказа.

Для обеспечения полного аудита работы приложения и облегчения отладки, в цепочку middleware включены компоненты, отвечающие за логирование:

1 Запрос-логирование. Этот компонент регистрирует информацию о каждом входящем HTTP-запросе, такую как метод запроса, URL, заголовки, IP-адрес клиента и время получения запроса.

2 Ответ-логирование. Этот компонент регистрирует информацию об исходящем HTTP-ответе, такую как код состояния (200 OK, 404 Not Found и т.д.), заголовки ответа и время отправки ответа.

Когда на сервер поступает HTTP-запрос, он проходит через сконфигурированную цепочку middleware. Каждый компонент в цепочке выполняет свою задачу (например, проверяет токен аутентификации, записывает информацию о запросе в лог) и затем вызывает следующий компонент в цепочке. Если какой-либо middleware-компонент решает, что дальнейшая обработка запроса невозможна (например, из-за неудачной аутентификации или отсутствия прав доступа), он может прервать цепочку и вернуть клиенту соответствующий ответ. В противном случае, после прохождения всех middleware, запрос достигает конечного обработчика, который выполняет основную бизнес-логику и формирует ответ. Ответ также может проходить через обратную цепочку middleware для обработки (например, добавление заголовков безопасности, логирование ответа).

Таким образом, применение паттерна Chain of Responsibility в middleware Ktor позволяет элегантно и модульно организовать важные аспекты обработки запросов, такие как безопасность и логирование, обеспечивая гибкость, расширяемость и поддерживаемость нашего backend-приложения.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения - это метод проверки соответствия фактического программного продукта ожидаемым требованиям, который также необходим, чтобы убедиться, что продукт не содержит дефектов. Подразумевает выполнение предварительно определенных алгоритмов с использованием ручных или автоматизированных инструментов для оценки одного или нескольких интересующих свойств. Целью тестирования ПО является выявление ошибок, пробелов или отсутствующих требований, заданных на этапе проектирования продукта [27].

Для обеспечения комплексного подхода к тестированию разработанного программного средства, направленного на достижение высокого уровня качества и надежности, применяется пирамида тестирования. Эта концепция, визуально представленная на рисунке 4.1, определяет оптимальное соотношение различных видов тестов на разных уровнях разработки.

Основание пирамиды составляют модульные тесты, которые являются самыми многочисленными и охватывают тестирование отдельных, наименьших компонентов приложения (функций, классов, методов).

На среднем уровне располагаются интеграционные тесты, которые проверяют взаимодействие между различными модулями и подсистемами приложения.

Вершину пирамиды занимают сквозные тесты или UI-тесты, которые имитируют поведение пользователя и проверяют работу приложения в целом, включая взаимодействие с пользовательским интерфейсом и различными компонентами инфраструктуры.

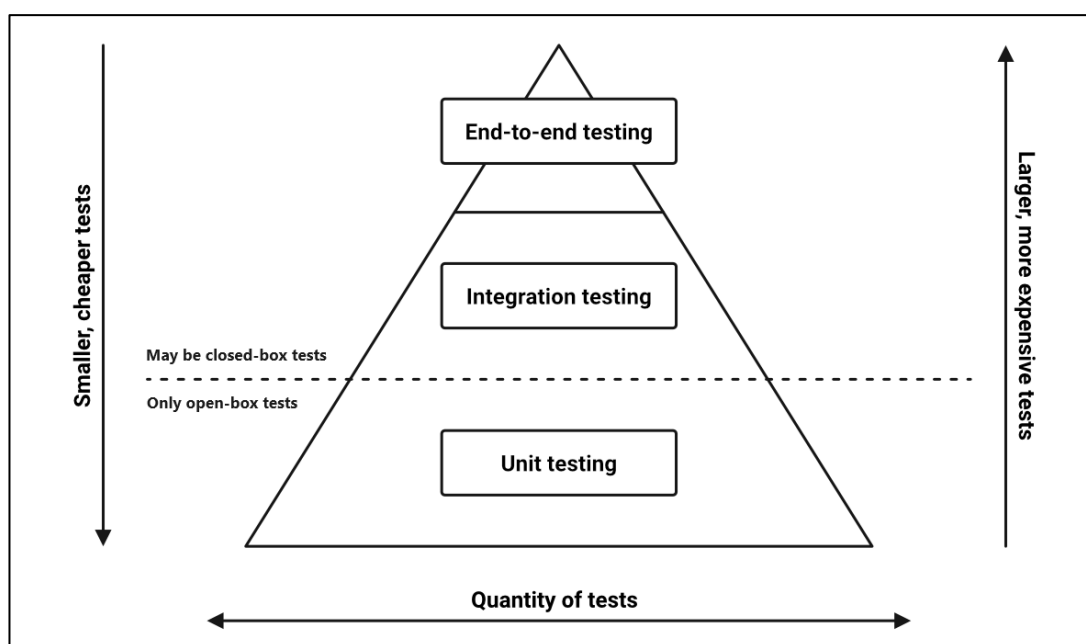


Рисунок 4.1 – Пирамида тестирования

Для обеспечения надежной и корректной работы каждого программного модуля, а также для проверки их слаженной интеграции и соответствия заявленным функциональным требованиям, в процессе разработки был применен комплексный подход к тестированию, включающий разработку и выполнение модульных и интеграционных тестов.

Модульные тесты были разработаны для тщательной проверки изолированной работы отдельных компонентов приложения. Целью данного вида тестирования является подтверждение правильности реализации логики каждой отдельной единицы кода и ее соответствия спецификациям. В случаях, когда полное тестирование всех возможных входных данных модуля нецелесообразно, проверка осуществлялась на основе тщательно подобранной выборки репрезентативных значений.

В дополнение к модульному тестированию, для проверки взаимодействия между различными модулями и подсистемами приложения, а также для подтверждения соответствия разработанного программного средства требованиям, сформулированным на начальных этапах проектирования, были разработаны интеграционные тесты. Успешное прохождение этих тестов должно подтвердить корректность взаимодействия ключевых компонентов системы и обеспечить ее работоспособность в основных сценариях использования.

Результаты проведенного тестирования разработанного программного средства представлены в подробных таблицах 4.1 – 4.13. Эти таблицы содержат описание тестовых сценариев, ожидаемые результаты и фактические результаты выполнения тестов для различных функциональных модулей платформы. Анализ представленных результатов позволяет провести всестороннюю оценку текущего уровня качества разработанного программного обеспечения и определить области, требующие дополнительной доработки или исправления выявленных недочетов.

Таблица 4.1 – Тест-кейс неудачной аутентификации с неверным логином

Тест	Ожидаемый результат	Результат
<p>Попытка аутентификации с несуществующими данными</p> <p>1. Воспользоваться командой с неверным логином «test123123@gmail.com»</p>	<p>Унифицированное сообщение об ошибке "Неверное имя пользователя или пароль", не указывая, что именно введено неверно: username или password.</p>	<p>Успех</p>

Таблица 4.2 – Тест-кейс неудачной аутентификации с неверным паролем

Тест	Ожидаемый результат	Результат
Попытка аутентификации с несуществующими данными Воспользоваться командой с неверным паролем «invalidPassword»	Унифицированное сообщение об ошибке "Неверное имя пользователя или пароль", не указывая, что именно введено неверно: username или password.	Успех

Таблица 4.3 – Тест-кейс удачной аутентификации

Тест	Ожидаемый результат	Результат
Успешная аутентификация с валидными данными Воспользоваться командой с корректными логином «email@gmail.com» и паролем «password»	Система предоставляет доступ, возвращая токен аутентификации или перенаправляя на защищенную страницу	Успех

Таблица 4.4 – Тест-кейс неудачной регистрации

Тест	Ожидаемый результат	Результат
Попытка регистрации с уже занятым email Использовать команду для регистрации с существующим логином «email@gmail.com» при регистрации	Система возвращает ошибку "Этот email уже зарегистрирован" и не создает нового пользователя. Проверяется, что в базе данных остается только одна запись с этим email.	Успех

Таблица 4.5 – Тест-кейс неудачного отклика на вакансию

Тест	Ожидаемый результат	Результат
Попытка отклика на вакансию от лица работодателя 1. Авторизоваться как работодатель 2. Попытаться отправить отклик на свою вакансию	Система возвращает ошибку "Работодатель не может откликаться на вакансии" и не создает отклик при регистрации	Успех

Таблица 4.6 – Тест-кейс успешного отклика на вакансию

Тест	Ожидаемый результат	Результат
Успешный отклик на вакансию соискателем 1. Авторизоваться как соискатель (user) 2. Отправить отклик на активную вакансию	Система возвращает ошибку "Этот email уже зарегистрирован" и не создает нового пользователя. Проверяется, что в базе данных остается только одна запись с этим email.	Успех

Таблица 4.7 – Тест-кейс успешного отклика на вакансию

Тест	Ожидаемый результат	Результат
Повторный отклик на вакансию 1. Использовать команду: 2. Пользователь, который уже откликался на вакансию, пытается отправить отклик на эту же вакансию еще раз	Система должна вернуть ошибку, например, "Вы уже откликнулись на эту вакансию" или "Невозможно отправить повторный отклик на данную вакансию". Новый отклик не должен быть создан и в базе данных должна остаться только одна запись об отклике этого пользователя на данную вакансию.	Успех

Таблица 4.8 – Тест-кейс применение фильтра по одной категории

Тест	Ожидаемый результат	Результат
Применить фильтр по опыту "Опыт работы: 1-3 года" к списку вакансий	Система возвращает только те вакансии/резюме, которые соответствуют выбранному фильтру. Проверяется, что все отображаемые элементы содержат указанное значение фильтра..	Успех

Таблица 4.9 – Тест-кейс применение нескольких фильтров одновременно

Тест	Ожидаемый результат	Результат
Применить несколько фильтров одновременно к списку вакансий/резюме.	Система отображает только те вакансии/резюме, которые соответствуют всем выбранным фильтрам. Проверяется, что все отображаемые элементы содержат все указанные значения фильтров.	Успех

Таблица 4.10 – Тест-кейс отсутствие результатов при применении фильтров

Тест	Ожидаемый результат	Результат
Применить комбинацию фильтров, по которой нет подходящих вакансий/резюме (например, "Город: Магадан" И "Зарплата: от 500 000 руб." И "Опыт работы: более 10 лет").	Система возвращает пустой список вакансий, исключение не выбрасывается.	Успех

Таблица 4.11 – Тест-кейс просмотр резюме другого пользователя

Тест	Ожидаемый результат	Результат
Авторизованный пользователь А пытается получить доступ к резюме, принадлежащему пользователю В, используя прямой ID резюме пользователя В или другой метод, который не предусматривает общего доступа.	Система возвращает ошибку "Доступ запрещен", "У вас нет прав для просмотра данного резюме" или аналогичное сообщение. Информация из резюме пользователя В не должна быть доступна пользователю А.	Успех

Таблица 4.12 – Тест-кейс отсутствие результатов при применении фильтров

Тест	Ожидаемый результат	Результат
Пользователь, обладающий ролью "Администратор компании" или "Владелец компании" (т.е., имеющий соответствующие права на управление составом сотрудников своей организации), инициирует процесс добавления нового пользователя с ролью "Работодатель" в свою компанию.	Система возвращает ошибку "Этот пользователь уже является работодателем в другой компании", "Невозможно добавить пользователя, который уже связан с другой компанией" или аналогичное сообщение. Пользователь-работодатель не должен быть добавлен в новую компанию, и его привязка к предыдущей компании должна сохраниться.	Успех

Таблица 4.13 – Тест-кейс просмотр своего резюме

Тест	Ожидаемый результат	Результат
Владелец скрытого резюме (авторизованный пользователь) запрашивает просмотр своего собственного скрытого резюме.	Система успешно отображает все детали скрытого резюме владельцу.	Успех

Для обеспечения надежной и корректной работы каждого программного модуля, а также для проверки их слаженной интеграции и соответствия заявленным функциональным требованиям, в процессе разработки был применен комплексный подход к тестированию. Этот подход включал разработку и выполнение как модульных, так и интеграционных тестов.

Модульные тесты были разработаны для тщательной проверки изолированной работы отдельных компонентов приложения. Их основной целью являлось подтверждение правильности реализации логики каждой отдельной единицы кода и ее соответствия спецификациям. В тех случаях, когда полное тестирование всех возможных входных данных модуля было нецелесообразным, проверка осуществлялась на основе тщательно подобранной выборки репрезентативных значений.

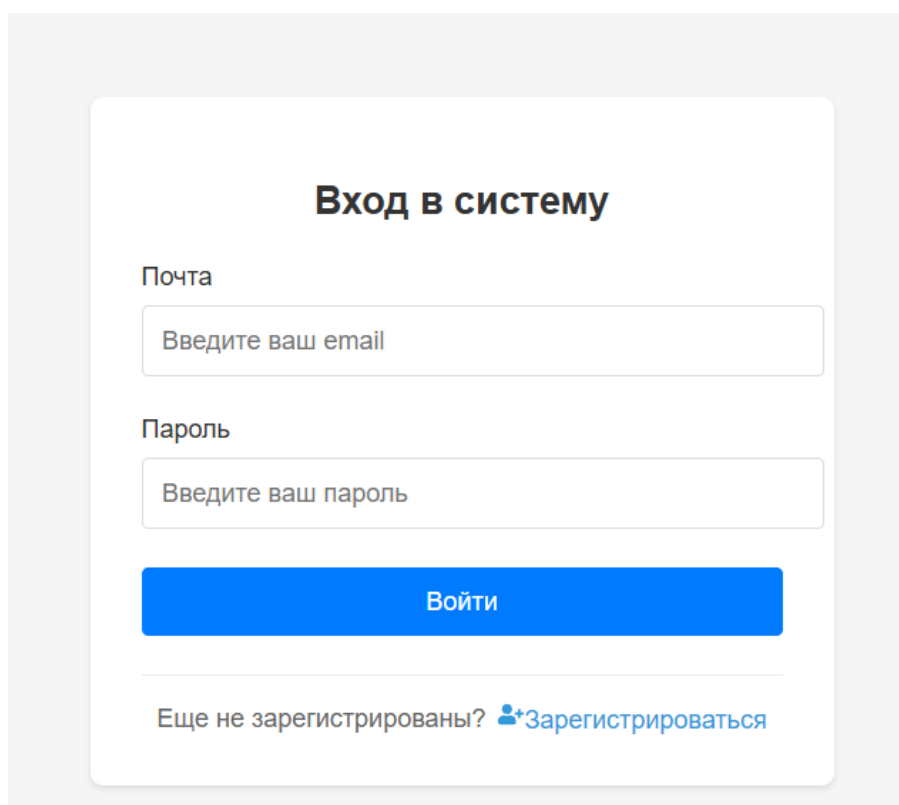
В дополнение к модульному тестированию, были разработаны интеграционные тесты. Они предназначались для проверки взаимодействия между различными модулями и подсистемами приложения, а также для подтверждения соответствия разработанного программного средства требованиям, сформулированным на начальных этапах проектирования. Успешное прохождение этих тестов подтвердило корректность взаимодействия ключевых компонентов системы и обеспечило ее работоспособность в основных сценариях использования.

Результаты проведенного тестирования разработанного программного средства подробно представлены в таблицах 4.1 – 4.13. Эти таблицы содержат описание тестовых сценариев, ожидаемые результаты и фактические результаты выполнения тестов для различных функциональных модулей платформы. Анализ представленных результатов позволил провести всестороннюю оценку текущего уровня качества разработанного программного обеспечения и определить области, требующие дополнительной доработки или исправления выявленных недочетов.

5 МЕТОДИКА ИСПОЛЬЗОВАНИЯ РАЗРАБОТАННОГО ПРОГРАММНОГО СРЕДСТВА

Разработанная платформа поиска работы для молодых специалистов представляет собой комплексное и многофункциональное веб-приложение, спроектированное для обеспечения эффективного взаимодействия между соискателями, работодателями и администрацией системы. Удобство доступа к платформе реализовано через стандартный веб-браузер, что позволяет пользователям взаимодействовать с системой с различных устройств (персональных компьютеров, планшетов, смартфонов) без необходимости установки какого-либо дополнительного программного обеспечения.

На рисунке 5.1 представлена страница входа в систему, являющаяся отправной точкой для авторизованных пользователей. Данная страница содержит стандартные поля для ввода логина (адрес электронной почты) и пароля, а также кнопку "Войти" для осуществления аутентификации. Кроме того, для новых пользователей предусмотрена кнопка "Регистрация", перенаправляющая на форму создания новой учетной записи.



The image shows a login form titled "Вход в систему" (Login to the system). It contains two input fields: "Почта" (Email) with the placeholder text "Введите ваш email" and "Пароль" (Password) with the placeholder text "Введите ваш пароль". Below these fields is a blue button labeled "Войти" (Login). At the bottom, there is a link that says "Еще не зарегистрированы? *Зарегистрироваться" (Not yet registered? *Register).

Рисунок 5.1 – Страница входа в систему

Однако, важной особенностью разработанной платформы является предоставление определенного уровня доступа для анонимных пользователей, то есть лиц, не прошедших процедуру аутентификации. Как видно на рисунках 5.2 и 5.3, даже без ввода учетных данных, анонимные пользователи имеют возможность просматривать доступные вакансии и знакомиться с их

детальным описанием. Это позволяет потенциальным соискателям оценить спектр предложений и принять решение о необходимости регистрации для получения полного доступа к функционалу платформы, такому как создание резюме, отклик на вакансии и сохранение избранных предложений.

Job First [Вакансии](#) [Войти](#) [Зарегистрироваться](#)

Вакансии

Поиск по вакансиям...

Зарплата
От BYN

Опыт
Любой

Формат работы
Любой

[Найти вакансии](#)

Junior Developer Position

Зарплата не указана

Гибридный

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has bee...

Без опыта Просмотров: 0

[Откликнуться](#)

Data Scientist

250,000 - 400,000 BYN

В офисе

Анализ данных, машинное обучение. Опыт работы с Python и библиотеками для анализа данных.

3 года Просмотров: 0

[Откликнуться](#)

Backend Developer (Python)

150,000 - 250,000 BYN

Удаленно

Разработка серверной части на Python/Django. Опыт с PostgreSQL.

Без опыта Просмотров: 0

[Откликнуться](#)

Рисунок 5.2 – Страница с вакансиям от лица анонимного пользователя

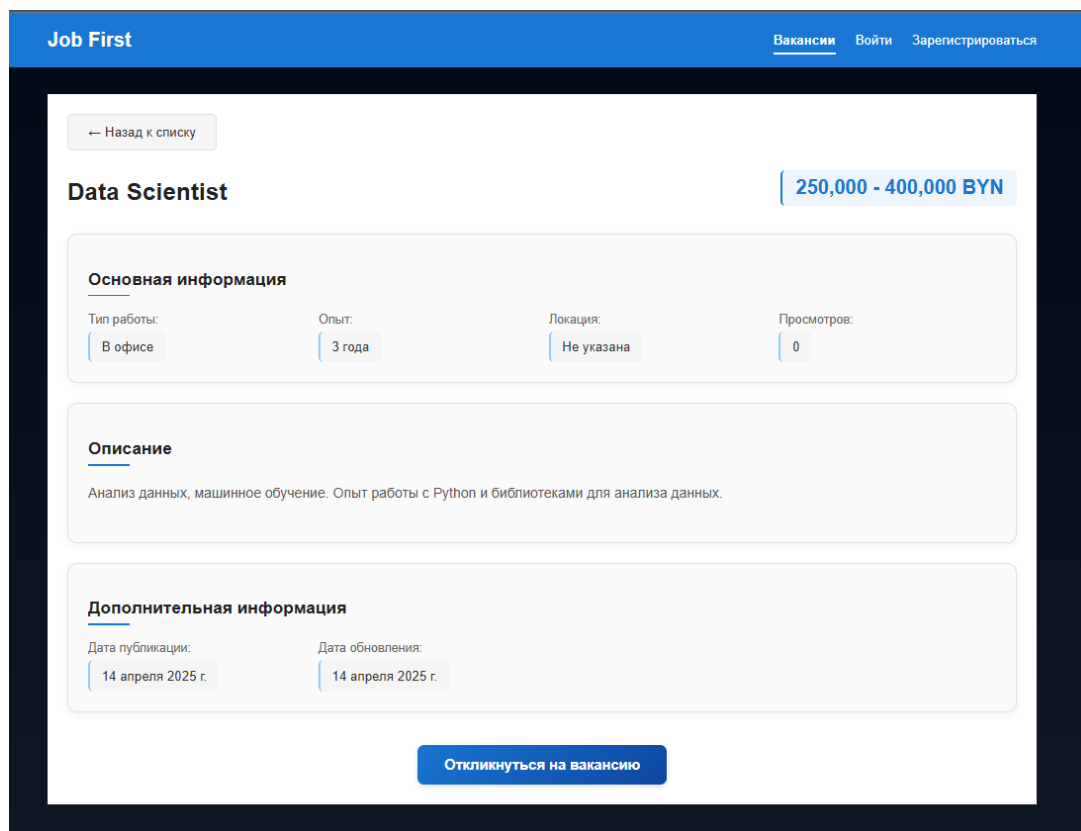


Рисунок 5.3 – Страница с деталями вакансии

Для обеспечения полноценного взаимодействия с платформой, такого как отклик на заинтересовавшую вакансию, требуется идентификация пользователя. В связи с этим, при нажатии кнопки "Откликнуться" анонимным пользователем, система автоматически перенаправляет его на страницу входа в систему. Это необходимо для того, чтобы пользователь мог авторизоваться, используя свои учетные данные, или пройти процедуру регистрации, если он еще не является участником платформы. После успешной аутентификации пользователь будет перенаправлен обратно к выбранной вакансии и сможет выполнить желаемое действие – отправить свой отклик.

После успешной аутентификации, основанной на введенных логине и пароле, система идентифицирует пользователя и предоставляет ему доступ к функционалу, соответствующему его роли (соискатель или работодатель), которая присваивается при регистрации. Сразу после успешного входа в систему, пользователь автоматически перенаправляется на страницу своего личного профиля, где представлена основная информация, указанная им при регистрации, а также доступ к ключевым функциям платформы, соответствующим его роли. Страница с информацией о пользователе представлена на рисунке 5.4.

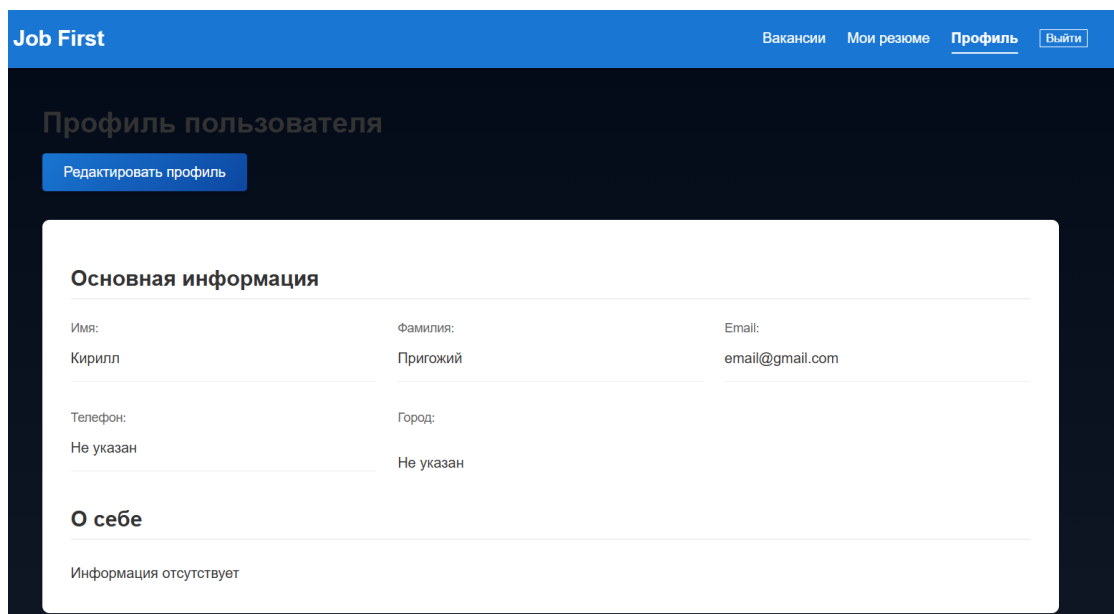


Рисунок 5.4 – Страница с информацией о пользователе

При успешной аутентификации в качестве соискателя, пользователь получает персонализированный доступ к функционалу платформы. Одним из ключевых элементов навигационной панели становится кнопка "Мои резюме". Переход по этой кнопке открывает страницу, представленную на рисунке 5.5, где соискатель может увидеть список всех созданных им резюме.

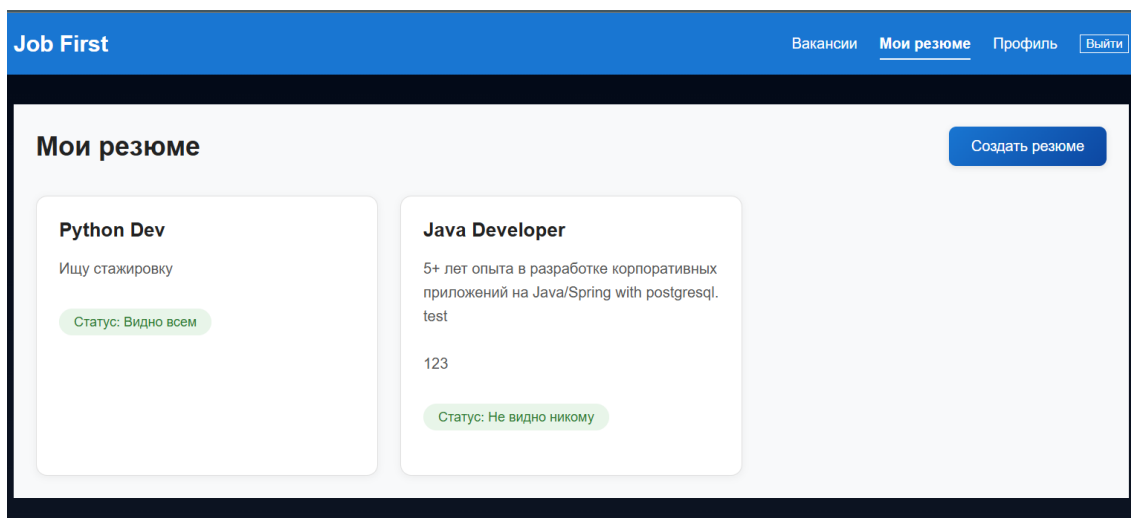


Рисунок 5.5 – Страница с резюме пользователя

Данная страница предоставляет полный набор инструментов для управления резюме. Соискатель имеет возможность создавать новые резюме, используя интуитивно понятный конструктор, редактировать ранее созданные резюме, внося необходимые обновления в информацию об опыте работы, образовании, навыках и других разделах. Также предусмотрена функция удаления неактуальных или более не нужных резюме из личного кабинета,

обеспечивая актуальность представленной информации для потенциальных работодателей.

При нажатии на кнопку "Создать резюме" на странице "Мои резюме", перед пользователем появляется всплывающее модальное окно показанное на рисунке 5.6. Это окно предоставляет соискателю удобный интерфейс для ввода основных деталей нового резюме. В частности, предусмотрены поля для указания названия резюме, которое поможет идентифицировать его в списке, а также для добавления краткого описани, дающего общее представление о квалификации и карьерных целях. Кроме того, в модальном окне реализована возможность добавления информации об опыте работы в структурированном формате. Важным элементом является настройка видимости резюме, позволяющая соискателю определить, будет ли данное резюме доступно для просмотра всем зарегистрированным работодателям или останется скрытым до момента отклика на конкретную вакансию.

Рисунок 5.6 – Модальное окно для создания резюме

После того как соискатель успешно отправляет свой отклик на выбранную вакансию, система незамедлительно информирует его об этом действии посредством всплывающего уведомления как показано на рисунке

5.7. Данное уведомление подтверждает, что заявка соискателя была принята и отправлена работодателю, и может содержать дополнительную информацию, например, дату и время отправки отклика, а также название вакансии, на которую был произведен отклик. Карточка вакансии при этом меняет текст кнопки на соответствующий, как показано на рисунке 5.8. Это обеспечивает обратную связь с пользователем и подтверждает успешное выполнение его действия.

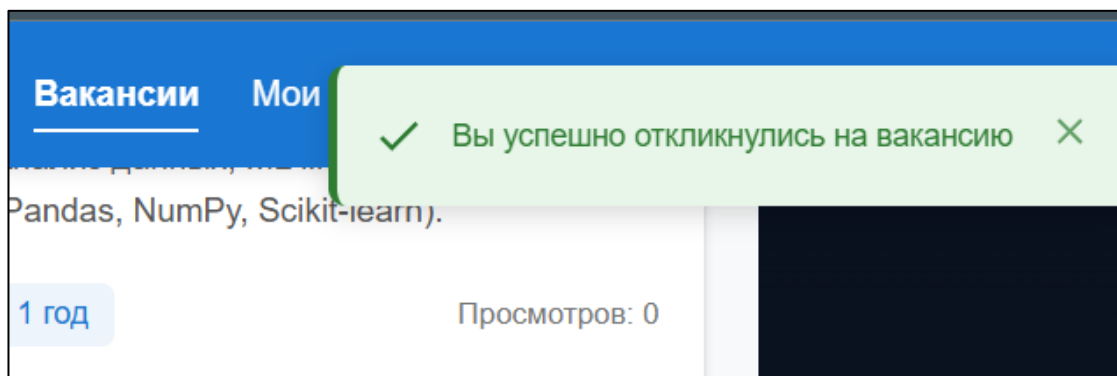


Рисунок 5.7– Уведомление об успешном отклике на вакансию

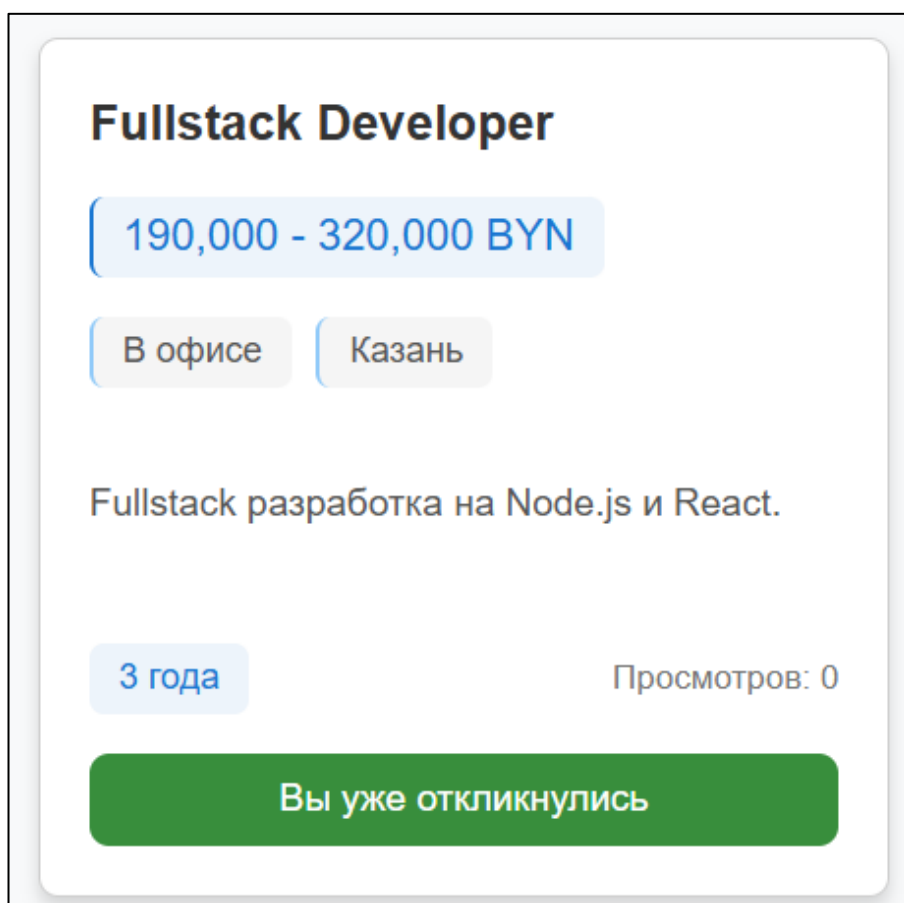


Рисунок 5.8 – Карточка с вакансией, на которую пользователь уже откликнулся

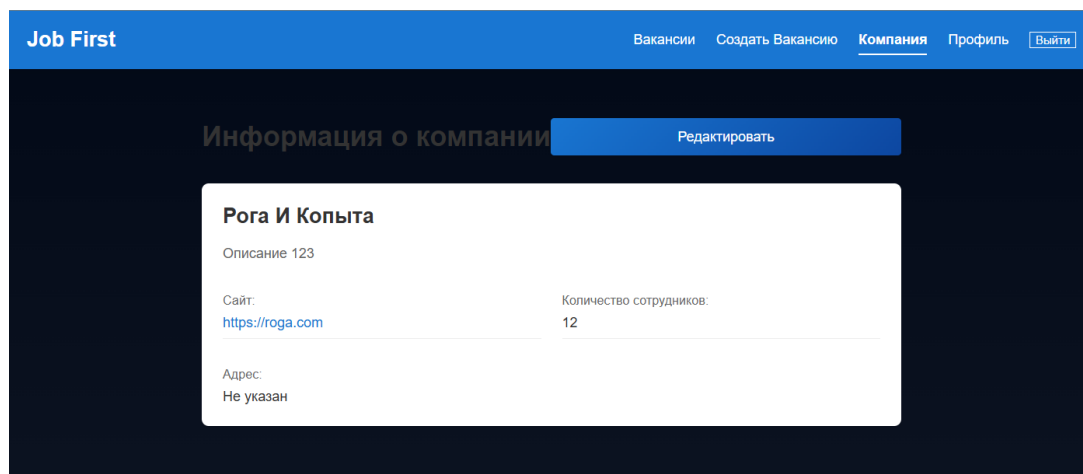


Рисунок 5.9 – Информация о компании

Соискатели получают доступ к инструментам для создания и управления резюме, включая возможность указания опыта работы, образования, ключевых навыков и другой релевантной информации. Реализован удобный конструктор резюме с возможностью предварительного просмотра и сохранения нескольких версий резюме. Для поиска работы соискателям предоставляется функциональность просмотра списка вакансий с возможностью фильтрации по различным критериям, таким как зарплата, опыт работы, ключевые слова (тэги) и формат работы. Найдя подходящую вакансию, соискатель может откликнуться на нее, прикрепив одно из своих резюме и сопроводительное письмо. Система также предусматривает механизм уведомлений для информирования соискателей о новых подходящих вакансиях и изменениях статуса их откликов.

Работодателям предоставляются инструменты для создания и управления вакансиями, включая подробное описание требований к кандидатам, условий работы и контактной информации. Для поиска подходящих кандидатов работодатели могут просматривать резюме соискателей, используя различные фильтры. Встроенная система обмена сообщениями обеспечивает прямую коммуникацию между работодателями и заинтересованными соискателями. Администраторы платформы обладают полным контролем над системой, включая управление пользователями (блокировка/разблокировка учетных записей), модерацию контента (вакансий и профилей), а также верификацию работодателей для обеспечения достоверности информации.

Таким образом, разработанная платформа представляет собой комплексное решение, охватывающее все ключевые аспекты процесса поиска работы и найма персонала, обеспечивая удобное и эффективное взаимодействие для всех участников.

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО СРЕДСТВА

6.1 Характеристика разрабатываемого продукта

Разрабатываемое программное средство представляет собой веб-платформ, предназначенную для помощи молодым специалистам в поиске работы. Основная цель приложения — упростить процесс трудоустройства, предоставляя пользователям удобный инструмент для поиска вакансий, взаимодействия с работодателями и повышения профессиональной квалификации.

Продукт обладает следующими возможностями:

1 Добавление резюме в различных форматах на веб-сайт. Пользователи могут загружать готовые резюме в формате PDF, DOCX и других популярных типах файлов.

2 Автоматизация откликов. Возможность автоматической подачи заявок на вакансии по заданным параметрам.

3 Поиск сотрудников для компаний. Работодатели могут находить кандидатов по ключевым навыкам, опыту и другим параметрам.

4 Чат и система уведомлений. Оперативное общение с работодателями и уведомления о статусе заявок.

5 Отзывы о работодателях. Соискатели могут оставлять и просматривать отзывы о компаниях, что помогает оценить условия работы.

6 Фильтрация вакансий. Гибкие фильтры по уровню заработной платы, графику работы, опыту и другим критериям.

7 Отслеживание статуса заявок. Пользователи могут видеть, на каком этапе находится их отклик, будь то рассмотрение, приглашение на собеседование или отказ.

Целевой аудиторией разрабатываемого продукта являются:

- выпускники вузов и колледжей, находящиеся в поиске первой работы;
- студенты, желающие пройти стажировку или получить первую практическую работу;
- молодые специалисты с небольшим опытом работы, заинтересованные в развитии карьеры;
- работодатели, заинтересованные в привлечении молодых талантов.

Приложение обеспечит удобный доступ к актуальным вакансиям, минимизирует временные затраты соискателей и работодателей, а также повысит качество подбора кандидатов. Благодаря сочетанию современных технологий и низкой стоимости услуг, платформа будет конкурентоспособной и востребованной на рынке труда.

Платформа для поиска работы молодыми специалистами будет бесплатной для соискателей, что обеспечит широкий охват аудитории и

повысит активность пользователей. Прибыль планируется получать за счет B2B-услуг для работодателей.

6.2 Расчет инвестиций в разработку программного средства для его реализации на рынке

Инвестициями являются затраты на разработку программного средства. Затраты на заработную плату команды, которая работает над проектом, исходят из состава команды и ее численности, размеров месячной заработной платы каждого участника команды и трудоемкости самой разработки.

Чтобы реализовать проект «Программное средство для поиска работы молодым специалистам» потребуется месяц работы команды разработки, состоящей из бэкенд разработчика, фронтенд разработчика, тестировщика, UI/UX-дизайнера и маркетолога.

Расчет основной заработной платы участников команды осуществляется по формуле:

$$Z_o = \sum_{i=1}^n Z_{\text{чи}} \cdot t_i, \quad (6.1)$$

где $K_{\text{пр}}$ – это коэффициент премий и иных стимулирующих выплат;

n – это категории исполнителей, занятых разработкой программного средства;

$Z_{\text{чи}}$ – часовой оклад исполнителя i -й категории, р;

t_i – трудоемкость работ, выполняемых исполнителем i -й категории, ч.

Расчет затрат на основную заработную плату команды представлен в таблице 6.1.

Таблица 6.1 – Расчет затрат на основную заработную плату команды разработчиков

Категория исполнителя	Месячный оклад, р.	Часовой оклад, р.	Трудоемкость работ, ч.	Итого, р.
Бэкенд разработчик	3000,00	17,86	336	6000,00
Фронтенд разработчик	3000,00	17,86	336	6000,00
UI/UX-дизайнер	2100,00	12,50	160	2000,00
Маркетолог	1800,00	10,71	160	1713,60
<i>Итого</i>				15713,60
Премия и иные стимулирующие выплаты (примем 50 %)				7856,80
<i>Всего</i> затрат на основную заработную плату разработчиков				23570,40

Затраты на дополнительную заработную плату команды разработчиков определяется по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (6.2)$$

где $Н_д$ – это норматив дополнительной заработной платы.

Норматив дополнительной заработной платы примем 15 %, подставляем полученные значения в формулу 6.2:

$$З_д = \frac{23570,4 \cdot 15}{100} = 3535,56 \text{ (р.)}$$

Отчисления на социальные нужды рассчитываются по формуле:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (6.3)$$

где $Н_{соц}$ – это норматив отчислений в ФСЗН и Белгострах.

Норматив отчислений в ФСЗН и Белгострах примем 34,6 %. Подставим полученные значения в формулу 6.3:

$$Р_{соц} = \frac{(23570,4 + 3535,56) \cdot 34,6}{100} = 9378,66 \text{ (р.)}$$

Затраты на прочие расходы рассчитываются по следующей формуле:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (6.4)$$

где $Н_{пр}$ – это норматив прочих расходов.

Примем показатель $Н_{пр}$ за 35 %. Подставим полученные значения в формулу 6.4:

$$Р_{пр} = \frac{23570,4 \cdot 35}{100} = 8249,64 \text{ (р.)}$$

Общая сумма инвестиций на разработку находится по следующей формуле:

$$З_p = З_о + З_д + Р_{соц} + Р_{пр}. \quad (6.5)$$

Подставим в формулу 6.5 полученные значения:

$$З_p = 23570,4 + 3535,56 + 9378,66 + 8249,64 = 44734,26 \text{ (р.)}$$

Таким образом получили общую сумму инвестиций на разработку.

6.3 Расчет экономического эффекта от реализации программного средства на рынке

Целевой аудиторией платформы являются молодые специалисты, студенты, выпускники вузов и работодатели, заинтересованные в поиске талантливых сотрудников. В настоящее время в Республике Беларусь ежегодно выпускается более 50 тысяч молодых специалистов, активно ищущих работу или стажировки [1]. Среди организаций, готовых пригласить учащуюся молодежь на работу с оплатой за их счет или за счет государства, в базе находится 1,7 тысяч компаний [2]. Ожидается, что продуктом будет пользоваться 50 небольших компаний., каждая из которых в среднем будет тратить 200,00 рублей в месяц на услуги поиска кандидатов [3]. Итого в год будет выходить 120000,00 рублей.

Прирост чистой прибыли, полученную командой разработки от реализации программного средства на рынке, можно рассчитать по формуле:

$$\Delta\P_{\text{ч}}^p = (D_p - \text{НДС}) \cdot \left(1 - \frac{H_p}{100}\right)$$

где D_p – предполагаемый доход от услуг за год, р.;

НДС – сумма налога на добавленную стоимость, р.;

H_p – ставка налога на прибыль согласно действующему законодательству, % (примем 25 %).

Налог на добавленную стоимость определяется по формуле

$$\text{НДС} = \frac{Ц \cdot H_{\text{д.с}}}{100 \% + H_{\text{д.с}}},$$

где $Ц$ – цена реализованных услуг за год,

$H_{\text{д.с}}$ – ставка налога на добавленную стоимость в соответствии с действующим законодательством, % (примем 20 %).

Используя имеющиеся значения, проведем расчет НДС:

$$\text{НДС} = \frac{120000 \cdot 20 \%}{100 \% + 20 \%} = 20000,00 \text{ (р.)}$$

Зная размер налога на добавленную стоимость, можно провести расчет прироста численной прибыли, которую получит команда разработки от реализации программного продукта. Используя имеющиеся данные, необходимо посчитать прирост чистой прибыли команды разработки:

$$\Delta\P_q^p = (120000 - 20000p) \cdot 0,75 = 75000,00 \text{ р.}$$

Таким образом прирост чистой прибыли от услуг платформы составит 75000,00 рублей в год.

6.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Оценка экономической эффективности разработки и реализации программного средства на рынке зависит от результатов сравнения инвестиций в его разработку и полученного годового прироста чистой прибыли.

Так как сумма инвестиций на разработку меньше суммы годового экономического эффекта, то есть инвестиции окупятся менее, чем через год, оценка экономической эффективности инвестиций в разработку программного средства осуществляется с помощью расчета рентабельности инвестиций по формуле

$$ROI = \frac{\Delta\P_q^p - Z_p}{Z_p} \cdot 100 \% \quad (6.6)$$

где $\Delta\P_q^p$ – прирост чистой прибыли, полученной от реализации программного средства на рынке, р.;

Z_p – затраты на разработку и реализацию программного средства, р.

Необходимо рассчитать оценку экономической эффективности инвестиций используя уже имеющиеся данные. Проведем расчет оценки эффективности используя формулу (6.6)

$$ROI = \frac{75000 - 44734,26}{44734,26} \cdot 100 \% = 67,66 \%$$

Оценка экономической эффективности платформы для поиска работы молодым специалистам показала рентабельность в 67,66 %. Это превышает установленную ставку рефинансирования (11,5 % годовых), что подтверждает экономическую целесообразность разработки и внедрения продукта. Высокий уровень рентабельности свидетельствует о том, что инвестиции окупаются и приносят прибыль за счет платных услуг для работодателей.

Основной доход платформы формируется за счет таких услуг, как размещение вакансий, продвижение объявлений, доступ к базе резюме и автоматический подбор кандидатов. Однако достижение прогнозируемых показателей возможно только при активном использовании сервиса работодателями. Существует риск, что реальное количество компаний, готовых оплачивать услуги, окажется ниже ожидаемого, что может снизить финансовую эффективность проекта.

При дальнейшем развитии платформы возможно расширение списка услуг, что позволит увеличить доходность, но и увеличить стоимость инвестиций.

ЗАКЛЮЧЕНИЕ

В рамках настоящего дипломного проекта было разработано программное средство для поиска работы, ориентированное на молодых специалистов, в форме интерактивного веб-приложения. Изучены актуальные потребности рынка труда для молодых специалистов и исследованы существующие решения в области онлайн-рекрутинга.

В результате проделанной работы сформулированы детальные функциональные и нефункциональные требования, легшие в основу технического проектирования. Спроектирована трехуровневая клиент-серверная архитектура программного средства, обеспечивающая масштабируемость, модульность и высокую интерактивность. Показана эффективность применения паттерна CQRS для разделения логики чтения и записи, а также использование паттерна Chain of Responsibility в middleware для организации безопасности и логирования запросов. В качестве основных технологий выбраны Kotlin с фреймворком Ktor для серверной части, React для клиентской части и PostgreSQL в качестве системы управления базами данных, с использованием библиотеки JOOQ и R2DBC драйвера для эффективного и типобезопасного доступа к данным.

В ходе практической реализации разработаны ключевые модули платформы, включая:

- модуль управления пользователями, обеспечивающий регистрацию, аутентификацию и авторизацию соискателей и работодателей;
- модуль управления вакансиями, предоставляющий работодателям инструменты для создания, публикации, редактирования и удаления вакансий, а соискателям – возможности поиска и фильтрации;
- модуль управления резюме, позволяющий соискателям создавать, редактировать, хранить и управлять несколькими версиями своих резюме;
- модуль откликов на вакансии, обеспечивающий механизм взаимодействия между соискателями и работодателями, включая отправку и просмотр откликов.

Разработан и испытан комплексный план тестирования, включающий модульные и интеграционные тесты, которые подтвердили работоспособность программного средства в основных сценариях использования и его устойчивость к некорректным входным данным. Проведено экономическое обоснование разработанного решения, подтверждающее его потенциальную эффективность.

Таким образом, изготовлена полноценная и функциональная платформа, которая предложена в качестве действенного инструмента для упрощения процесса трудоустройства молодых специалистов. Разработанное программное средство отличается ориентацией на специфические потребности целевой аудитории, интуитивно понятным интерфейсом и использованием современных технологических решений.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Что такое kotlin и для чего он используется [Электронный ресурс]. – Электронные данные. – Режим доступа: https://code-basics.com/ru/blog_posts/chto-takoe-kotlin-i-dlya-chego-on-ispolzuetsya.
- [2] ktor.io [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ktor.io>.
- [3] About PostgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/about/>.
- [4] jOOQ: The easiest way to write SQL in Java [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.jooq.org/>.
- [5] Марк Тиленс Томас. React в действии. – 3-е изд. – М.: Питер, 2023. – 368 с.
- [6] Понимание REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://yandex.cloud/ru/docs/glossary/rest-api>. – Дата доступа: 3.03.25
- [7] Что такое Docker и как с ним работать [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://selectel.ru/blog/what-is-docker/>.
- [8] Работа в Минске [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ares.by/blog/kak_nayti_rabotu_v_minske/.
- [9] Что такое LinkedIn [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.linkedin.com/help/linkedin/answer/a548441>.
- [10] Praca.by [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://praca.by/faq/>.
- [11] HTTP versus HTTPS [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://aws.amazon.com/compare/the-difference-between-https-and-http/>.
- [12] SQL инъекции и как их предотвратить [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.kaspersky.ru/resource-center/definitions/sql-injection>.
- [13] Межсайтовый скриптинг [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.kaspersky.ru/resource-center/definitions/what-is-a-cross-site-scripting-attack>.
- [14] Дмитрий Жемеров, Светлана Исакова, Айгнер Себастьян, Елизаров Роман. Kotlin в действии. – 2-е изд. – М.: Питер, 2025. – 560 с.
- [15] Ktor Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ktor.io/docs/welcome.html>.
- [16] Рогов Е. В. PostgreSQL 17 изнутри. – М.: ДМК Пресс, 2025. – 668 с.
- [17] IntelliJ IDEA [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.jetbrains.com/idea/>.
- [18] Docker Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.docker.com/>.

[19] Getting Started with react [Электронный ресурс]. – Электронные данные. – Режим доступа: https://developer.mozilla.org/en/US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started.

[20] React Router Official Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://reactrouter.com/>.

[21] UUID [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Glossary/UUID>.

[22] Клиент-серверная архитектура [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://servergate.ru/articles/klient-servernaya-arkhitektura/>.

[23] PostgreSQL индексы [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/indexes-types>.

[24] Введение в jOOQ [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://for-each.dev/lessons/b/-jooq>.

[25] CQRS Pattern [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>.

[26] Chain Of Responsibility Pattern [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://refactoring.guru/design-patterns/chain-of-responsibility>.

[27] Тестирование программного обеспечения [Электронный ресурс]. – Электронные данные. – Режим доступа: https://logrocon.ru/news/testing_is.

ПРИЛОЖЕНИЕ А (обязательное)

Отчет о проверке на антиплагиат

Результаты проверки текста на оригинальность



Проверка на ИИ: Написано человеком ☒

ПРИЛОЖЕНИЕ Б (обязательное)

Исходный код

```
package com.job.library.jooq

import com.job.library.common.db.DatabaseCredentials
import com.job.library.common.pagination.Cursor
import com.job.library.common.pagination.Page
import com.job.library.common.pagination.PageInfo
import com.job.library.jooq.exception.IntegrityViolationException
import com.job.library.jooq.mapper.RecordMapper
import io.r2dbc.spi.ConnectionFactories
import io.r2dbc.spi.ConnectionFactory
import io.r2dbc.spi.ConnectionFactoryOptions
import io.r2dbc.spi.R2dbcDataIntegrityViolationException
import org.jooq.Condition
import org.jooq.DMLQuery
import org.jooq.DSLContext
import org.jooq.Field
import org.jooq.Select
import org.jooq.Table
import org.jooq.exception.IntegrityConstraintViolationException
import org.jooq.impl.DSL
import org.jooq.impl.DSL.count
import org.slf4j.LoggerFactory
import reactor.core.publisher.Flux
import reactor.core.publisher.Mono

class JooqR2dbcContextFactory(
    private val databaseCredentials: DatabaseCredentials,
) {

    companion object {
        private val logger = LoggerFactory.getLogger(JooqR2dbcContextFactory::class.java)
    }

    private val connectionFactory: ConnectionFactory = buildConnectionFactory()

    private fun buildConnectionFactory() = with(databaseCredentials) {
        ConnectionFactories.get(
            ConnectionFactoryOptions.builder()
                .option(ConnectionFactoryOptions.DRIVER, driver)
                .option(ConnectionFactoryOptions.HOST, host)
                .option(ConnectionFactoryOptions.PORT, port)
                .option(ConnectionFactoryOptions.USER, userName)
        )
    }
}
```

```

        .option(ConnectionFactoryOptions.PASSWORD, password)
        .option(ConnectionFactoryOptions.DATABASE, databaseName)
        .build()
    )
}

fun <T> fetchOneAndAwaitNullable(
    mapper: RecordMapper<T>,
    block: DSLContext.() -> Select<out org.jooq.Record>
): T? {
    val dslContext = DSL.using(connectionFactory)

    val select = block(dslContext)

    val value = Flux.from(select)
        .map { mapper(it) }

    return value.blockFirst()
}

fun <T> fetchManyAndAwait(
    mapper: RecordMapper<T>,
    block: DSLContext.() -> Select<*>
): List<T> {
    val dslContext = DSL.using(connectionFactory)

    val select = block(dslContext)

    val value = Flux.from(select)
        .map { mapper(it) }
        .collectList()

    return value.block()
}

fun <T> fetchPageAndAwait(
    cursor: Cursor,
    mapper: RecordMapper<T>,
    table: Table<*>,
    fields: Array<Field<*>>,
    whereConditions: List<Condition> = emptyList(),
): Page<T> {
    val dslContext = DSL.using(connectionFactory)

    val select = dslContext
        .select(*fields)
        .from(table)
        .where(whereConditions)
        .offset(cursor.getOffset())
        .limit(cursor.getLimit())

```

```

        val data = Flux.from(select)
            .map { mapper(it) }
            .collectList()

        val totalCountRecord =
            Mono.from(
                dslContext
                    .select(count().`as`("total_count"))
                    .from(table)
                    .where(whereConditions)
            ).block()
        val totalCount = totalCountRecord?.get("total_count",
Int::class.java) ?: 0

        return Page(
            data = data.block(),
            pageInfo = PageInfo(
                totalPages = computeTotalPages(totalElements =
totalCount, pageSize = cursor.pageSize),
                pageSize = cursor.pageSize,
                pageNumber = cursor.pageNumber,
            )
        )
    }

    private fun computeTotalPages(totalElements: Int, pageSize:
Int): Int {
        return (totalElements + pageSize - 1) / pageSize
    }

    // TODO: add on duplicate factory
    fun use(
        duplicateFactory: (Throwable) -> Throwable = { Integrity-
tyViolationException(it) },
        block: DSLContext.() -> DMLQuery<out org.jooq.Record>
    ) {
        try {
            val dslContext = DSL.using(connectionFactory)

            val query = block(dslContext)

            val result = Flux.from(query).blockFirst()

            if (result == null || result < 1) {
                error("No rows were updated") // todo: throw
only if needed
            }
        } catch (e: IntegrityConstraintViolationException) {
            throw duplicateFactory(e)
        }
    }
}

```

```

package com.job.library.jackson

import com.fasterxml.jackson.core.type.TypeReference
import com.fasterxml.jackson.databind.ObjectMapper
import org.jooq.JSONB
import kotlin.reflect.KClass

fun ObjectMapper.wrapAsJsonbOrNull(value: Any?): JSONB? {
    return JSONB.jsonbOrNull(this.writeValueAsString(value))
}

fun <E : Any> ObjectMapper.listFromJsonbOrNull(value: JSONB?,
elementType: KClass<E>): List<E>? {
    val data = value?.data() ?: return null

    val result: List<E> = this.readValue(
        data,
        this.typeFactory.constructCollection-
Type(List::class.java, elementType.java)
    )

    return result
}

fun <T> ObjectMapper.fromJsonbOrNull(value: JSONB?): T? {
    val data = value?.data() ?: return null

    return this.readValue(data, object : TypeReference<T>() {})
}

//private fun String.isArray() = this.startsWith("[")
package com.job.core.user.command

import com.job.core.user.domain.AccessToken
import com.job.library.command.AbstractCommand
import com.job.library.command.UriAware

data class LoginUserCommand(
    val email: String,
    val password: String,
) : AbstractCommand<AccessToken>() {
    companion object : UriAware {
        override fun uri(): String = "com.job.user.login"
    }
}

package com.job.core.user.handler

import com.job.core.user.command.LoginUserCommand
import com.job.core.user.dao.UserDao
import com.job.core.user.domain.AccessToken
import com.job.core.user.service.TokenManager
import com.job.library.command.CommandHandler

```

```

import com.job.library.security.PasswordEncoder

class LoginUserCommandHandler(
    private val userDao: UserDao,
    private val tokenManager: TokenManager,
    private val passwordEncoder: PasswordEncoder,
) : CommandHandler<LoginUserCommand, AccessToken> {

    companion object {
        val invalidCredentialsError = IllegalStateException(
            "User with such credentials not found"
        )
    }

    override suspend fun handle(command: LoginUserCommand): AccessToken {
        val userLoginInfo = userDao.findUserLoginInfoByEmail(
            command.email
        ) ?: throw invalidCredentialsError

        if (!passwordEncoder.verifyPassword(
            command.password,
            userLoginInfo.passwordHash
        )) {
            throw invalidCredentialsError
        }

        return tokenManager.generateAccessToken(
            email = userLoginInfo.email,
            userRole = userLoginInfo.role,
            userId = userLoginInfo.userId,
        )
    }
}

package com.job.core.user.handler

import com.job.core.user.command.RegisterUserCommand
import com.job.core.user.dao.UserDao
import com.job.core.user.domain.AccessToken
import com.job.core.user.domain.UserRole
import com.job.core.user.domain.UserStatus
import com.job.core.user.domain.command.CreateUserDomainCommand
import com.job.core.user.service.TokenManager
import com.job.library.command.CommandHandler
import com.job.library.common.uuid.Uuid
import com.job.library.security.PasswordEncoder
import java.time.Instant

class RegisterUserCommandHandler(
    private val userDao: UserDao,
    private val passwordEncoder: PasswordEncoder,
    private val tokenManager: TokenManager,
) : CommandHandler<RegisterUserCommand, AccessToken> {

    override suspend fun handle(command: RegisterUserCommand): AccessToken {
        validateCommand(command)
    }
}

```

```

        val userId = Uuid.v7()

        val domainCommand = CreateUserDomainCommand(
            id = userId,
            firstName = command.firstName,
            lastName = command.lastName,
            passwordHash = passwordEncoder.encode(command.password),
            registeredAt = Instant.now(),
            email = command.email,
            emailVerified = false,
            role = command.role,
            userStatus = UserStatus.ACTIVE,
        )

        userDao.create(domainCommand)

        return tokenManager.generateAccessToken(
            userId = userId,
            email = command.email,
            userRole = command.role,
        )
    }

    private fun validateCommand(command: RegisterUserCommand) {
        if (command.role == UserRole.ADMIN) {
            error("Cannot register user with role ${UserRole.ADMIN}")
        }
    }
}

package com.job.library.http

import com.fasterxml.jackson.databind.ObjectMapper
import com.job.library.command.Command
import com.job.library.command.CommandHandlerRegistry
import com.job.library.common.exception.BaseException
import com.job.library.common.pagination.WithCursor
import com.job.library.http.middleware.MiddlewareRegistry
import com.job.library.http.middleware.exception.NoHandlerFoundUriException
import com.job.library.http.response.ErrorResponse
import com.job.library.http.response.HttpResponse
import io.ktor.http.*
import org.slf4j.LoggerFactory
import java.io.InputStream

class RpcRouter(
    private val objectMapper: ObjectMapper,
    private val commandHandlerRegistry: CommandHandlerRegistry,
    private val middlewareRegistry: MiddlewareRegistry,
) {

```

```

        companion object {
            private val logger = LoggerFactory.getLogger(RpcRouter::class.java)
        }

        suspend fun routePost(uri: String, body: InputStream, headers: Headers): HttpResponse {
            try {
                val handlerInfo = commandHandlerRegistry.findHandlerInfo(uri) ?: throw NoHandlerForUriException(uri = uri)

                val command = objectMapper.readValue(body, handlerInfo.commandClass.java)

                validateCommand(command)

                runMiddlewaresBefore(command, headers)

                val response = handlerInfo.commandHandler.handle(command)

                return HttpResponse(statusCode = HttpStatusCode.OK, body = response)
            } catch (e: BaseException) {
                logger.error("Error occurred on uri {}. Message: {}", uri, e.message, e)

                return HttpResponse(
                    statusCode = HttpStatusCode.OK,
                    body = ErrorResponse(
                        code = e.code,
                        message = e.message,
                    )
                )
            } catch (e: Throwable) {
                logger.error("Error occurred on uri {}. Message: {}", uri, e.message, e)

                return HttpResponse(
                    statusCode = HttpStatusCode.OK,
                    body = ErrorResponse(
                        message = e.message ?: "Internal Server Error",
                    )
                )
            } finally {
                // todo: rewrite to chain of responsibility
                runMiddlewaresAfter(command, headers)
            }
        }
    }

```



```

        private fun runMiddlewaresBefore(command: Command<*>, headers: Headers) {
            middlewareRegistry.getMiddlewares()
                .forEach { it.doBefore(command, headers) }
        }

        private fun runMiddlewaresAfter(command: Command<*>, headers: Headers) {
            middlewareRegistry.getMiddlewares()
                .forEach { it.doAfter(command, headers) }
        }

        private fun validateCommand(command: Command<*>) {
            if (command is WithCursor) {
                command.cursor.validate()
            }
        }
    }
}

package com.job.library.security

import org.mindrot.jbcrypt.BCrypt

class PasswordEncoder(
    private val secretKey: String
) {

    fun encode(password: String): String {
        return BCrypt.hashpw(password, secretKey)
    };

    fun verifyPassword(password: String, passwordHash: String): Boolean {
        return BCrypt.checkpw(password, passwordHash)
    }
}

package com.job.core.di

import com.fasterxml.jackson.databind.DeserializationFeature
import com.fasterxml.jackson.databind.ObjectMapper
import com.fasterxml.jackson.databind.SerializationFeature
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule
import com.fasterxml.jackson.module.kotlin.jacksonObjectMapper
import com.job.core.company.di.companyModule
import com.job.core.email.di.emailModule
import com.job.core.resume.di.resumeModule
import com.job.core.user.di.userModule
import com.job.core.user.service.TokenManager
import com.job.core.vacancy.di.vacancyModule
import com.job.library.command.Command
import com.job.library.command.CommandHandler
import com.job.library.command.CommandHandlerRegistry
import com.job.library.command.SubjectRegistry

```

```

import com.job.library.common.db.DatabaseCredentials
import com.job.library.common.security.JwtInfo
import com.job.library.di.autoBind
import com.job.library.http.RpcRouter
import com.job.library.http.middleware.AuthenticationMiddleware
import com.job.library.http.middleware.Middleware
import com.job.library.http.middleware.MiddlewareRegistry
import com.job.library.jooq.JooqR2dbcContextFactory
import com.job.library.security.PasswordEncoder
import org.kodein.di.DI
import org.kodein.di.allInstances
import org.kodein.di.bind
import org.kodein.di.direct
import org.kodein.di.singleton

val mainModule = DI.Module("mainModule") {
    bind<CommandHandlerRegistry>() with singleton {
        val registry = CommandHandlerRegistry()

        di.direct.allInstances<CommandHandler<Command<*>, *>>()
            .forEach { registry.register(it) }

        registry
    }

    autoBind<RpcRouter>()
    autoBind<AuthenticationMiddleware>()

    bind<TokenManager>() with singleton {
        // TODO: store secret in env
        TokenManager(
            jwtInfo = JwtInfo(
                issuer = "com.job.first",
                audience = "audience",
                secret = "supersecret"
            ),
        )
    }

    bind<MiddlewareRegistry>() with singleton {
        MiddlewareRegistry(
            allInstances<Middleware>()
        )
    }

    bind<ObjectMapper>() with singleton {
        jacksonObjectMapper()
            .registerModule(JavaTimeModule())
            .configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false)
    }
}

```

```

        .configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false)
    }

    autoBind<JooqR2dbcContextFactory>()

    autoBind<SubjectRegistry>()

    bind<DatabaseCredentials>() with singleton {
        // TODO: think of where to store credentials
        DatabaseCredentials(
            userName = "postgres",
            password = "password",
            databaseName = "job_first",
            host = "localhost",
            driver = "postgresql",
            port = 5432,
        )
    }

    bind<PasswordEncoder>() with singleton {
        // TODO: store secret
        PasswordEncoder(
            secretKey = "\$2a\$10\$AVPT05HwxGW.2eXHQmzKBO"
        )
    }
}

val di = DI {
    import(mainModule)
    import(vacancyModule)
    import(userModule)
    import(resumeModule)
    import(companyModule)
    import(emailModule)
}

package com.job.core.vacancy.handler

import com.job.core.vacancy.command.GetEmployerVacancyDetailsQuery
import com.job.core.vacancy.dao.VacancyDao
import com.job.core.vacancy.dto.EmployerVacancyDetails
import com.job.library.command.CommandHandler

class GetEmployerVacancyDetailsQueryHandler(
    private val vacancyDao: VacancyDao,
) : CommandHandler<GetEmployerVacancyDetailsQuery, EmployerVacancyDetails> {

    override suspend fun handle(query: GetEmployerVacancyDetailsQuery): EmployerVacancyDetails {

```

```

        val vacancy = vacancyDao.findById(query.vacancyId) ?:
error("Vacancy not found")

        val applies = vacancyDao.getVacancyApplies(vacancyId =
query.vacancyId)

        return EmployerVacancyDetails(
            id = vacancy.id,
            status = vacancy.status,
            publisher = vacancy.publisher,
            title = vacancy.title,
            salaryMin = vacancy.salaryMin,
            salaryMax = vacancy.salaryMax,
            workType = vacancy.workType,
            location = vacancy.location,
            description = vacancy.description,
            experienceLevel = vacancy.experienceLevel,
            createdAtMillis = vacancy.createdAtMillis,
            editedAtMillis = vacancy.editedAtMillis,
            applies = applies.sortedByDescending { it.appliedAt
},
        )
    }
}

package com.job.library.command

import kotlin.reflect.KClass
import kotlin.reflect.KType
import kotlin.reflect.KTypeProjection
import kotlin.reflect.full.allSuperclasses
import kotlin.reflect.full.allSupertypes
import kotlin.reflect.full.companionObjectInstance

data class CommandHandlerInfo(
    val commandClass: KClass<Command<*>>,
    val commandHandler: CommandHandler<Command<*>, *>
)

class CommandHandlerRegistry {

    private val uriToCommandHandlerInfo: MutableMap<String, Com-
mandHandlerInfo> = mutableMapOf()

    fun register(handler: CommandHandler<Command<*>, *>) {
        val commandClass = handler.getCommandClass()

        val companionObjectInstance = commandClass.companionOb-
jectInstance
        ?: error("Command ${commandClass.simpleName} should
contain object instance")

        if (companionObjectInstance is UriAware) {
            val uri = companionObjectInstance.uri()

```

```

        if (uriToCommandHandlerInfo.containsKey(uri)) {
            error("Command ${commandClass.simpleName} already registered as ${companionObjectInstance.uri()}")
        }

        uriToCommandHandlerInfo[uri] = CommandHandlerInfo(commandClass, handler)
    }

    fun findHandlerInfo(uri: String): CommandHandlerInfo? {
        return uriToCommandHandlerInfo[uri]
    }

    private fun CommandHandler<*, *>.getCommandClass():
    KClass<Command<*>> {
        this.javaClass.kotlin.allSupertypes.forEach { type:
    KType ->
        type.arguments.forEach arguments@{ typeProjection:
    KTypeProjection ->
            val classifier = typeProjection.type?.classifier
        ?: return@arguments

            classifier as KClass<*>

            @Suppress("UNCHECKED_CAST")
            if (classifier.allSuperclasses.contains(Com-
mand::class)) {
                return classifier as KClass<Command<*>>
            }
        }
    }
}

```