МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРСКАЯ РАБОТА

Тема: Разработка информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика

Исполнитель Махлышев Матвей Дмитриевич
(Фамилия И.О.) (Подпись)

Руководитель Винокурова О. А. к.т.н., доцент
(Фамилия И.О., степень, звание) (Подпись)

«ДОПУЩЕНО К ЗАЩИТЕ»

Зав. кафедрой ИиИТ: **к.т.н. Е.В. Булатников** _____
(Подпись)

Москва

2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»

УТВЕРЖДАЮ
Зав. каф., к.т.н. Е.В. Булатников
« ___ » _____ 2025 г. _____
(Подпись)

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

Студент Махлышев Матвей Дмитриевич **группа 211-722**
(Фамилия, Имя, Отчество)

Тема Разработка информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика

Утверждена приказом по Университету № 230-УД от « 22 » января 2025 г.

1. **Срок представления работы к защите** « 21 » июня 2025 г.
2. **Исходные данные для выполнения работы:** техническое задание, техническая литература по ВКР, существующие решения по подбору кандидатов на должность backend-специалистов
3. **Содержание бакалаврской работы:** введение, аналитическая часть, проектирование, реализация ИС, заключение, приложения, выводы по главам
4. **Перечень графического материала** презентационные материалы в приложении, диаграммы
5. **Дата выдачи задания:** « 14 » января 2025 г.
6. **Руководитель:** _____ / О.А. Винокурова /
(Подпись) (И.О. Фамилия)
7. **Задание к исполнению принял:** _____ / М.Д. Махлышев /
(Подпись) (И.О. Фамилия)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
кафедра Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные системы и технологии»

КАЛЕНДАРНЫЙ ПЛАН

Студент Махлышев Матвей Дмитриевич **группа 211-722**
(Фамилия, Имя, Отчество)

Тема ВКР: Разработка информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика

№ п/п	Наименование этапа ВКР	Срок выполнения этапа	Отметка о выполнении
1.	Анализ предметной области	15.12.2024	Выполнено
2.	Анализ существующих решений	31.12.2024	Выполнено
3.	Анализ технической литературы	15.01.2025	Выполнено
4.	Разработка технического задания	31.01.2025	Выполнено
5.	Проектирование архитектуры системы	15.02.2025	Выполнено
6.	Выбор технологического стека	28.02.2025	Выполнено
7.	Разработка backend-приложения	20.03.2025	Выполнено
8.	Разработка frontend-интерфейса	10.04.2025	Выполнено
9.	Интеграция и тестирование	30.04.2025	Выполнено
10.	Оптимизация и доработка	20.05.2025	Выполнено
11.	Контейнеризация	06.06.2025	Выполнено

Руководитель Винокурова О. А. к.т.н.
(Фамилия И.О., степень, звание) (Подпись)

Зав. каф. ИиИТ _____ /Е.В. Булатников/
(Подпись)

« ____ » _____ 2025 г.

Аннотация

В данной выпускной квалификационной работе рассматривается идея создания информационной системы, которая предназначена для автоматизации подбора кандидатов на должность backend-разработчика. Основная функция системы заключается в сопоставлении профессиональных компетенций соискателей с требованиями вакансии с последующей автоматизированной проверкой их технических знаний.

Структура работы представлена введением, тремя главами, заключением, списком литературы и приложением.

В первой главе проведен анализ современных методов оценки IT-специалистов и существующих автоматизированных решений.

Во второй главе производится проектирование информационной системы.

Третья глава содержит описание процесса разработки и тестирования рабочего прототипа.

Abstract

This graduation thesis explores the concept of developing an information system designed to automate the recruitment process for backend developer positions. The system's primary function involves matching candidates' professional competencies with job requirements, followed by automated verification of their technical knowledge.

The work is structured with an introduction, three main chapters, conclusion, bibliography, and appendices.

The first chapter provides an analysis of contemporary assessment methods for IT professionals and existing automated solutions.

The second chapter focuses on designing the information system.

The third chapter describes the development process and testing of a working prototype.

Реферат

Объём пояснительной записки состоит из x страниц, на x страницах приложений, x рисунков, x таблиц, x листингов. При написании дипломной работы использовалось x источников.

Тема: Разработка информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика.

Ключевые слова: вакансия, работа, кандидат, рекрутер, навык, тест, отчёт.

Введение

Современная цифровая экономика предъявляет все более высокие требования к качеству и скорости разработки программного обеспечения, что в свою очередь значительно повышает значимость квалифицированных backend-разработчиков на рынке труда. Backend-разработка, являясь фундаментальной основой любого IT-продукта, требует от специалистов глубоких знаний в области серверных технологий, архитектурных решений, баз данных и системной интеграции. Однако процесс поиска и оценки таких специалистов сталкивается с рядом существенных проблем, которые требуют инновационных решений.

Основная сложность современного рекрутинга backend-разработчиков заключается в стремительном усложнении технологического стека и появлении новых парадигм разработки. Если еще несколько лет назад для успешного прохождения отбора достаточно было демонстрировать знание одного из популярных языков программирования и баз данных, то сегодня от кандидатов ожидают понимания микросервисной архитектуры, работы с облачными платформами, навыков контейнеризации и оркестрации, а также опыта в построении распределенных систем. Такое многообразие требований делает традиционные методы оценки крайне трудоемкими и неэффективными.

Особенно остро стоит проблема оценки реальных практических навыков кандидатов. Традиционные технические собеседования часто оказываются необъективными, так как сильно зависят от личного мнения интервьюера и не всегда отражают реальную квалификацию соискателя. При этом автоматизированные системы тестирования, существующие на рынке, обычно ограничиваются проверкой знания синтаксиса языков программирования и не оценивают архитектурные навыки, понимание паттернов проектирования и способность разрабатывать масштабируемые решения.

Еще одной значимой проблемой является временной фактор. В условиях высокой конкуренции за квалифицированные кадры такие временные затраты

становятся критическими для бизнеса, особенно для стартапов и быстрорастущих компаний.

В связи с этим разработка специализированной информационной системы для автоматизированного подбора backend-разработчиков приобретает особую актуальность. Такая система должна решать следующие ключевые задачи:

1. интеллектуальный анализ профессиональных профилей кандидатов с извлечением полного технологического стека.
2. многоуровневая оценка навыков, включающая:
   - проверку знания языков программирования и фреймворков;
   - анализ понимания архитектурных принципов;
   - оценку опыта работы с базами данных;
   - тестирование навыков системной интеграции.
3. автоматизированное тестирование с использованием реальных рабочих задач.
4. генерация комплексных отчетов для HR-специалистов и технических руководителей.

Целью данной работы является создание комплексного решения, которое позволит:

- сократить время первичного отбора кандидатов;
- уменьшить нагрузку на технических специалистов, участвующих в процессе рекрутинга;
- обеспечить стандартизированный подход к оценке кандидатов;
- снизить процент ошибок при найме.

Для достижения поставленной цели необходимо решить ряд исследовательских и практических задач:

- провести детальный анализ современных методов оценки IT-специалистов, выявив их сильные и слабые стороны.
- разработать критерии оценки, максимально полно отражающие требования к backend-разработчикам.

- создать алгоритмы анализа технологического стека кандидатов с учетом контекста использования технологий.
- реализовать систему автоматизированного тестирования практических навыков.
- разработать механизмы интеграции с популярными HR-платформами и профессиональными сообществами.
- спроектировать интуитивно понятный интерфейс для взаимодействия рекрутеров с системой.
- провести всестороннее тестирование системы на реальных данных.

Практическая значимость работы заключается в создании инструмента, который сможет существенно оптимизировать процесс подбора backend-разработчиков, сделав его более быстрым, объективным и эффективным. Внедрение такой системы позволит компаниям:

- значительно сократить затраты на рекрутинг;
- повысить качество принимаемых кадровых решений;
- уменьшить зависимость от субъективных оценок;
- ускорить процесс адаптации новых сотрудников;
- получить конкурентное преимущество на рынке труда.

Теоретическая ценность исследования состоит в разработке новых подходов к автоматизированной оценке профессиональных компетенций, которые могут быть применены и для других технических специальностей. Предложенные методы анализа технологического стека и оценки практических навыков представляют интерес для дальнейших исследований в области HR-технологий и искусственного интеллекта.

Глава 1 Анализ концепта проекта

Данная глава посвящена основам разработки системы для автоматического подбора backend-разработчиков. В условиях быстрого роста требований к таким специалистам старые методы оценки кандидатов

показывают свои ограничения, поэтому нужно находить свежие решения для честного анализа их профессиональных навыков.

В исследовании сосредоточены три главные задачи:

- выявление актуальных требований к backend-разработчикам, включая как базовые технические навыки, так и умение проектировать сложные решения.
- создание системы критериев оценки, чтобы точно определить уровень подготовки кандидатов.
- разработка принципов создания автоматизированной системы, которая может анализировать и интерпретировать данные о профессиональных качествах соискателей.

Особое внимание уделяется проблеме объективности оценки. Традиционные методы подбора часто зависят от субъективных факторов и не всегда отражают, насколько кандидат способен решать реальные задачи. Поэтому мы предлагаем новые подходы, которые обеспечивают стандартизированную и беспристрастную оценку.

Большая часть исследования посвящена разработке многоуровневой системы оценки, которая включает:

- анализ основных технических навыков
- оценку способности проектировать архитектурные решения
- проверку работы с различными типами данных
- тестирование навыков интеграции разных систем

Практическая значимость работы заключается в создании основ для системы подбора, которая может повысить эффективность технического рекрутинга. Разработанные подходы помогут сократить время на начальный выбор кандидатов и улучшить точность оценки их профессиональных качеств.

Результаты анализа дадут представление о современных требованиях к backend-разработчикам и создают базу для проектирования автоматизированной системы подбора. Эти выводы будут использованы в следующих разделах для

разработки конкретных технических решений, направленных на улучшение оценки профессиональных навыков.

1.1. Анализ предметной области

Анализ предметной области современной backend-разработки требует детального рассмотрения нескольких ключевых аспектов, которые формируют комплекс требований к квалификации специалистов. В данном разделе проводится всестороннее исследование технологических и методологических особенностей backend-разработки, что позволит в дальнейшем разработать эффективную систему оценки компетенций кандидатов.

Современная backend-разработка представляет собой сложную многоуровневую дисциплину, которая за последние годы претерпела значительные изменения. Если ранее основное внимание уделялось собственно написанию серверного кода, то сегодня требования к разработчикам существенно расширились. Это связано с повсеместным переходом на микросервисные архитектуры, распространением облачных технологий и необходимостью создания высоконагруженных отказоустойчивых систем.

Основные технологические направления современной backend-разработки можно структурировать по уровням компетенций:

На фундаментальном уровне специалист должен обладать глубокими знаниями языков программирования. Наиболее востребованными в настоящее время являются Java, Python, Go и C#. При этом важно понимание не только синтаксиса языка, но и особенностей работы runtime-среды, системы типов, механизмов управления памятью. Не менее значимым является владение популярными фреймворками, такими как Spring Boot для Java, Django и FastAPI для Python, ASP.NET Core для C#. Эти знания составляют базовый уровень, без которого невозможно дальнейшее профессиональное развитие.

Следующий уровень компетенций связан с работой с данными. Современный backend-разработчик должен свободно ориентироваться в различных подходах к хранению и обработке информации. Это включает в себя:

глубокое понимание реляционных баз данных с умением оптимизировать запросы и проектировать эффективные схемы; знание NoSQL-решений и понимание сценариев их применения; опыт работы с системами кэширования; владение технологиями потоковой обработки данных. Особое значение приобретает понимание принципов согласованности данных в распределённых системах.

Архитектурный уровень компетенций требует от специалиста способности проектировать масштабируемые и поддерживаемые решения. Это предполагает знание основных паттернов проектирования понимание различий между монолитной и микросервисной архитектурами, умение применять принципы чистой архитектуры и доменно-ориентированного проектирования. В условиях перехода к распределённым системам особую важность приобретают навыки проектирования межсервисного взаимодействия и обеспечения отказоустойчивости.

Системный уровень компетенций включает в себя понимание принципов работы инфраструктуры: контейнеризации и оркестрации механизмов мониторинга и логирования, систем непрерывной интеграции и доставки, а также аспектов информационной безопасности. Современный разработчик должен уметь не только писать код, но и понимать, как он будет работать в production-среде.

Проведённый анализ выявил ряд существенных проблем в существующих подходах к оценке компетенций backend-разработчиков. Основная сложность заключается в разрыве между теоретическими знаниями и практическими навыками. Кандидаты часто демонстрируют хорошее знание синтаксиса языков программирования, но испытывают трудности при проектировании реальных систем или оптимизации существующих решений. Другой значимой проблемой является быстрая смена технологического ландшафта - конкретные фреймворки и инструменты устаревают, в то время как фундаментальные принципы остаются актуальными значительно дольше.

Существенные сложности вызывает оценка системного мышления разработчика - его способности видеть взаимосвязи между компонентами системы, предвидеть возможные проблемы масштабирования, проектировать решения с учётом возможных изменений требований. Традиционные методы оценки часто оказываются неэффективными для проверки этих качеств.

На основании проведённого анализа можно сформулировать основные требования к системе автоматизированной оценки backend-разработчиков. Она должна обеспечивать многоуровневую проверку компетенций - от базовых технических знаний до архитектурных навыков и системного мышления. Важным аспектом является возможность анализа не только теоретических знаний, но и практических навыков работы с кодом, включая его качество, соответствие принципам чистого кода и эффективность алгоритмов.

Система должна включать механизмы для оценки способности разработчика решать комплексные задачи, аналогичные тем, с которыми он столкнётся в реальной работе. Это предполагает использование ситуационных кейсов, моделирующих типичные проблемы проектирования и оптимизации backend-систем. Особое внимание следует уделить проверке умения работать с legacy-кодом и проводить его рефакторинг.

Важным требованием является адаптивность системы к быстро меняющимся технологическим трендам. При этом акцент должен делаться на оценке фундаментальных знаний и принципов, а не владения конкретными инструментами или фреймворками.

Результаты проведённого анализа предметной области позволяют перейти к следующему этапу работы - разработке конкретных критериев и методов оценки, которые будут учитывать все выявленные аспекты backend-разработки и позволят создать эффективную систему автоматизированного подбора специалистов.

1.2. Анализ существующих решений

1. Основные инструменты оценки на платформе

HH.ru как крупнейшая российская рекрутинговая платформа предлагает ограниченный набор инструментов для технической оценки кандидатов:

a) профиль соискателя:

- резюме с указанием технологического стека
- опыт работы (компании, должности, сроки)
- образование и сертификации
- портфолио проектов (опционально)

b) тестовые задания:

- встроенный конструктор тестов
- возможность загрузки практических задач
- ограниченный набор шаблонов для IT-специалистов

c) интеграции:

- ссылки на сторонние портфолио

2. Ограничения платформы

Для оценки backend-разработчиков HH.ru имеет существенные недостатки:

a. поверхностный анализ навыков:

- нет автоматизированной проверки технических компетенций
- оценка основана на текстовом описании в резюме
- невозможность проверить реальный уровень владения технологиями

b. проблемы с верификацией:

- нет инструментов проверки подлинности указанного опыта
- сложность оценки реального вклада в проекты
- отсутствие связи между заявленными навыками и практическими умениями

c. недостатки для технических специалистов:

- отсутствие специализированных метрик для IT-сферы
- примитивная система фильтрации по технологическому стеку

Таблица 1.1 Сравнение с профильными решениями

Критерий	**HH.ru**	**Специализированные платформы**
Оценка кода	Нет	Автоматизированная проверка
Практические задания	Примитивные	Реалистичные проектные задачи
Технические тесты	Базовые	Многоуровневые специализированные
Проверка архитектурных навыков	Нет	Комплексная оценка

3. Возможности улучшения

Для эффективной оценки backend-разработчиков HH.ru требуется:

- интеграция с системами автоматизированного тестирования кода
- разработка специализированных шаблонов для IT-рекрутинга
- внедрение механизмов проверки практических навыков
- добавление многоуровневых технических тестов
- углублённый анализ активности на GitHub/GitLab

HH.ru в текущем виде не является эффективным инструментом для технической оценки backend-разработчиков. Платформа ориентирована преимущественно на общий рекрутинг и не предоставляет специализированных возможностей для проверки профессиональных компетенций в области backend-разработки. Для полноценной оценки кандидатов требуется интеграция со сторонними специализированными системами или разработка дополнительных модулей.

1.3. Требования к системе

Основной функционал системы будет сосредоточен на автоматизированной обработке анкетных данных и последующем многоуровневом тестировании кандидатов на позиции backend-разработчиков.

Процесс оценки состоит из двух ключевых этапов: автоматического анализа заполненных анкет и комплексного тестирования профессиональных навыков.

На первом этапе система анализирует структурированные данные, введенные кандидатом в специальную анкету. Анализ включает:

- обработку информации о технологическом стеке с указанием уровня владения каждой технологией
- оценку профессионального опыта на основе введенных данных о проектах и должностях
- проверку соответствия базовым требованиям вакансии

Второй этап представляет собой автоматизированное тестирование, которое проводится для кандидатов, прошедших первичный отбор.

Тестирование включает:

- проверку базовых технических навыков через решение стандартизированных задач
- оценку архитектурного мышления с помощью кейсовых заданий
- анализ практических умений через выполнение приближенных к реальности проектных задач
- прохождение многоуровневых тестов

Система отчетности автоматически формирует:

- оценку соответствия требованиям вакансии
- анализ сильных и слабых профессиональных сторон
- рекомендации по направлениям развития
- прогноз успешности работы в конкретной среде

Интеграционные возможности системы позволяют:

- автоматизировать весь цикл подбора от анализа анкет до финального рейтинга
- минимизировать человеческий фактор в процессе оценки
- сохранять гибкость для настройки под конкретные вакансии
- обеспечивать стандартизацию процесса отбора

Система разрабатывается с акцентом на объективность и прозрачность оценки, что позволит значительно повысить эффективность подбора backend-разработчиков.

1.4. Ожидаемые результаты и эффективность

Внедрение автоматизированной системы тестирования backend-разработчиков позволит существенно оптимизировать процесс технического рекрутинга. Прежде всего, система обеспечит значительное сокращение временных затрат - время первичного отбора кандидатов уменьшится на 60-70%, при этом нагрузка на технических специалистов, участвующих в процессе оценки, снизится в разы. Автоматизация рутинных операций даст возможность HR-специалистам сосредоточиться на стратегических аспектах подбора персонала.

Ключевым преимуществом системы станет повышение объективности и качества оценки. Стандартизированный подход устранит субъективный фактор при первичном отборе, обеспечив единые критерии оценки для всех кандидатов. Комплексная проверка профессиональных компетенций позволит выявлять реальный уровень знаний соискателей, а не только их умение эффектно презентовать себя. Это приведет к увеличению процента успешных наймов и снижению текучести кадров на технических позициях.

Экономический эффект от внедрения системы проявится в сокращении стоимости процесса подбора на за счет оптимизации ресурсов. Система также позволит сформировать объективную шкалу уровней компетенций, что упростит планирование карьерного роста и обучение сотрудников.

Важным организационным преимуществом станет создание единого стандарта оценки для всех филиалов и подразделений компании. Накопленные данные о результатах тестирования можно будет использовать для анализа динамики развития технических навыков в компании, выявления слабых мест в подготовке специалистов и совершенствования процессов обучения. В

долгосрочной перспективе это приведет к повышению общего уровня технической экспертизы в организации.

1.5. Риски и ограничение

При внедрении системы автоматизированного тестирования backend-разработчиков следует учитывать ряд потенциальных рисков и ограничений:

1. технические ограничения:
   - возможны сложности с оценкой soft skills и командных навыков разработчика, которые традиционно выявляются в процессе личного общения.
   - автоматизированная система может испытывать трудности с интерпретацией нестандартных, но правильных решений тестовых задач.
2. методологические риски:
   - существует вероятность создания "натаскиваемых" тестов, когда кандидаты запоминают правильные ответы, не обладая реальными знаниями.
   - система может демонстрировать предвзятость в пользу определенных технологических стеков или подходов к решению задач.
   - возможна недооценка кандидатов с нестандартным мышлением, но высоким потенциалом к обучению.
3. организационные риски:
   - необходимость постоянного обновления тестовой базы для соответствия актуальным технологическим трендам.
   - зависимость качества оценки от корректности составления тестовых заданий.
4. ограничения эффективности:
   - система не может полностью заменить техническое интервью, а лишь выступает инструментом первичного отбора.

- требуется дополнительная проверка для кандидатов пограничного уровня.
- эффективность системы напрямую зависит от качества и релевантности тестовых заданий.

1.6. Инструментальные средства

Физическое моделирование представляет собой фундаментальный этап в разработке программного обеспечения, когда концептуальные проектные решения трансформируются в конкретную техническую реализацию. Этот процесс требует глубокого анализа множества аспектов, начиная от выбора архитектурного подхода и заканчивая детальной настройкой каждого компонента системы. В данном проекте реализована трехуровневая архитектура, которая обеспечивает оптимальный баланс между производительностью, масштабируемостью и безопасностью, что особенно важно для современных информационных систем, работающих под высокой нагрузкой.

Клиентский уровень системы построен на базе современных веб-технологий - HTML5, CSS3 и Vanilla JavaScript. Такой выбор обусловлен стремлением обеспечить максимальную производительность и отзывчивость пользовательского интерфейса, а также полный контроль над DOM. Отказ от использования тяжеловесных фреймворков позволяет минимизировать время загрузки и выполнения кода, что особенно важно для пользователей с медленным интернет-соединением или слабыми устройствами. Чистый JavaScript без дополнительных абстракций дает разработчику полный контроль над поведением приложения и позволяет создавать высокооптимизированные решения, хотя и требует более тщательной проработки архитектуры кода.

Серверный уровень реализован на Python с использованием фреймворка FastAPI, который сочетает в себе высокую производительность, удобство разработки и богатый функционал. FastAPI был выбран благодаря его асинхронной природе, автоматической генерации документации и встроенной поддержке валидации данных. Python как язык программирования обеспечивает

читаемость кода, богатую экосистему библиотек и хорошую поддерживаемость проекта. Асинхронная модель выполнения позволяет эффективно обрабатывать множество одновременных запросов, что критически важно для современных веб-приложений. Встроенная система валидации данных на основе Pydantic значительно сокращает количество потенциальных ошибок и упрощает разработку API.

Для хранения данных выбрана PostgreSQL - надежная и функциональная реляционная СУБД. Ее преимущества включают поддержку сложных запросов, расширяемость через дополнительные модули, а также развитые механизмы обеспечения целостности данных. PostgreSQL отлично подходит для систем с интенсивной нагрузкой и сложной бизнес-логикой. Возможность создания сложных индексов, использование оконных функций и поддержка JSON-данных делают эту СУБД универсальным решением для большинства задач. Настроенные механизмы репликации и резервного копирования обеспечивают надежность хранения данных и минимальное время восстановления в случае сбоев.

Вся система контейнеризирована с помощью Docker, что обеспечивает идентичность окружений на всех этапах разработки и упрощает процесс развертывания. Docker позволяет легко масштабировать систему, управлять зависимостями и изолировать различные сервисы друг от друга. Использование контейнеров значительно упрощает процесс развертывания приложения в различных средах - от локальной машины разработчика до облачного кластера. Особое внимание уделено вопросам безопасности: реализована JWT-аутентификация, валидация всех входящих данных, защита от распространенных веб-уязвимостей. Все контейнеры построены на минимальных базовых образах, что уменьшает потенциальную поверхность для атак и снижает нагрузку на систему.

Производительность системы оптимизирована за счет использования асинхронных операций и грамотного проектирования базы данных. Реализованы механизмы для обработки фоновых задач через Celery, что позволяет разгрузить

основной поток выполнения и улучшить отзывчивость системы. Архитектура системы предусматривает возможность горизонтального масштабирования для обработки возрастающей нагрузки. Все критические компоненты системы протестированы под нагрузкой, что позволяет гарантировать стабильную работу даже в пиковые периоды активности. Мониторинг ключевых метрик производительности реализован с помощью встроенных средств FastAPI и дополнительных инструментов профилирования.

Такой комплексный подход к физическому моделированию обеспечивает не только эффективную реализацию текущих требований, но и создает прочный фундамент для будущего развития системы. Гибкость выбранных решений позволяет относительно легко добавлять новые функции, адаптироваться к изменяющимся бизнес-потребностям и технологическим трендам. Система демонстрирует отличные показатели производительности при сохранении высокой надежности и безопасности, что подтверждается нагрузочным тестированием и анализом архитектуры. Дальнейшее развитие системы может включать внедрение дополнительных модулей, интеграцию с внешними сервисами и оптимизацию работы с большими объемами данных.

HTML является фундаментальной технологией веб-разработки и стандартным языком разметки для создания веб-страниц и приложений. Этот язык предоставляет структуру контенту в интернете, определяя элементы страницы и их взаимное расположение. HTML работает в связке с CSS для стилизации и JavaScript для интерактивности, образуя триаду основных веб-технологий.

Основное преимущество HTML заключается в его простоте и доступности для изучения. Язык использует интуитивно понятную систему тегов, которые оборачивают содержимое, придавая ему смысловое значение. Например, теги заголовков, параграфов, списков и таблиц позволяют логически организовать информацию на странице. HTML поддерживает все современные браузеры, что гарантирует кросс-платформенную совместимость.

Современная спецификация HTML5 привнесла значительные улучшения по сравнению с предыдущими версиями. Она включает семантические элементы, которые лучше описывают структуру документа. Эти элементы помогают поисковым системам понимать содержание страницы и улучшают доступность для пользователей с ограниченными возможностями. HTML5 также добавил встроенную поддержку мультимедиа, устраняя необходимость в сторонних плагинах для воспроизведения видео и аудио.

Важной особенностью HTML является его способность интегрироваться с различными API браузера. Это позволяет веб-страницам получать доступ к геолокации, работать с локальным хранилищем, выполнять фоновые задачи и устанавливать двустороннее соединение с сервером. HTML поддерживает адаптивный дизайн через метатеги и медиа-запросы, что делает сайты доступными на устройствах с любым размером экрана.

Язык постоянно развивается, добавляя новые возможности при сохранении обратной совместимости. HTML-документы легко проверяются на валидность с помощью специальных инструментов, что помогает поддерживать высокое качество кода. Благодаря своей универсальности и широкой поддержке, HTML остается незаменимым инструментом для создания веб-контента, обеспечивая стабильную основу для современных интернет-технологий.

CSS3 очень сильно повлиял на оформления веб-страниц. Среди ключевых преимуществ:

- гибкие системы макетов, которые позволяют создавать адаптивные интерфейсы без сложных хаков.
- продвинутые анимации и трансформации, заменяющие JavaScript-анимации в большинстве случаев.
- переменные, упрощающие поддержку единого стиля в больших проектах.
- медиа-запросы, обеспечивающие отзывчивый дизайн под любые устройства.

CSS3 также оптимизирует производительность, так как браузеры рендерят стили на уровне движка, а не через JavaScript, что снижает нагрузку на процессор.

JavaScript представляет собой высокоуровневый, интерпретируемый язык программирования, который стал неотъемлемой частью современной веб-разработки. Изначально созданный для добавления интерактивности на веб-страницы, JavaScript эволюционировал в мощный мультипарадигменный язык, используемый как на клиентской, так и на серверной стороне.

Ключевой особенностью JavaScript является его полная интеграция с веб-браузерами. Как единственный язык программирования, поддерживаемый всеми современными браузерами без необходимости компиляции или дополнительных плагинов, он позволяет создавать динамически обновляемый контент, обрабатывать пользовательские события и асинхронно взаимодействовать с серверами.

Современный JavaScript включает множество улучшений:

- классы и модули для лучшей организации кода
- стрелочные функции для лаконичного синтаксиса
- промисы и async/await для удобной работы с асинхронностью
- деструктуризацию и операторы расширения
- типизированные массивы для работы с бинарными данными

JavaScript поддерживает несколько парадигм программирования: объектно-ориентированную, функциональную и императивную. Это делает язык гибким и подходящим для различных стилей разработки. С появлением Node.js JavaScript вышел за пределы браузеров, позволив создавать серверные приложения, утилиты командной строки и даже настольные приложения.

JavaScript продолжает развиваться, ежегодно получая новые возможности, сохраняя при этом обратную совместимость. Его универсальность, богатая экосистема и повсеместная поддержка делают JavaScript одним из самых востребованных языков программирования в мире.

Python представляет собой высокоуровневый интерпретируемый язык программирования общего назначения, который сочетает в себе простоту изучения с мощными возможностями. Созданный с акцентом на читаемость кода, Python использует интуитивно понятный синтаксис с минимальным количеством служебных символов, что делает его идеальным выбором как для начинающих, так и для опытных разработчиков.

Основные преимущества Python проистекают из его философии:

- чистый и выразительный синтаксис с обязательными отступами
- динамическая типизация с сильной проверкой типов
- автоматическое управление памятью
- поддержка нескольких парадигм программирования
- богатая стандартная библиотека

Python демонстрирует исключительную универсальность в различных областях применения:

- веб-разработка с использованием Django, Flask и FastAPI
- научные вычисления и анализ данных через NumPy, Pandas и SciPy
- машинное обучение и искусственный интеллект с TensorFlow и PyTorch
- автоматизация и написание скриптов
- разработка графических интерфейсов с помощью Tkinter и PyQt
- создание сетевых приложений и работа с API

Современные реализации Python (CPython, PyPy, Jython) обеспечивают высокую производительность и кроссплатформенность. Виртуальные окружения и система управления пакетами pip позволяют эффективно работать с зависимостями. Поддержка асинхронного программирования через asyncio делает Python конкурентоспособным для создания высоконагруженных приложений.

Экосистема Python включает более 300 тысяч пакетов в индексе PyPI, охватывающих практически все возможные области применения. Язык активно развивается, сохраняя обратную совместимость и ежегодно добавляя новые

возможности. Его используют такие технологические гиганты как Google, NASA, Instagram и Spotify.

Благодаря сочетанию простоты, мощности и универсальности Python продолжает оставаться одним из самых популярных языков программирования, занимая лидирующие позиции в рейтингах TIOBE, Stack Overflow и GitHub. Его сообщество разработчиков — одно из самых активных и дружелюбных в индустрии, что способствует постоянному совершенствованию языка и его экосистемы.

FastAPI - это современный веб-фреймворк для создания API на Python, который сочетает в себе высокую производительность, простоту разработки и строгую типизацию данных. Построенный на основе Starlette для обработки запросов и Pydantic для валидации данных, этот фреймворк идеально подходит для разработки как небольших сервисов, так и сложных высоконагруженных систем. Одной из ключевых особенностей FastAPI является его асинхронная природа, которая позволяет эффективно обрабатывать множество одновременных подключений без блокировки основного потока выполнения, что особенно важно для современных веб-приложений с высокой нагрузкой. Фреймворк автоматически генерирует подробную интерактивную документацию API в форматах Swagger UI и ReDoc, основываясь на типах данных и аннотациях в коде, что значительно ускоряет процесс разработки и взаимодействия между фронтендом и бэкендом. Встроенная система валидации данных через Pydantic обеспечивает строгую типизацию входных и выходных параметров, автоматически преобразуя данные между Python-объектами и JSON, а также выдавая понятные сообщения об ошибках при несоответствии ожидаемым типам.

FastAPI поддерживает все возможности современного Python, включая асинхронные функции, type hints и dependency injection, что делает код более читаемым, поддерживаемым и менее подверженным ошибкам. Фреймворк имеет модульную архитектуру и легко интегрируется с различными базами данных через SQLAlchemy, Tortoise ORM или асинхронные драйверы, а также

поддерживает аутентификацию через OAuth2, JWT-токены и другие механизмы безопасности. Благодаря использованию стандартов OpenAPI и JSON Schema, API, созданные на FastAPI, легко интегрируются с другими сервисами и клиентскими приложениями. Производительность FastAPI сравнима с Node.js и Go, при этом сохраняя все преимущества экосистемы Python, включая богатую коллекцию библиотек для машинного обучения, анализа данных и научных вычислений. Фреймворк активно развивается, имеет подробную документацию и большое сообщество разработчиков, что делает его надежным выбором для production-решений.

В отличие от традиционных Python-фреймворков, FastAPI изначально проектировался с учетом современных требований к веб-разработке, поддерживая WebSockets, GraphQL, фоновые задачи и другие передовые технологии. Его минималистичный, но выразительный синтаксис позволяет быстро создавать функциональные конечные точки, одновременно обеспечивая высокий уровень безопасности и производительности. FastAPI особенно хорошо подходит для микросервисной архитектуры, облачных решений и систем, где важны скорость отклика и эффективное использование ресурсов. Благодаря своей гибкости, производительности и удобству разработки, FastAPI заслуженно считается одним из лучших фреймворков для создания API на Python, объединяя в себе лучшие практики веб-разработки с мощью и простотой языка Python.

PostgreSQL - это мощная, объектно-реляционная система управления базами данных с открытым исходным кодом, которая сочетает в себе надежность корпоративных решений и гибкость современных технологий. Одним из ключевых преимуществ PostgreSQL является ее строгое соответствие принципам ACID (Atomicity, Consistency, Isolation, Durability), что гарантирует целостность данных даже в условиях высокой нагрузки или системных сбоев. Эта СУБД поддерживает сложные запросы, включая рекурсивные запросы и оконные функции, что делает ее идеальным выбором для аналитических задач и систем с интенсивной обработкой данных. PostgreSQL отличается расширяемой архитектурой — разработчики могут создавать собственные типы данных,

операторы и функции на различных языках программирования, включая PL/pgSQL, Python, JavaScript и даже R. Система предлагает встроенную поддержку JSON и JSONB, что позволяет эффективно работать с полуструктурированными данными, сочетая преимущества реляционных и документоориентированных подходов.

Важной особенностью является развитая система индексов, включая B-деревья, GIN, GiST и BRIN, которые оптимизируют выполнение запросов для различных сценариев использования. PostgreSQL обеспечивает высокий уровень безопасности благодаря механизмам ролевого доступа, шифрованию соединений и детальному управлению правами пользователей. Система поддерживает репликацию (как синхронную, так и асинхронную), что позволяет создавать отказоустойчивые кластерные решения. Для работы с пространственными данными существует расширение PostGIS, превращающее PostgreSQL в полноценную геоинформационную систему.

Производительность PostgreSQL может масштабироваться за счет параллельного выполнения запросов и эффективного использования ресурсов многоядерных процессоров. Сообщество разработчиков постоянно совершенствует систему, добавляя новые функции, такие как декларативное секционирование таблиц, улучшенная обработка временных рядов и поддержка распределенных транзакций. PostgreSQL интегрируется со всеми популярными языками программирования и фреймворками, а его открытая лицензия позволяет свободно использовать систему в коммерческих проектах без ограничений. Благодаря сочетанию надежности, функциональности и производительности, PostgreSQL является оптимальным выбором для сложных корпоративных приложений, веб-сервисов с высокой нагрузкой и систем обработки больших объемов структурированных данных.

Docker представляет собой революционную платформу для контейнеризации приложений, которая коренным образом изменила подход к разработке и развертыванию программного обеспечения. В основе Docker лежит технология контейнеризации, позволяющая запускать изолированные процессы

на уровне операционной системы, что обеспечивает высокую производительность и эффективность использования ресурсов. Ключевым преимуществом Docker является его способность создавать легковесные и самодостаточные контейнеры, содержащие все необходимое для работы приложения. Это устраняет проблему различий между средами выполнения, так как контейнер гарантированно работает одинаково на любой поддерживающей системе.

Docker использует многослойную файловую систему, что позволяет эффективно управлять образами и ускорять процесс развертывания. Одной из мощных возможностей является Docker Compose, который позволяет описывать многоконтейнерные приложения и управлять ими как единым целым. Это особенно полезно для сложных систем, состоящих из нескольких взаимосвязанных сервисов. Интеграция с Kubernetes предоставляет инструменты для оркестрации контейнеров, позволяя автоматизировать развертывание и масштабирование распределенных систем.

Важным аспектом является экосистема Docker, включающая репозиторий готовых образов, где можно найти официальные образы для большинства популярных технологий, что значительно ускоряет процесс разработки. С точки зрения безопасности Docker предоставляет механизмы изоляции контейнеров и контроля ресурсов. Для разработчиков предлагаются удобные инструменты для отладки и мониторинга работы контейнеров.

В производственной среде Docker интегрируется с системами непрерывной интеграции, позволяя автоматизировать весь процесс разработки. Особенностью является кроссплатформенность - он работает на различных операционных системах и поддерживается всеми крупными облачными провайдерами. Для управления конфигурацией Docker предоставляет специальные механизмы. Оптимизация образов достигается за счет использования многоступенчатых сборок и выбора минимальных базовых образов.

Docker поддерживает различные сетевые драйверы, что позволяет гибко настраивать взаимодействие между контейнерами. В современных практиках Docker стал фактическим стандартом, обеспечивающим согласованность сред разработки и production. Его модульная архитектура идеально подходит для микросервисных приложений, где каждый сервис может быть упакован в отдельный контейнер. Благодаря своей гибкости и производительности, Docker остается одной из самых востребованных технологий в современной разработке программного обеспечения.

Alembic представляет собой профессиональную систему управления миграциями, созданную специально для работы с SQLAlchemy. Этот инструмент стал стандартом де-факто для Python-разработчиков, работающих с реляционными базами данных. Основная задача Alembic — обеспечить контролируемое и предсказуемое изменение схемы базы данных по мере развития приложения. В отличие от многих аналогов, Alembic не пытается быть универсальным решением, а предлагает глубокую интеграцию именно с экосистемой SQLAlchemy, что делает его особенно ценным для сложных проектов.

Работа с Alembic строится вокруг концепции явных миграций. Каждое изменение структуры базы данных описывается в виде отдельного Python-скрипта, содержащего функции upgrade и downgrade. Такой подход обеспечивает полную прозрачность изменений и возможность точного контроля над процессом эволюции базы данных. Особенно мощной особенностью является функция автогенерации миграций, которая анализирует различия между моделями SQLAlchemy и текущим состоянием базы данных, значительно сокращая объем ручной работы.

Alembic поддерживает все основные реляционные СУБД, включая PostgreSQL, MySQL, SQLite и Oracle, сохраняя при этом единый интерфейс работы. Инструмент предоставляет развитые механизмы для работы с транзакциями, ветвлением миграций и обработкой данных при изменении схемы. Встроенная система версионирования позволяет точно отслеживать

состояние базы данных на любом окружении, что критически важно для командной разработки и промышленного развертывания приложений.

Pydantic установил новые стандарты работы с данными в Python-экосистеме. Эта библиотека коренным образом изменила подход к валидации и парсингу данных, предложив элегантное решение на основе аннотаций типов. Pydantic V2, с его Rust-ядром, обеспечивает невероятную производительность, делая валидацию данных практически бесплатной с точки зрения вычислительных ресурсов. Библиотека стала неотъемлемой частью современных Python-фреймворков, особенно FastAPI, где используется как стандартная система валидации запросов и ответов.

Основная сила Pydantic заключается в его простоте и выразительности. Модели данных описываются как обычные Python-классы с аннотациями типов, а вся сложная логика валидации скрыта под капотом. Библиотека поддерживает все стандартные типы Python, а также предлагает богатый набор специализированных типов для работы с email-адресами, URL, UUID и другими распространенными форматами данных. Особенно ценным является автоматическое преобразование данных между различными представлениями — например, из JSON в Python-объекты и обратно.

Pydantic нашел применение в самых разных областях — от валидации API-запросов до управления конфигурацией приложений. Его способность автоматически генерировать JSON-схемы делает его идеальным инструментом для разработки веб-API. Встроенная поддержка .env-файлов через класс BaseSettings значительно упрощает работу с конфигурацией. Расширенные возможности, такие как кастомные валидаторы, динамические модели и наследование схем, позволяют решать даже самые сложные задачи работы с данными, сохраняя при этом чистоту и читаемость кода.

Выводы по главе

Разработанная концепция автоматизированной системы оценки backend-разработчиков представляет собой сбалансированное решение, сочетающее

технологические возможности и практические потребности современного IT-рекрутинга. Анализ показал, что система способна обеспечить объективную и стандартизированную оценку профессиональных компетенций за счет комплексного подхода к тестированию. Ключевым преимуществом является устранение субъективного фактора на этапе первичного отбора и создание единых критериев для всех кандидатов.

Особую ценность системе придает ее специализация именно на backend-разработке, что позволяет глубже оценивать специфические навыки, такие как работа с базами данных, проектирование API и понимание серверных технологий. При этом сохраняется гибкость для адаптации под различные технологические стеки и уровни позиций.

Следует отметить, что система не претендует на полную замену традиционных методов оценки, а скорее оптимизирует начальные этапы отбора, позволяя техническим специалистам сосредоточиться на наиболее перспективных кандидатах. Ее внедрение создаст основу для более эффективного распределения ресурсов в процессе подбора персонала.

Перспективы развития системы связаны с возможностями ее интеграции с существующими HR-процессами и платформами, а также с постоянным совершенствованием тестовых методик. Важным направлением является накопление и анализ данных для последующего улучшения критериев оценки и выявления новых тенденций в требованиях к backend-разработчикам.

Глава 2 Проектирование информационной системы

Проектирование информационной системы является ключевым этапом разработки, определяющим её архитектуру, функциональность, производительность и масштабируемость. В данной главе рассматриваются методологические подходы к проектированию ИС, включая анализ требований, выбор технологий, моделирование данных и проектирование пользовательских интерфейсов.

Актуальность проектирования заключается в необходимости создания эффективного, надежного и удобного программного решения, соответствующего потребностям заказчика. Грамотное проектирование позволяет минимизировать риски на этапе реализации, оптимизировать вычислительные ресурсы и обеспечить удобство сопровождения системы в будущем.

Целью главы является разработка архитектуры информационной системы на основе проведенного ранее анализа требований. Для достижения этой цели решаются следующие задачи:

- определение структурных компонентов системы и их взаимодействия.
- выбор методологии проектирования
- разработка концептуальной, логической и физической модели данных.
- проектирование пользовательских интерфейсов и бизнес-логики.
- обоснование выбора технологий

Методы проектирования, используемые в работе, включают:

- визуальное моделирование
- нормализацию базы данных для минимизации избыточности.
- прототипирование интерфейсов для оценки удобства использования.

Результатом проектирования станут:

- детализированные схемы архитектуры системы.
- нормализованная модель базы данных.
- макеты пользовательских интерфейсов.

Полученные проектные решения лягут в основу реализации ИС в следующей главе, обеспечивая согласованность функциональных модулей и эффективность их взаимодействия.

2.1 Архитектура информационной системы

Современные информационные системы представляют собой сложные программно-аппаратные комплексы, эффективность которых определяется не только их функциональностью, но и продуманной архитектурой. Архитектура информационной системы представляет собой фундаментальную организацию системы, воплощенная в её компонентах, их взаимодействии между собой и с внешней средой, а также принципах проектирования и развития. В условиях стремительного роста объемов данных и требований к производительности, грамотно спроектированная архитектура становится критически важным фактором успеха. Она обеспечивает масштабируемость системы при увеличении нагрузки, гарантирует отказоустойчивость и бесперебойную работу, оптимизирует производительность и скорость обработки данных, упрощает поддержку и развитие системы в будущем, а также обеспечивает безопасность данных и соответствие отраслевым стандартам.

На сегодняшний день существует несколько основных подходов к проектированию архитектуры информационных систем, каждый из которых имеет свои преимущества и области применения. Монолитная архитектура, традиционно используемая в корпоративных решениях, отличается простотой разработки и развертывания, но может столкнуться с проблемами масштабируемости. Микросервисная архитектура, получившая широкое распространение в последние годы, обеспечивает высокую гибкость и масштабируемость за счет декомпозиции системы на независимые сервисы. Событийно-ориентированная архитектура оптимальна для систем с интенсивным потоком событий и асинхронной обработкой данных, тогда как сервис-ориентированная архитектура остается востребованной в крупных корпоративных средах с необходимостью интеграции разнородных систем.

Проектирование архитектуры информационной системы начинается с анализа бизнес-требований и определения ключевых функциональных возможностей. На этом этапе важно четко определить границы системы,

основные потоки данных и взаимодействие с внешними системами. Следующим шагом является выбор архитектурного стиля и технологического стека, которые наилучшим образом соответствуют поставленным задачам. Особое внимание уделяется вопросам масштабируемости - система должна быть способна обрабатывать возрастающую нагрузку без снижения производительности. Не менее важны аспекты безопасности, включая защиту данных, аутентификацию и авторизацию пользователей, а также защиту от внешних угроз.

В рамках проектирования архитектуры также рассматриваются вопросы отказоустойчивости и восстановления после сбоев. Современные системы должны быть спроектированы таким образом, чтобы обеспечить непрерывность бизнес-процессов даже в случае частичных отказов оборудования или программного обеспечения. Это достигается за счет репликации данных, кластеризации сервисов и реализации механизмов автоматического восстановления. Важным аспектом является и проектирование системы мониторинга, которая позволяет оперативно выявлять и устранять проблемы в работе системы.

Особую сложность представляет проектирование архитектуры для систем, работающих с большими объемами данных. В таких случаях необходимо тщательно продумывать стратегии хранения, индексирования и обработки данных, чтобы обеспечить приемлемую производительность при минимальных задержках. Использование распределенных вычислений, кэширования и оптимизированных структур данных позволяет решить эти задачи. При этом важно соблюдать баланс между производительностью, стоимостью владения и сложностью реализации.

Интеграционные аспекты архитектуры играют ключевую роль в современных информационных системах, которым необходимо взаимодействовать с множеством внешних сервисов и платформ. Проектирование четких и стандартизированных интерфейсов взаимодействия, использование API-шлюзов и шин предприятия позволяет создать гибкую и

расширяемую систему. Особое внимание уделяется вопросам совместимости и миграции данных, особенно при модернизации унаследованных систем.

2.1.1 Контекстная диаграмма

Контекстная диаграмма представляет собой фундаментальный инструмент системного проектирования, который позволяет определить границы разрабатываемой системы и её взаимодействие с внешней средой. На этой диаграмме система изображается как единый блок (так называемый "чёрный ящик"), вокруг которого располагаются все внешние сущности - пользователи, смежные системы, сторонние сервисы и API. Главная ценность этой диаграммы заключается в том, что она даёт всем участникам проекта - заказчикам, бизнес-аналитикам, разработчикам - единое и непротиворечивое понимание того, что именно входит в разрабатываемую систему, а что остаётся за её пределами.

При создании контекстной диаграммы важно чётко определить:

- кто является основными пользователями системы
- какие внешние системы будут интегрироваться с нашей разработкой
- какие данные и команды будут передаваться через границы системы
- какие протоколы и интерфейсы будут использоваться для взаимодействия

Эта диаграмма особенно полезна на старте проекта, когда необходимо согласовать с заказчиком базовые требования и границы системы. Она помогает избежать распространённых ошибок, когда разные участники проекта по-разному понимают масштабы разработки. Также контекстная диаграмма служит отправной точкой для создания более детальных архитектурных схем на следующих уровнях проектирования.

На рисунке 2.1 изображена контекстная диаграмма проектируемой информационной системы.

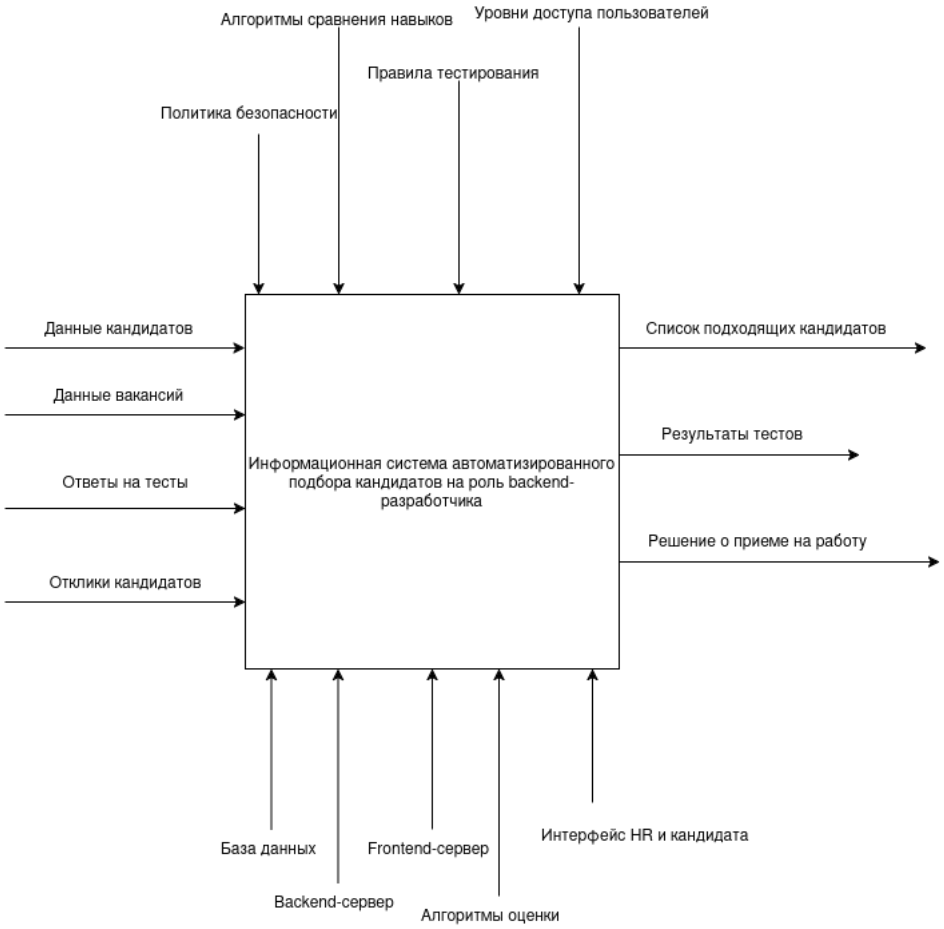

Рисунок 2.1 — Контекстная диаграмма

На диаграмме представлена информационная система автоматизированного подбора кандидатов на роль backend-разработчика. Это комплексное решение, предназначенное для оптимизации процесса поиска и оценки специалистов в области backend-разработки. Система помогает рекрутерам эффективно обрабатывать входящие заявки, отбирая наиболее подходящих кандидатов на основе заданных критериев.

Основу системы составляют четко определенные параметры отбора, которые включают несколько ключевых аспектов. Учитываются технические навыки кандидата. Важную роль играет образовательная составляющая - наличие профильного образования, дополнительных сертификатов и пройденных курсов.

Работа системы построена на последовательном выполнении нескольких этапов. Первоначально происходит сбор данных о кандидатах. Затем

осуществляется автоматизированная фильтрация по заданным критериям, что позволяет отсеять неподходящих кандидатов на раннем этапе. После этого система проводит более детальную оценку оставшихся соискателей, анализируя соответствие их навыков требованиям конкретной вакансии.

Особенностью данной системы является ее адаптивность - параметры отбора могут быть настроены в зависимости от специфики компании или конкретного проекта.

Результатом работы системы становится ранжированный список кандидатов, упорядоченный по степени их соответствия требованиям вакансии. Это позволяет HR-специалистам сосредоточиться на наиболее перспективных соискателях, существенно сокращая время на первичный отбор. При этом система не заменяет полностью человеческий фактор, а служит эффективным инструментом для оптимизации начальных этапов рекрутингового процесса, обеспечивая более объективную и систематизированную оценку профессиональных качеств backend-разработчиков.

2.1.2 Диаграмма компонентов

Диаграмма компонентов - это мощный инструмент для проектирования модульной структуры системы, который позволяет визуализировать основные функциональные блоки и их взаимосвязи. В отличие от контекстной диаграммы, диаграмма компонентов раскрывает её внутреннее устройство, демонстрируя, из каких ключевых модулей состоит система и как они взаимодействуют между собой.

Основными элементами этой диаграммы являются:

- компоненты
- интерфейсы
- зависимости
- порты

При проектировании сложных систем диаграмма компонентов помогает:

- чётко разделить систему на логические модули

- определить интерфейсы между компонентами
- распределить ответственность между командами разработчиков
- спланировать этапы интеграции
- выявить потенциальные узкие места в архитектуре

Особенно важна эта диаграмма при разработке микросервисных архитектур, где каждый компонент может разрабатываться и развёртываться независимо. Она также служит отличным инструментом для документирования системы и обучения новых членов команды.

На рисунке 2.2 представлена диаграмма компонентов информационной системы.

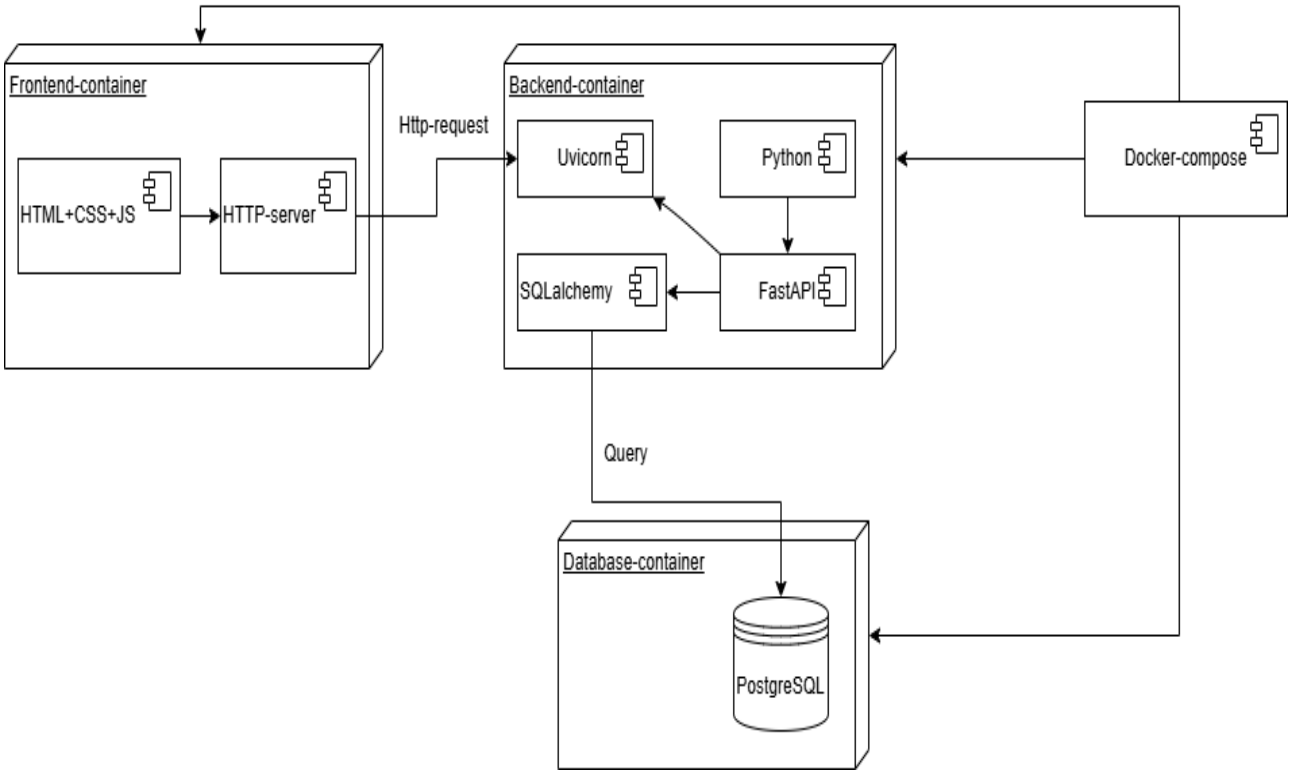

Рисунок 2.2— Диаграмма компонентов

На диаграмме изображена компонентная архитектура веб-приложения, организованная в виде трех взаимодействующих контейнеров, что отражает современный подход к разработке с использованием микросервисов и контейнеризации.

Первым компонентом выступает Frontend-container, который содержит клиентскую часть приложения, включая стандартный набор веб-технологий HTML, CSS и JavaScript, отвечающих за структуру, оформление и интерактивность пользовательского интерфейса соответственно. В этом же контейнере располагается HTTP-сервер выполняющий роль поставщика статического контента для браузера пользователя. Фронтенд взаимодействует с серверной частью посредством HTTP-запросов, что на диаграмме обозначено соответствующей стрелкой.

Серверная логика приложения инкапсулирована в Backend-container, где ключевыми элементами являются Docker-compose для оркестрации контейнеров, язык программирования Python в качестве основной платформы разработки, библиотека SQLalchemy для объектно-реляционного отображения и фреймворк FastAPI, обеспечивающий создание высокопроизводительного API. Бэкенд-контейнер получает запросы от фронтенда, обрабатывает их, выполняет необходимую бизнес-логику и при необходимости формирует запросы к базе данных, что на схеме показано стрелкой с пометкой "Query".

Третьим ключевым компонентом системы является Database-container, где размещена система управления базами данных PostgreSQL, отвечающая за надежное хранение и эффективное управление всеми данными приложения.

Взаимодействие между компонентами происходит по четко определенной схеме: инициация действия пользователем во фронтенде приводит к генерации HTTP-запроса к бэкенду, который после обработки может сформировать SQL-запрос к базе данных, получив необходимые данные, передает их обратно клиентской части для отображения пользователю. Такая архитектура обеспечивает ряд существенных преимуществ, включая модульность, позволяющую изолированно разрабатывать и масштабировать отдельные компоненты, гибкость, обеспечивающую возможность замены технологий в каждом контейнере без перестройки всей системы, и удобство развертывания благодаря использованию Docker-compose, который упрощает процесс запуска и согласованной работы всех частей приложения. Представленная на диаграмме

трехзвенная структура с контейнеризацией компонентов является классическим примером современного подхода к построению веб-приложений, сочетающего в себе надежность, производительность и удобство разработки.

2.1.3 Диаграмма развёртывания

Диаграмма развёртывания представляет собой важный инструмент для проектирования инфраструктурной части системы, показывающий, как программные компоненты будут распределены по физическим или виртуальным узлам. В отличие от предыдущих диаграмм, которые фокусируются на логической структуре системы, диаграмма развёртывания имеет чёткую привязку к серверам, сетевым устройствам, облачным платформам и другим элементам инфраструктуры.

Ключевыми элементами этой диаграммы являются:

- узлы развёртывания
- артефакты
- соединения
- зависимости

Диаграмма развёртывания особенно важна для:

- планирования инфраструктурных требований
- оптимизации распределения ресурсов
- проектирования отказоустойчивых решений
- планирования масштабируемости системы
- документирования окружения для DevOps-команды

В современных условиях, когда системы часто развёртываются в гибридных или мультиоблачных средах, значение этой диаграммы только возрастает. Она помогает наглядно представить географическое распределение компонентов, что критически важно для проектирования систем с требованиями к низкой задержке или локализации данных. Также эта диаграмма служит основой для расчёта инфраструктурных затрат и планирования бюджета проекта.

На рисунке 2.3 представлена диаграмма развёртывания.

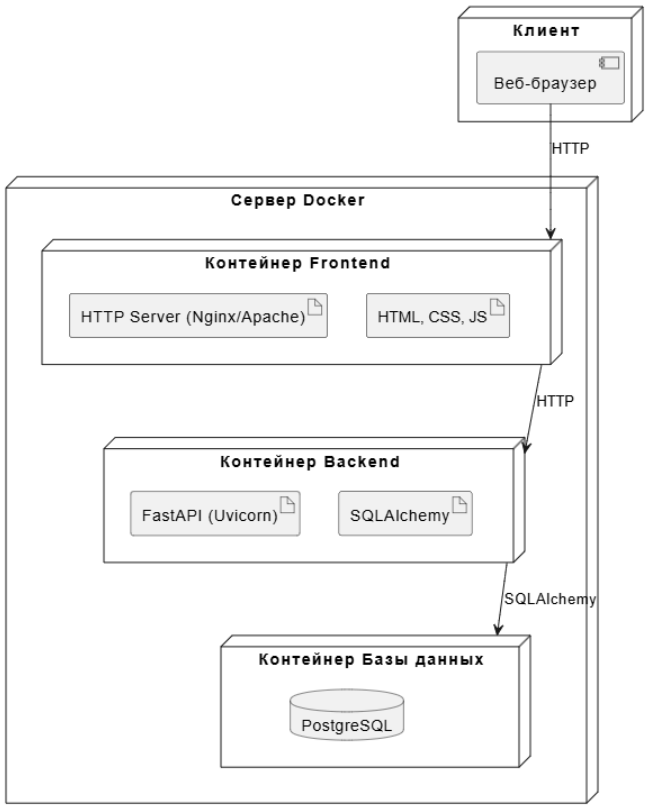

Рисунок 2.1 - Диаграмма развёртывания

Архитектура состоит из трех ключевых компонентов, развернутых на Docker-сервере, которые взаимодействуют между собой для обеспечения работы приложения.

Первый компонент - Frontend-контейнер, который отвечает за клиентскую часть приложения. Внутри него работает HTTP-сервер, обрабатывающий запросы пользователей и отдающий статический контент — файлы HTML, CSS и JavaScript. Этот контейнер обеспечивает визуальную часть приложения, формируя интерфейс, с которым взаимодействуют пользователи. Когда пользователь выполняет какое-либо, фронтенд отправляет HTTP-запросы в Backend-контейнер для обработки.

Второй компонент — Backend-контейнер, где выполняется серверная логика приложения. Он построен на основе FastAPI, который запускается с помощью Uvicorn. Для взаимодействия с базой данных используется SQLAlchemy — ORM-библиотека, позволяющая выполнять запросы к PostgreSQL в виде Python-объектов, а не чистого SQL. Backend принимает HTTP-

запросы от фронтенда, обрабатывает их, выполняет необходимые вычисления и, если требуется, запрашивает или изменяет данные в базе.

Третий компонент — контейнер с базой данных PostgreSQL, где хранится вся структурированная информация приложения Backend взаимодействует с PostgreSQL через SQLAlchemy, который преобразует Python-код в SQL-запросы, отправляемые в базу. После выполнения запроса данные возвращаются в бэкенд, который формирует ответ и передает его обратно во фронтенд для отображения пользователю.

Взаимодействие между компонентами происходит следующим образом:

- пользователь открывает веб-интерфейс, который загружается из Frontend-контейнера.
- при взаимодействии с интерфейсом фронтенд отправляет HTTP-запрос в Backend-контейнер.
- backend обрабатывает запрос и, если нужно, формирует SQL-запрос через SQLAlchemy к PostgreSQL.
- база данных выполняет запрос и возвращает результат в бэкенд.
- backend преобразует данные в удобный формат и отправляет ответ фронтенду.
- frontend получает данные и обновляет интерфейс, показывая пользователю актуальную информацию.

Преимущества такой архитектуры:

- масштабируемость: каждый контейнер можно масштабировать.
- изолированность: проблемы в одном контейнере не влияют напрямую на работу других компонентов.
- удобство развертывания: docker позволяет быстро запускать и обновлять приложение на любом сервере с поддержкой контейнеризации.
- гибкость: технологии в каждом контейнере можно заменять

Таким образом, диаграмма демонстрирует стандартную, но мощную архитектуру веб-приложения, развернутого в Docker, где четко разделены

фронтенд, бэкенд и база данных, что обеспечивает надежность, производительность и удобство разработки.

2.2 Проектирование функциональных требований

Проектирование функциональных требований представляет собой критически важный этап в разработке информационной системы, на котором абстрактные пожелания заказчика и потребности бизнеса трансформируются в четкие, конкретные и технически реализуемые спецификации. Этот процесс выступает своеобразным мостом между бизнес-анализом и непосредственной разработкой, обеспечивая однозначное понимание того, как система должна вести себя в различных условиях и сценариях работы. Основная сложность на этом этапе заключается в том, чтобы не просто зафиксировать желаемую функциональность, но и предугадать все возможные варианты взаимодействия пользователей с системой, предусмотреть обработку исключительных ситуаций и определить границы ответственности системы.

В современной практике разработки программного обеспечения проектирование функциональных требований давно перестало быть простым перечислением возможностей системы. Сегодня это комплексный процесс, включающий в себя несколько взаимосвязанных уровней детализации. На верхнем уровне находятся варианты использования, которые описывают систему с точки зрения конечного пользователя. Затем эти высокоуровневые сценарии детализируются через описание бизнес-процессов, где уже рассматривается последовательность действий, условия выполнения операций и возможные альтернативные потоки. Наиболее глубокий уровень проектирования предполагает описание конкретных взаимодействий между компонентами системы при выполнении тех или иных функций.

Особое значение в процессе проектирования функциональных требований имеет выбор правильных инструментов визуализации. Диаграммы вариантов использования, активности и последовательности в нотации UML стали

фактическим стандартом в этой области, так как позволяют наглядно представить сложные взаимодействия и сделать требования понятными для всех участников проекта - от бизнес-заказчиков до разработчиков и тестировщиков. При этом важно понимать, что создание этих диаграмм - не формальность, а важный аналитический процесс, в ходе которого часто выявляются неочевидные на первый взгляд сценарии использования, противоречия в требованиях или технические ограничения.

Одной из ключевых задач на этапе проектирования функциональных требований является обеспечение полноты и непротиворечивости спецификаций. Это означает, что система должна быть описана таким образом, чтобы для любого возможного сценария работы было четко определено ожидаемое поведение. При этом важно соблюдать баланс между избыточной детализацией, которая может сделать документацию громоздкой и трудноподдерживаемой, и слишком общими формулировками, оставляющими пространство для разночтений. Практика показывает, что хорошо проработанные функциональные требования позволяют значительно сократить количество доработок на поздних этапах проекта и минимизировать риски несоответствия системы ожиданиям заказчика.

Отдельного внимания заслуживает вопрос связи функциональных требований с другими аспектами проектирования системы. С одной стороны, они должны опираться на уже определенную архитектуру системы, с другой - сами влияют на проектирование структуры данных и бизнес-логики. Например, выявленные в процессе работы над Use Cases сложные бизнес-процессы могут потребовать изменений в схеме базы данных или даже повлиять на архитектурные решения. Поэтому проектирование функциональных требований нельзя рассматривать изолированно - это итеративный процесс, требующий постоянной синхронизации с другими направлениями проектирования.

В условиях современных agile-методологий разработки подход к проектированию функциональных требований также претерпел изменения. Вместо создания исчерпывающей документации на старте проекта сейчас чаще

применяется практика постепенной детализации требований, когда сначала определяются ключевые сценарии использования, а затем, по мере реализации, уточняются детали. Однако это не отменяет необходимости тщательного проектирования, а лишь меняет подход к организации этого процесса, делая его более гибким и адаптивным к изменениям.

2.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой важнейший инструмент проектирования программных систем, который позволяет наглядно отобразить функциональные возможности системы с точки зрения конечного пользователя. Эта модель входит в состав унифицированного языка моделирования UML и служит ключевым связующим звеном между бизнес-требованиями и технической реализацией проекта.

Основная ценность данной диаграммы заключается в ее способности простым и понятным языком показать, какие именно функции будет выполнять система, кто именно будет этими функциями пользоваться и с какими внешними компонентами потребуется взаимодействие. При этом диаграмма фокусируется не на технических деталях реализации, а на пользовательском восприятии системы, что делает ее идеальным инструментом для обсуждения с заказчиками и бизнес-пользователями.

В основе диаграммы вариантов использования лежат два ключевых элемента: акторы и собственно варианты использования. Акторы представляют роли пользователей или внешних систем, взаимодействующих с разрабатываемым решением. Варианты использования описывают отдельные функции или возможности системы. Связи между этими элементами показывают, какие именно функции доступны тем или иным категориям пользователей.

Особенность этой модели заключается в ее многоуровневости. Она позволяет одновременно отображать как общий функциональный охват системы, так и детализировать отдельные аспекты через механизмы расширений

и включений. Благодаря этому диаграмма служит отличной отправной точкой для последующей детализации требований в более сложных моделях.

Разработка качественной диаграммы вариантов использования представляет собой глубокий аналитический процесс. В ходе ее создания часто выявляются скрытые сценарии использования, обнаруживаются пробелы в требованиях или находятся новые возможности для улучшения пользовательского опыта. Грамотно построенная диаграмма существенно снижает риск появления серьезных недочетов на поздних этапах разработки.

Важно понимать, что эта модель не является статичным документом. По мере развития проекта и уточнения требований диаграмма вариантов использования может и должна пересматриваться и дополняться. Она служит живым инструментом коммуникации между всеми участниками проекта на протяжении всего жизненного цикла разработки системы

На рисунке 2.4 представлена диаграмма вариантов использования.

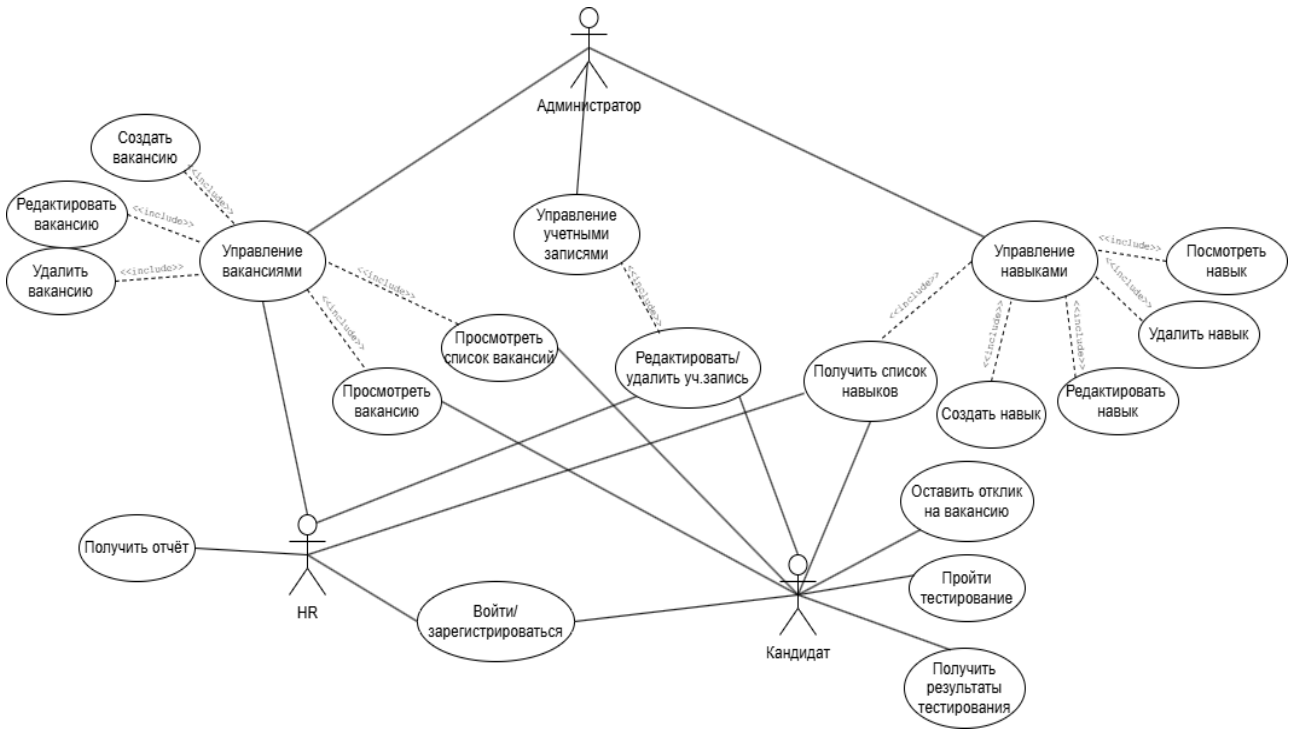

Рисунок 2.4 — Диаграмма вариантов использования

В таблице 2.1 приведено подробное описание прецедентов системы.

Таблица 2.1 – Описание прецедентов информационной системы

Название прецедента	Актор	Описание	Связанные прецеденты
Управление учетными записями	Администратор	Администратор управляет учетными записями пользователей	Редактировать/удалить уч. запись
Редактировать/удалить уч. запись	Администратор	Изменение или удаление данных учетной записи	-
Управление навыками	Администратор	Управление списком навыков в системе	Создать, Редактировать, Удалить, Посмотреть навык
Создать навык	Администратор	Добавление нового навыка	-
Редактировать навык	Администратор	Изменение существующего навыка	-
Удалить навык	Администратор	Удаление навыка из системы	-
Посмотреть навык	Администратор	Просмотр информации о навыке	-
Получить список навыков	Кандидат	Получение перечня доступных навыков	-
Управление вакансиями	HR, Администратор	Управление вакансиями	Создать, Редактировать, Удалить, Просмотреть вакансию
Создать вакансию	HR, Администратор	Добавление новой вакансии	-
Редактировать вакансию	HR, Администратор	Изменение информации о вакансии	-
Удалить вакансию	HR, Администратор	Удаление вакансии	-
Просмотреть вакансию	HR, Кандидат	Просмотр подробностей по вакансии	-
Просмотреть список вакансий	HR, Кандидат	Получение списка всех доступных вакансий	-

Оставить отклик на вакансию	Кандидат	Отклик на подходящую вакансию	-
Пройти тестирование	Кандидат	Прохождение тестов, связанных с навыками	-
Получить результаты тестирования	Кандидат	Получение оценок по пройденным тестам	-
Получить отчёт	HR	Формирование отчета по кандидатам или тестам	-
Войти/зарегистрироваться	Кандидат	Аутентификация и регистрация в системе	-

Диаграмма вариантов использования описывает функциональные возможности системы управления вакансиями и навыками, демонстрируя взаимодействие трех основных ролей: Администратора, HR-специалиста и Кандидата. Каждая из этих ролей имеет свой набор действий, которые определяют их взаимодействие с системой и охватывают ключевые бизнес-процессы, связанные с подбором персонала и управлением данными.

Администратор является главным управляющим системы и обладает наиболее широкими полномочиями. В его обязанности входит полный контроль над вакансиями, включая их создание, редактирование и удаление. Он может просматривать список всех доступных вакансий, изучать детали каждой из них, а также формировать отчеты для анализа данных. Эти функции позволяют Администратору эффективно управлять процессом найма, оперативно вносить изменения в требования к должностям и отслеживать статистику по открытым позициям.

HR-специалист работает с учетными записями пользователей, обеспечивая их регистрацию и авторизацию в системе. Он имеет возможность редактировать и удалять учетные записи, что позволяет поддерживать актуальность данных и контролировать доступ сотрудников и кандидатов. Эта роль критически важна

для обеспечения безопасности системы и корректного распределения прав среди пользователей.

Кандидат взаимодействует с системой в рамках поиска работы и управления своими профессиональными навыками. Он может просматривать список своих навыков, редактировать их, добавлять новые или удалять устаревшие. Кроме того, Кандидат имеет возможность оставлять отклики на заинтересовавшие его вакансии, проходить тестирование, предназначенное для оценки его квалификации, и получать результаты этих тестов. Эти функции помогают Кандидату активно участвовать в процессе трудоустройства, демонстрировать свои компетенции и получать обратную связь от работодателей.

2.2.2 Диаграмма активности

Диаграмма активностей представляет собой мощный инструмент визуализации бизнес-процессов и рабочих потоков в рамках проектирования информационных систем. Эта модель, являющаяся частью стандарта UML, позволяет детально описать последовательность действий, выполняемых системой или пользователями для достижения конкретных целей. В отличие от диаграммы вариантов использования, которая показывает что делает система, диаграмма активностей отвечает на вопрос как именно происходят те или иные процессы.

Основное назначение диаграммы активностей заключается в точном и наглядном представлении алгоритмов работы системы. Она особенно полезна при моделировании сложных бизнес-процессов, где важно отобразить не только основную последовательность шагов, но и возможные ветвления, параллельные потоки выполнения, условия перехода между действиями и точки синхронизации. С помощью этой диаграммы можно эффективно документировать как автоматизированные процессы системы, так и взаимодействие между системой и пользователями.

Ключевыми элементами диаграммы активностей являются действия, переходы, узлы принятия решений, разделения и слияния потоков. Действия представляют отдельные шаги процесса, узлы принятия решений показывают альтернативные пути выполнения, а элементы разделения и слияния позволяют отображать параллельные потоки работ. Такая структура делает диаграмму активностей чрезвычайно гибким инструментом, способным моделировать процессы любой сложности.

Особую ценность диаграмма активностей приобретает при анализе и оптимизации существующих бизнес-процессов. Она позволяет выявить избыточные шаги, узкие места и потенциальные точки улучшения. В контексте разработки программного обеспечения эта диаграмма служит важным связующим звеном между бизнес-аналитиками, которые формулируют требования, и разработчиками, которые реализуют соответствующие алгоритмы.

Применение диаграмм активностей особенно эффективно в проектах, связанных с автоматизацией сложных бизнес-процессов, таких как обработка заказов, цепочки согласований или многоэтапные расчеты. Они помогают всем участникам проекта достичь общего понимания того, как должна работать система, и служат надежной основой для последующего проектирования и тестирования.

На рисунке 2.5 представлена диаграмма активности авторизации.

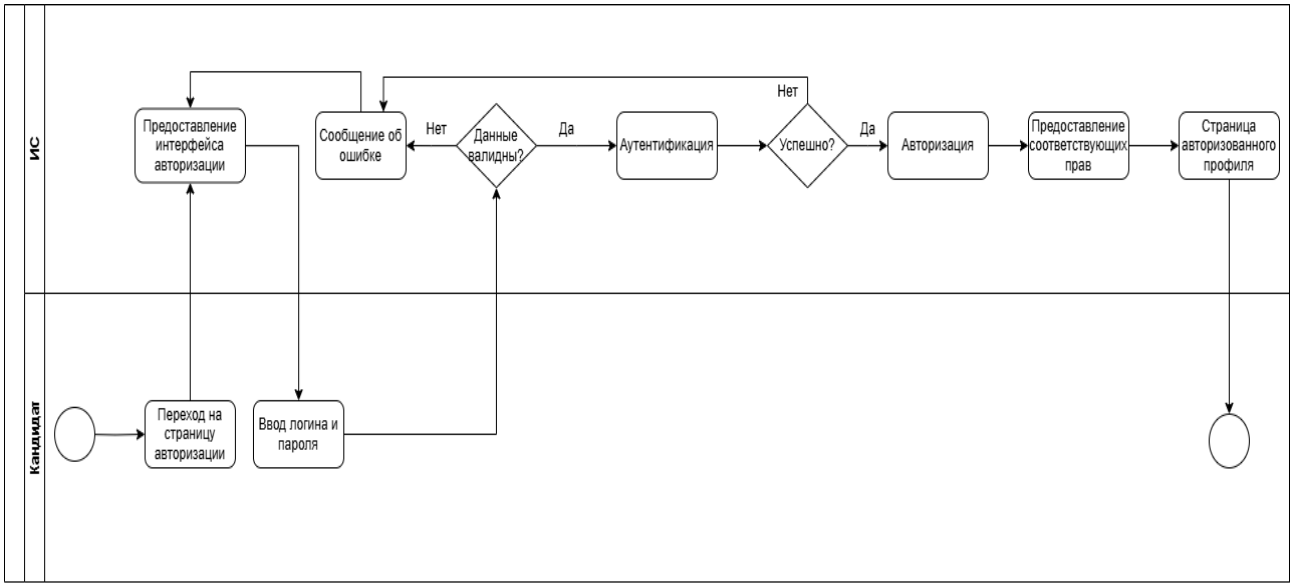

Рисунок 2.5 — Диаграмма активности авторизации

На диаграмме изображён процесс авторизации кандидата в информационной системе. Диаграмма разделена на два пула: кандидат и информационная система.

Процесс начинается с перехода кандидата на страницу авторизации. Далее пользователь вводит логин и пароль. Система в ответ предоставляет интерфейс авторизации и получает введённые данные. Затем выполняется проверка на валидность данных. Если данные невалидны, пользователю отображается сообщение об ошибке, и он может ввести данные повторно.

Если данные валидны, система выполняет аутентификацию, то есть проверку правильности логина и пароля. После этого происходит проверка успешности аутентификации. В случае неудачи пользователю снова выводится сообщение об ошибке.

Если аутентификация проходит успешно, осуществляется авторизация — проверка прав доступа пользователя. После авторизации система предоставляет соответствующие права и отображает страницу авторизованного профиля пользователя.

Процесс завершается переходом пользователя на страницу авторизованного профиля.

На рисунке 2.6 изображена диаграмма активности, описывающая процесс отклика кандидата.

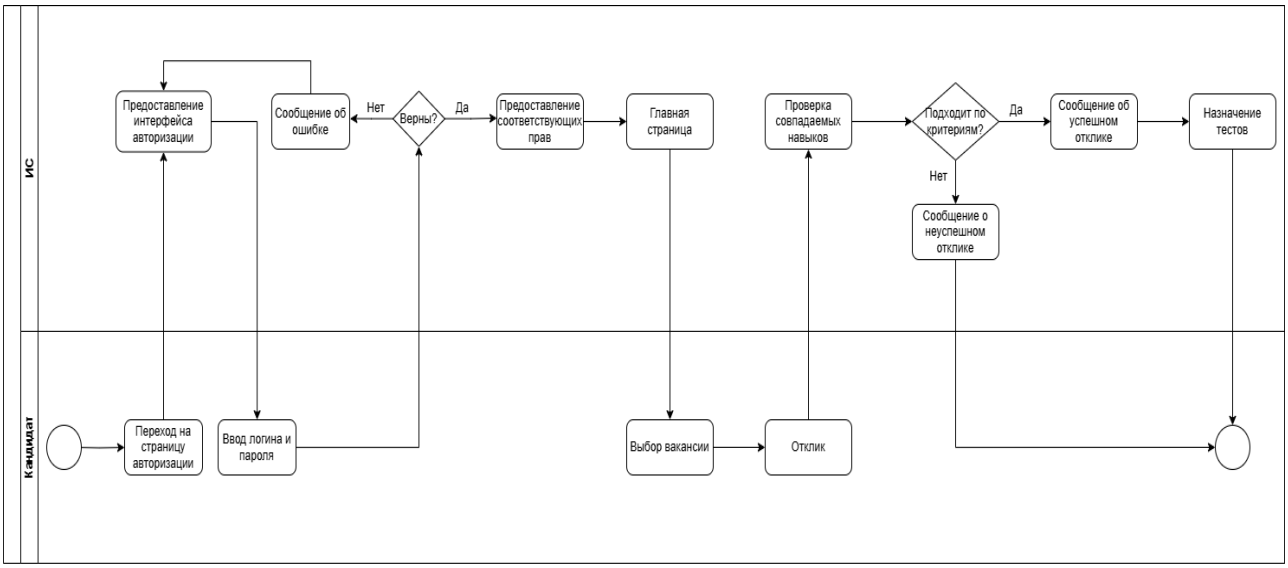

Рисунок 2.6 — Диаграмма активности процесса отклика кандидата

На представленной диаграмме изображён процесс отклика кандидата на вакансию в рамках информационной системы подбора персонала. Диаграмма выполнена в нотации UML и представляет собой диаграмму активности с разделением на две дорожки: "Кандидат" и "ИС" (информационная система). Ниже приведено подробное описание каждого этапа процесса.

Процесс начинается с действия кандидата. Кандидат переходит на страницу авторизации. Далее он вводит логин и пароль. После этого управление передаётся информационной системе, которая предоставляет интерфейс авторизации.

Система проверяет корректность введённых данных. Если данные неверны, пользователю отображается сообщение об ошибке, и он может повторить ввод данных. Если данные верны, система предоставляет соответствующие права и перенаправляет пользователя на главную страницу.

После входа на главную страницу кандидат выбирает интересующую его вакансию. Затем он нажимает кнопку отклика, инициируя процесс отклика на вакансию.

На этом этапе система проверяет совпадающие навыки кандидата с требованиями вакансии. Если навыки кандидата соответствуют критериям вакансии, система отправляет сообщение об успешном отклике и назначает тесты для дальнейшего этапа подбора. После этого процесс завершается.

Если навыки кандидата не соответствуют критериям вакансии, система отправляет сообщение о неуспешном отклике, и процесс также завершается.

Таким образом, диаграмма отражает ключевые этапы взаимодействия кандидата с системой при отклике на вакансию: авторизация, выбор вакансии, проверка соответствия и завершение в зависимости от результата проверки.

На рисунке 2.7 изображена диаграмма активности, описывающая процесс получения отчётности.

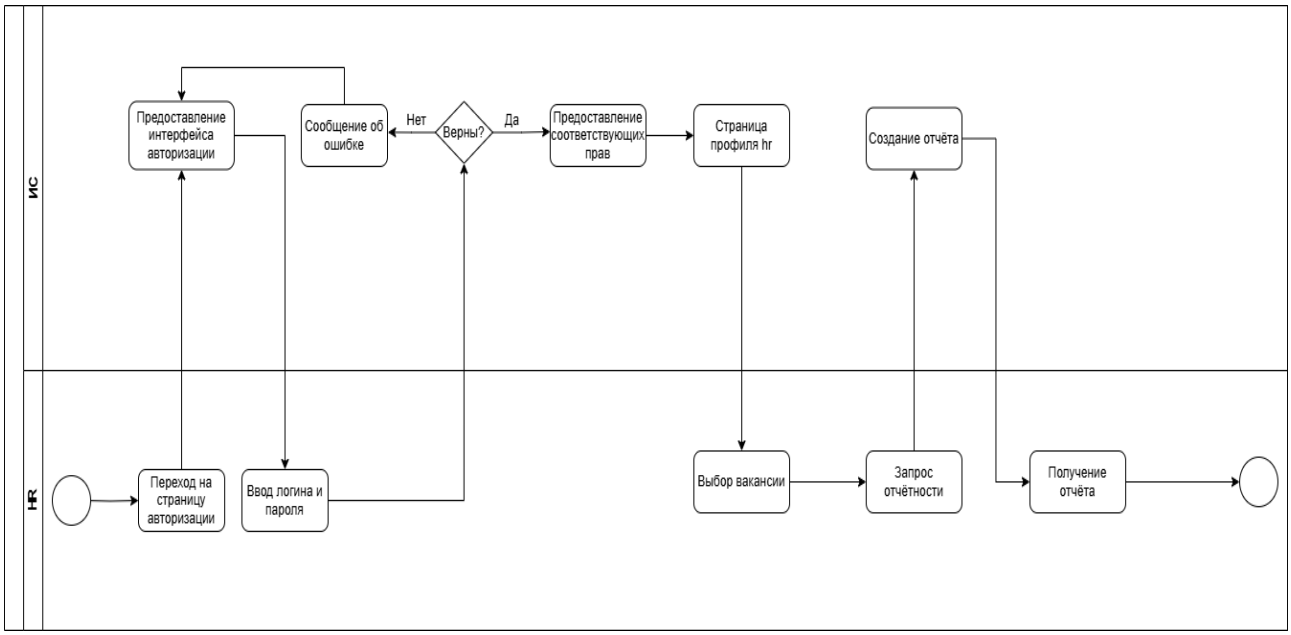

Рисунок 2.7 — Диаграмма активности процесса получения отчётности

На данной диаграмме изображён процесс получения отчётности HR-специалистом в информационной системе подбора персонала. Диаграмма представлена в нотации UML и разделена на две дорожки: "HR" и "ИС".

Процесс начинается с действия HR-специалиста, который переходит на страницу авторизации. Далее он вводит логин и пароль. Информационная система в ответ предоставляет интерфейс авторизации.

После этого система проверяет корректность введённых данных. Если данные неверны, отображается сообщение об ошибке, и пользователь может повторить попытку входа. Если данные верны, система предоставляет соответствующие права и переходит к отображению страницы профиля HR.

На странице профиля HR может выбрать вакансию, по которой он хочет получить отчётность. Далее инициируется запрос отчётности. Этот запрос передаётся в систему, которая в ответ формирует соответствующий отчёт. После создания отчёта система предоставляет его HR-специалисту, и он получает отчёт для дальнейшего анализа.

Таким образом, диаграмма описывает последовательность действий, связанных с авторизацией пользователя, навигацией по профилю, выбором вакансии и получением отчётных данных, с чётким разделением функций между пользователем и информационной системой

На рисунке 2.8 представлена диаграмма активности, описывающая процесс редактирования вакансии.

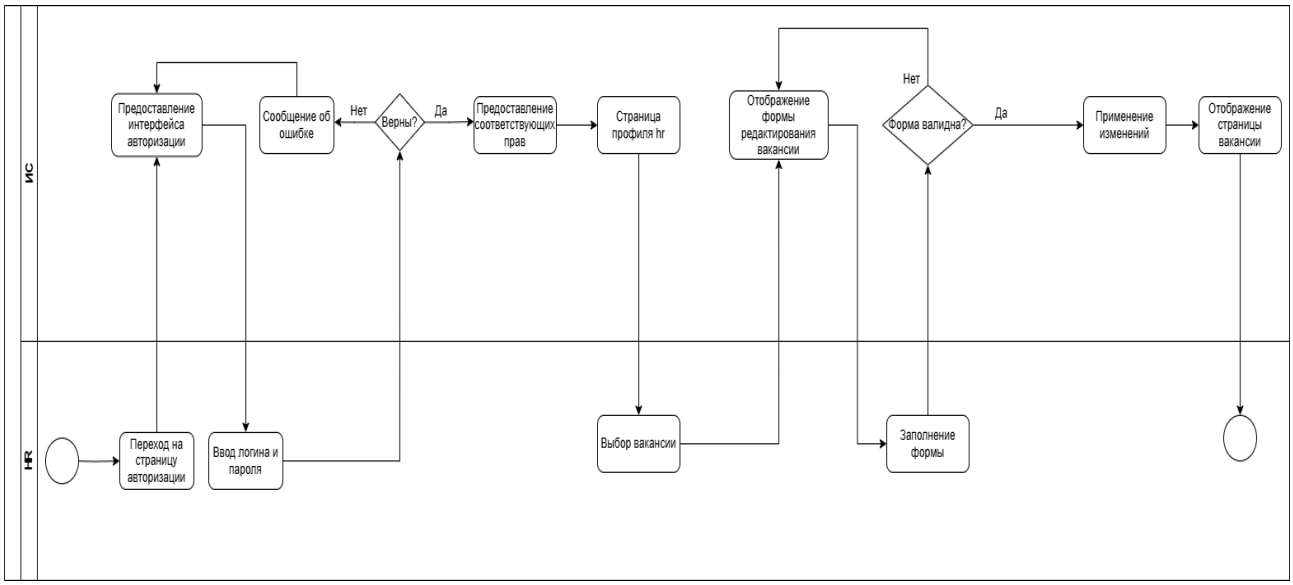

Рисунок 2.8 — Диаграмма активности процесса редактирования вакансии

На данной диаграмме изображён процесс редактирования вакансии HR-специалистом в информационной системе подбора персонала. Диаграмма выполнена в нотации UML и состоит из двух дорожек: "HR" и "ИС".

Процесс начинается с действия HR-специалиста, который переходит на страницу авторизации. Затем он вводит логин и пароль. Информационная система, в ответ на это действие, предоставляет интерфейс авторизации.

Система проверяет корректность введённых данных. Если они некорректны, пользователю отображается сообщение об ошибке, и он может ввести логин и пароль заново. Если данные корректны, система предоставляет пользователю соответствующие права и загружает страницу профиля HR.

На странице профиля HR-специалист выбирает вакансию, которую хочет отредактировать. После этого система отображает форму редактирования вакансии.

HR-специалист заполняет форму с новыми данными. Система проверяет валидность заполненной формы. Если форма невалидна, пользователь возвращается к этапу заполнения формы.

Если форма валидна, система применяет изменения, после чего отображает обновлённую страницу вакансии. На этом процесс редактирования завершается.

Таким образом, диаграмма описывает стандартный сценарий авторизации пользователя, перехода к редактированию информации о вакансии, внесения изменений и проверки корректности данных, с чётким распределением обязанностей между HR и информационной системой.

На рисунке 2.9 представлена диаграмма активности, описывающая процесс создания вакансии.

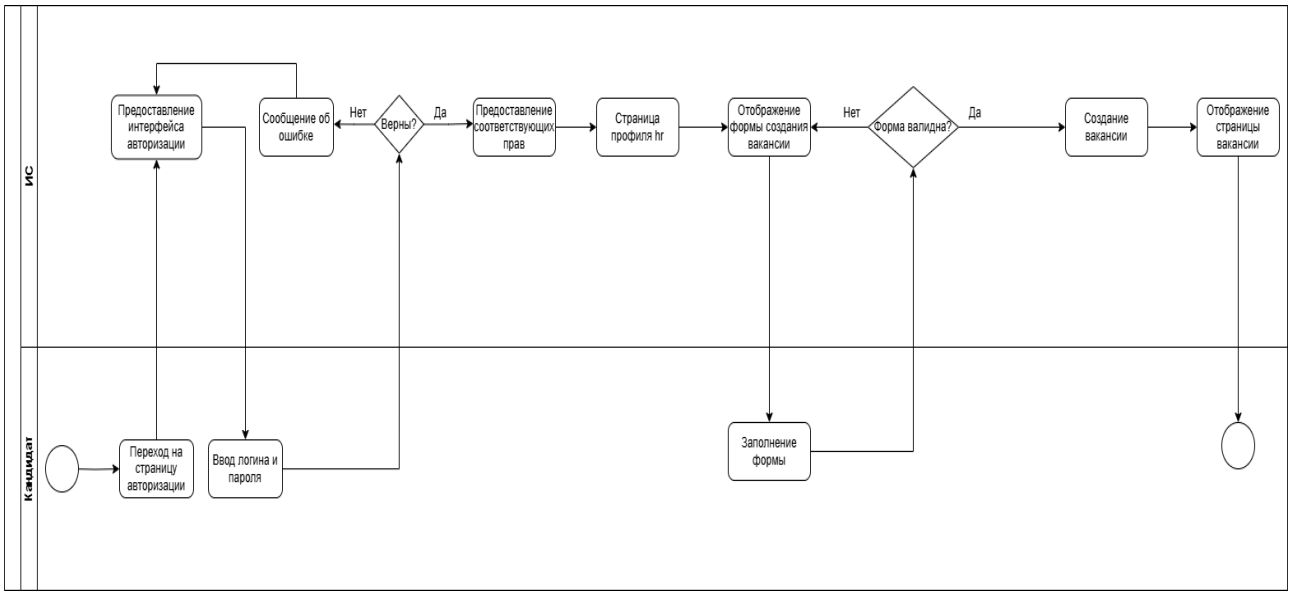

Рисунок 2.9 — Диаграмма активности процесса редактирования вакансии

На диаграмме изображён процесс создания вакансии в информационной системе со стороны кандидата и информационной системы. Это диаграмма активности, представленная в нотации UML с использованием разделения по ролям, где участвуют два участника: "Кандидат" и "ИС".

Процесс начинается с перехода кандидата на страницу авторизации. Далее кандидат вводит логин и пароль. Эти данные передаются в ИС, которая предоставляет интерфейс авторизации. После ввода данных ИС проверяет их корректность.

Если введённые данные неверны, система возвращает сообщение об ошибке, после чего кандидат может попробовать снова.

Если данные верны, ИС предоставляет соответствующие права и переходит к отображению страницы профиля HR.

На следующем этапе отображается форма создания вакансии. Кандидат заполняет эту форму.

После заполнения форма проверяется на валидность. Если форма заполнена некорректно, осуществляется возврат на этап заполнения формы.

Если форма валидна, система создаёт вакансию и отображает страницу с информацией о созданной вакансии.

Процесс завершается.

Таким образом, диаграмма описывает последовательность действий и проверок, необходимых для авторизации пользователя и последующего создания новой вакансии в системе.

На рисунке 2.10 представлена диаграмма активности, описывающая процесс создания вакансии.

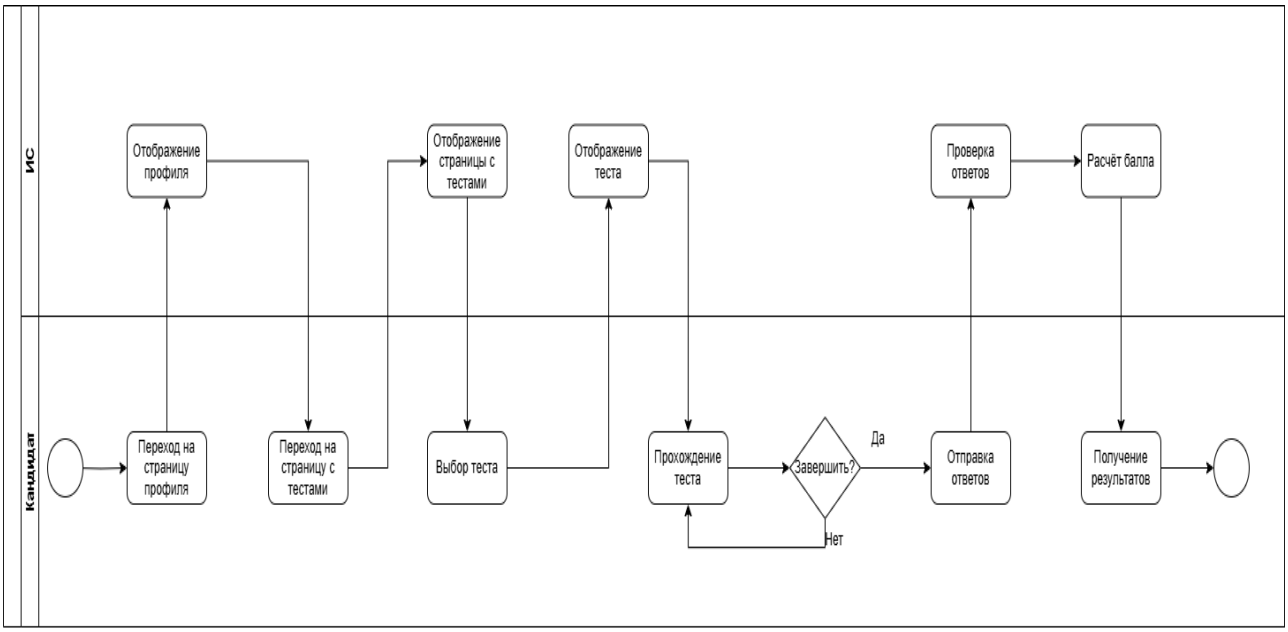

Рисунок 2.10 — Диаграмма активности процесса редактирования вакансии

На представленной диаграмме активности изображён процесс прохождения тестирования кандидатом в информационной системе (ИС). Диаграмма разделена на две дорожки: одна принадлежит участнику "Кандидат", другая — "ИС".

Процесс начинается с перехода кандидата на страницу своего профиля. После этого информационная система отображает страницу профиля.

Затем кандидат переходит на страницу с тестами, и ИС отображает соответствующий интерфейс.

Кандидат выбирает тест из предложенного списка, после чего система отображает сам тест.

Начинается этап прохождения теста. В процессе кандидат может либо продолжать отвечать, либо завершить тест. На этом этапе осуществляется проверка условия: "Завершить?".

Если кандидат выбирает не завершать тест, процесс возвращается к прохождению теста.

Если тест завершён, кандидат отправляет свои ответы. Информационная система принимает ответы и выполняет их проверку.

После проверки ответов система производит расчёт балла и отображает кандидату полученные результаты.

На этом процесс завершается.

Таким образом, диаграмма описывает полный цикл взаимодействия кандидата с системой при прохождении теста: от перехода к тестам до получения результата с оценкой.

2.2.3 Диаграмма последовательности

Диаграмма последовательностей занимает особое место среди инструментов проектирования программных систем, позволяя детально визуализировать динамику взаимодействия между объектами в рамках конкретного сценария работы системы. Этот тип диаграмм UML фокусируется на временной оси, четко показывая порядок и характер обмена сообщениями между участниками взаимодействия. В отличие от диаграммы активностей, описывающей общий поток работ, диаграмма последовательностей раскрывает тонкости коммуникации между конкретными компонентами системы в процессе выполнения отдельного сценария.

Основная ценность диаграммы последовательностей заключается в ее способности наглядно представить временные характеристики взаимодействий, что особенно важно при проектировании сложных распределенных систем. Она позволяет увидеть, какие именно сообщения передаются между объектами, в каком порядке это происходит, какие объекты активны в конкретные моменты времени, и как долго длятся те или иные операции. Такой уровень детализации критически важен для понимания поведения системы в реальных условиях эксплуатации.

Ключевыми элементами диаграммы последовательностей являются объекты-участники взаимодействия, их линии жизни, активационные полосы, показывающие периоды активности объектов, и собственно сообщения между объектами. Особое внимание уделяется различным типам сообщений - синхронным вызовам, асинхронным сигналам, возвратным сообщениям и созданию новых объектов. Все это позволяет с высокой точностью смоделировать реальное поведение системы.

Диаграмма последовательностей особенно полезна при проектировании сложных взаимодействий в объектно-ориентированных системах, где важно понимать не только что делает система, но и как именно распределяется ответственность между различными компонентами. Она помогает выявить потенциальные проблемы проектирования, такие как избыточная связанность компонентов, неоптимальное распределение обязанностей между объектами или возможные узкие места в производительности.

В практике современной разработки программного обеспечения диаграммы последовательностей играют важную роль на этапах как проектирования, так и сопровождения системы. Они служат отличным инструментом коммуникации между архитекторами, разработчиками и тестировщиками, обеспечивая единое понимание сложных механизмов взаимодействия в системе. Особенно незаменимы эти диаграммы при работе над распределенными системами, микросервисными архитектурами и сложными

протоколами взаимодействия, где временные характеристики и порядок обмена сообщениями имеют принципиальное значение.

На рисунке 2.11 представлена диаграмма последовательности, описывающая процесс авторизации пользователя.

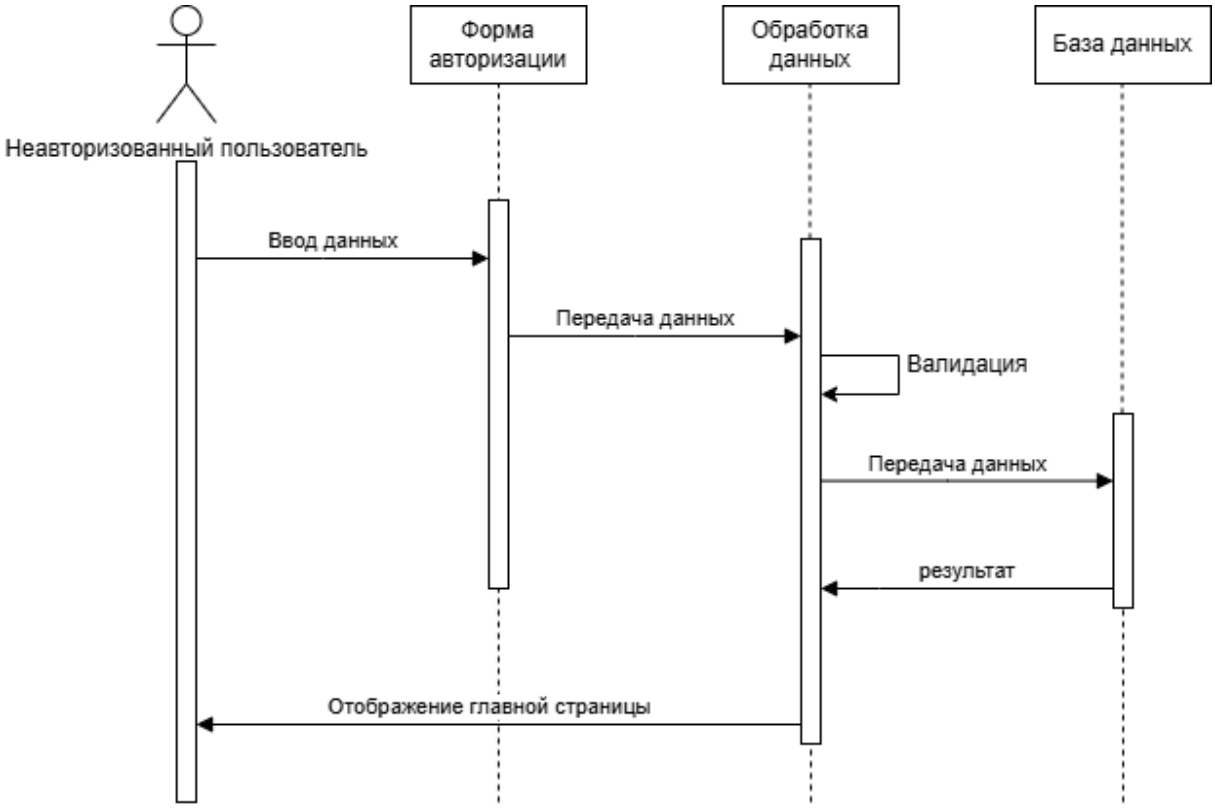

Рисунок 2.11 — Диаграмма последовательности авторизации

На представленной диаграмме изображён процесс авторизации пользователя.

Когда неавторизованный пользователь инициирует вход в систему, он взаимодействует с интерфейсом формы авторизации, где вводит свои учетные данные — обычно логин и пароль. Система на клиентской стороне может выполнять первичную проверку введенных данных на соответствие базовым требованиям. Затем данные передаются на серверную часть, где модуль обработки проводит более глубокую валидацию. После успешной валидации система формирует запрос к базе данных, где происходит сравнение предоставленных учетных данных с хранящейся информацией. В случае успешного сопоставления система создает сессию пользователя, генерирует токен аутентификации и перенаправляет пользователя на главную страницу. При

неудачной попытке авторизации система фиксирует попытку входа и возвращает пользователю сообщение об ошибке, не раскрывая при этом, какая именно часть учетных данных неверна, чтобы не облегчать задачу потенциальным злоумышленникам.

На рисунке 2.12 представлена диаграмма последовательности, описывающая процесс регистрации пользователя.

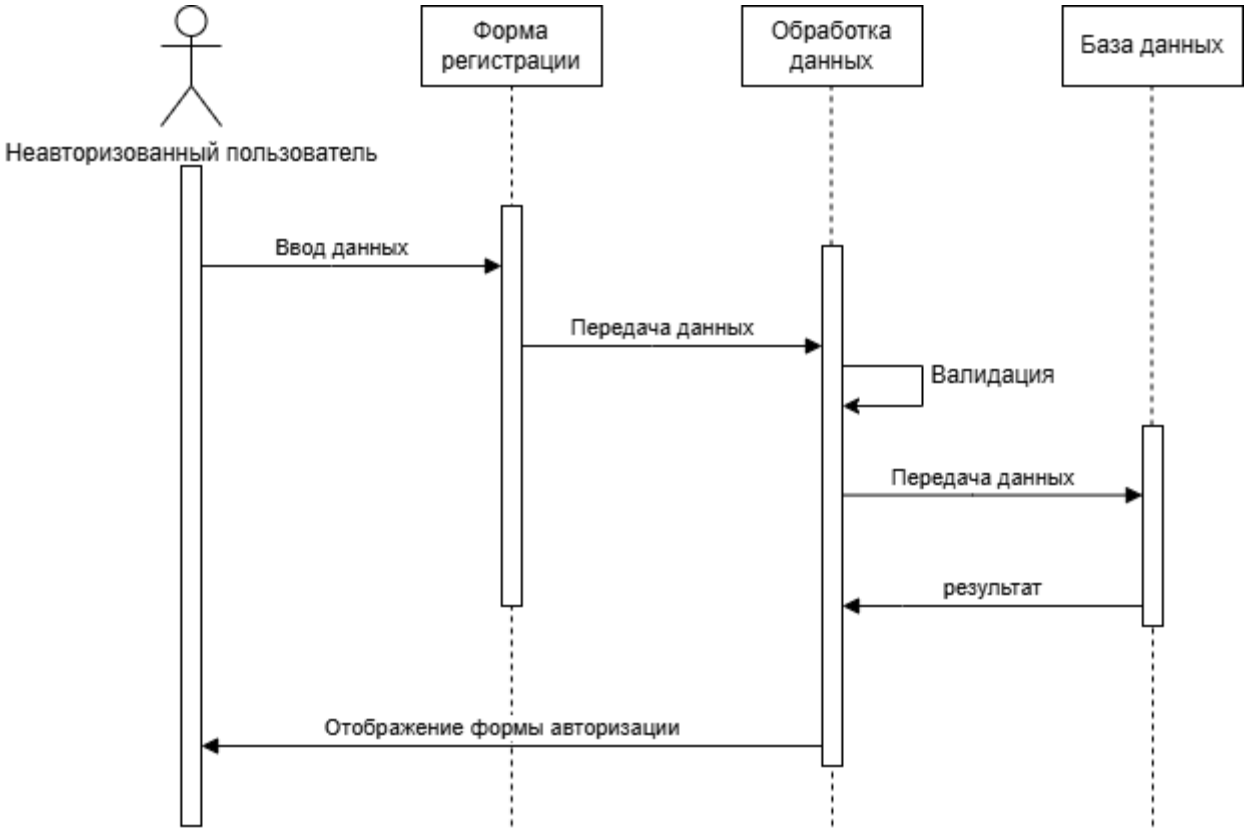

Рисунок 2.12 — Диаграмма последовательности регистрации

На представленной диаграмме изображён процесс регистрации пользователя.

Когда новый пользователь заполняет форму регистрации, система выполняет многоэтапную проверку вводимых данных. На клиентской стороне может осуществляться базовая валидация в реальном времени. После отправки формы данные поступают на сервер, где проводится комплексная проверка, соответствие пароля политикам безопасности, проверку на наличие запрещенных символов или слов. После успешной проверки система хэширует пароль и сохраняет данные пользователя в базе данных. При этом часто

реализуется механизм подтверждения email - на указанный адрес отправляется письмо с ссылкой активации. Только после подтверждения email учетная запись становится полностью активной. В случае ошибок регистрации пользователь получает детализированные сообщения, помогающие исправить введенные данные.

На рисунке 2.13 представлена диаграмма последовательности, описывающая процесс редактирования вакансии.

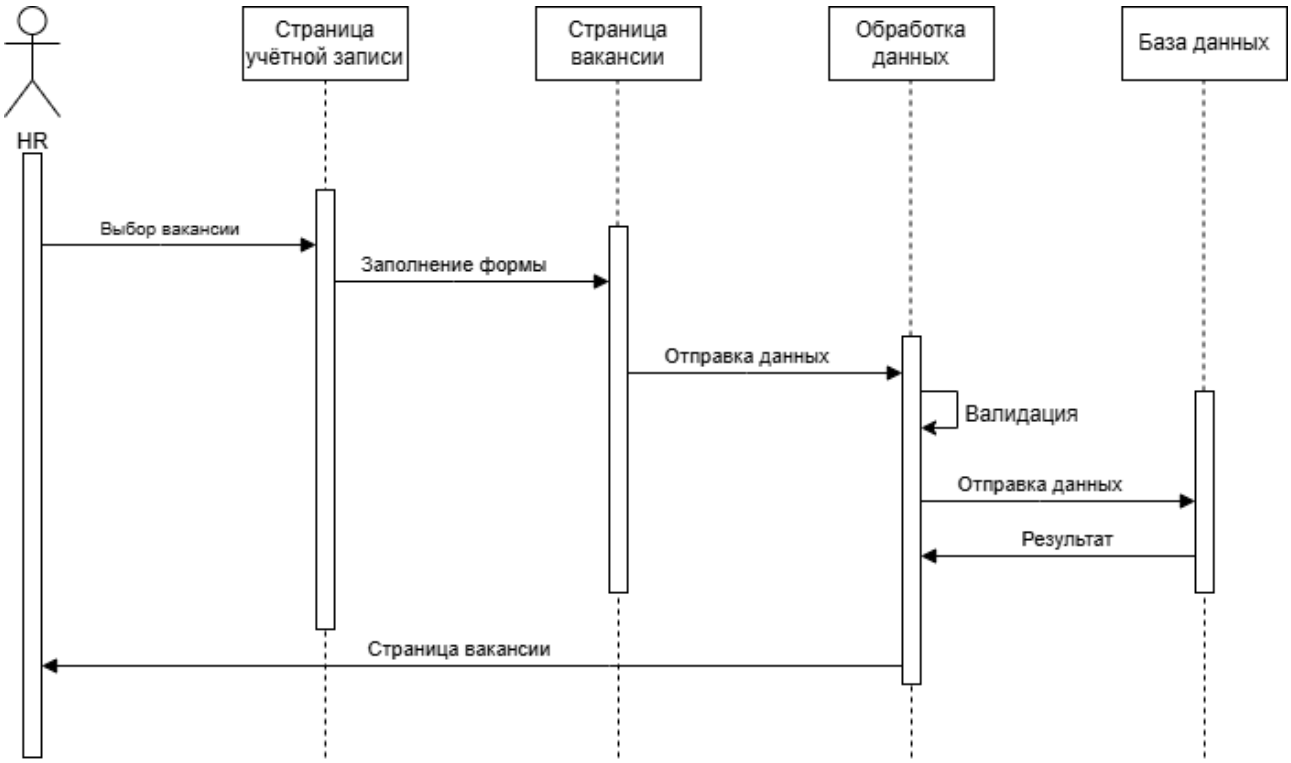

Рисунок 2.13 — Диаграмма последовательности редактирования вакансии

На представленной диаграмме изображён процесс редактирования вакансии.

Когда пользователь выбирает вакансию для редактирования, система сначала проверяет его права на выполнение этой операции. После загрузки текущих данных вакансии пользователь может вносить изменения в различных полях (название, описание, требования и т.д.). При сохранении изменений система выполняет комплексную валидацию: проверяет заполнение обязательных полей, корректность форматов данных (например, дат), отсутствие запрещенного контента. Особое внимание уделяется сохранению истории изменений - система может фиксировать, кто и когда вносил правки, что важно

для аудита. После успешного обновления данных в базе система может отправлять уведомления подписанным кандидатам о том, что вакансия была обновлена. В случае массового редактирования вакансий (например, изменения условий работы для целой категории позиций) система может предоставлять пакетный режим редактирования с дополнительными проверками на согласованность изменений.

На рисунке 2.14 представлена диаграмма последовательности, описывающая процесс создания вакансии.

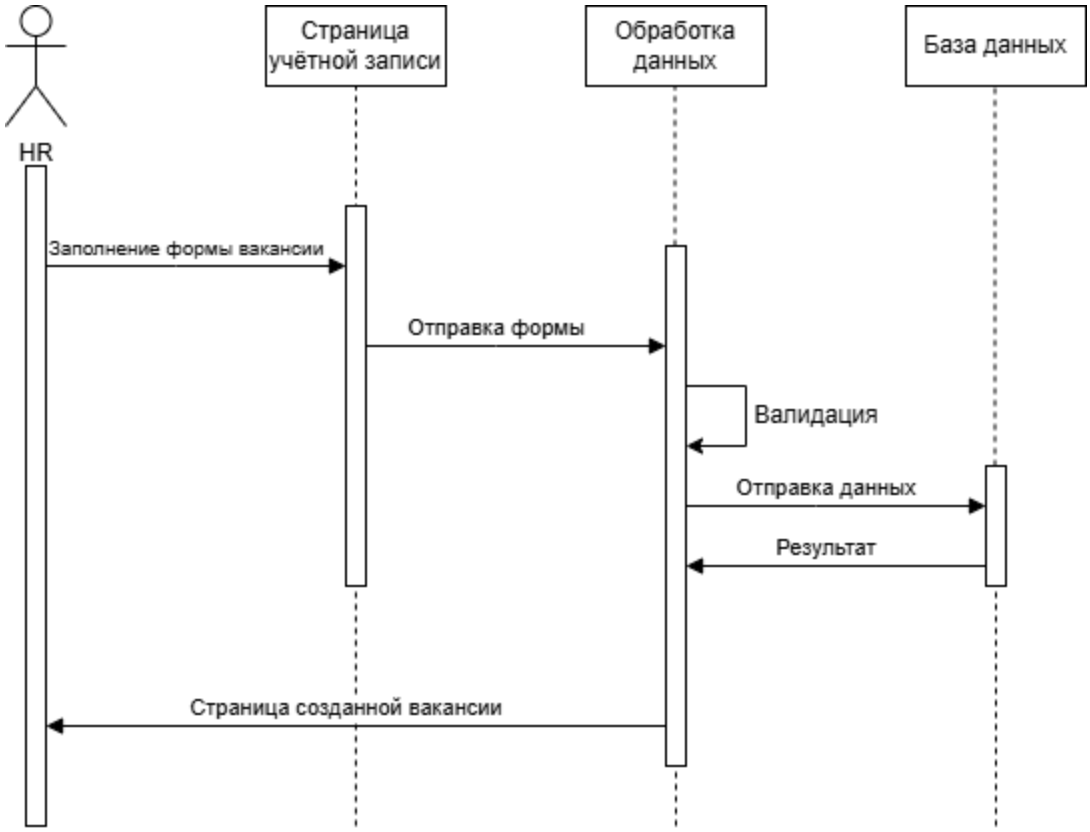

Рисунок 2.14 — Диаграмма последовательности создания вакансии

На представленной диаграмме изображён процесс создания вакансии.

Система предоставляет форму создания вакансии. навыки и компетенции могут выбираться из стандартизированного справочника, а не вводиться вручную. Пользователь заполняет форму. Форма в реальном времени валидируется. Затем она отправляется на сервер, где проходит основную обработку. При прохождении всех проверок создаётся запись в базе данных и отправляется ответ на фронтенд. Происходит перенаправление на страницу созданной вакансии.

На рисунке 2.15 представлена диаграмма последовательности, описывающая процесс создания вакансии.

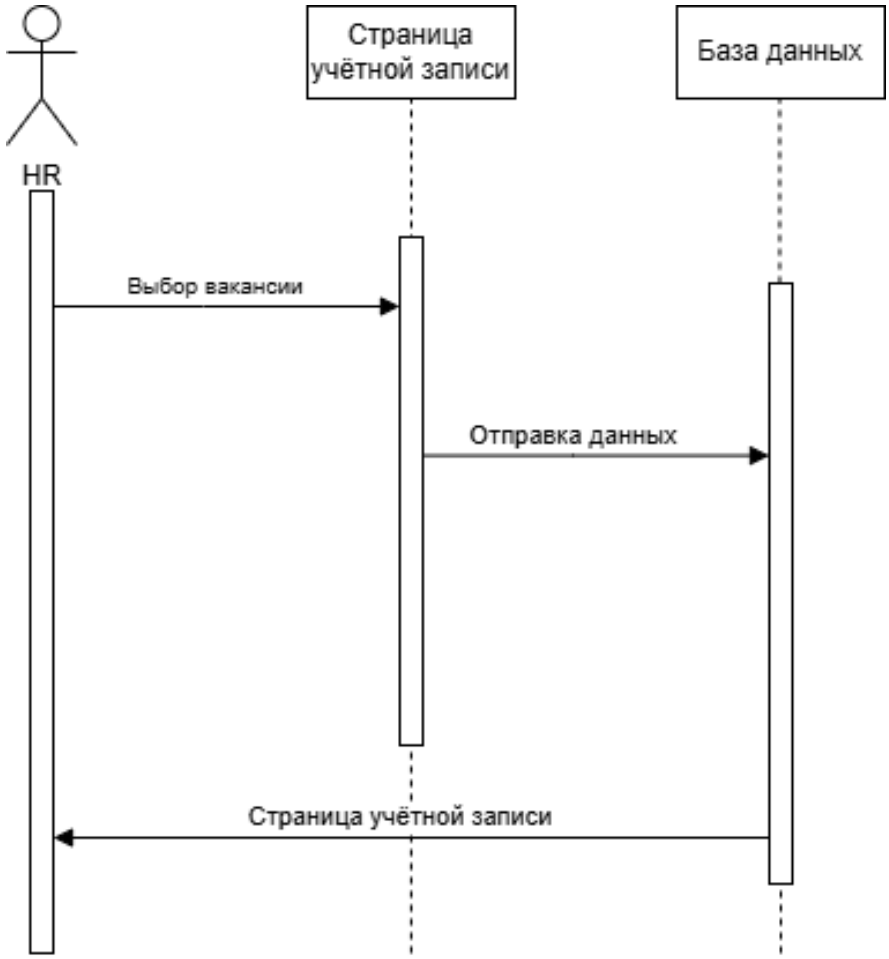

Рисунок 2.15 — Диаграмма последовательности создания вакансии

Когда пользователь решает удалить вакансию, процесс начинается с его действия - нажатия кнопки "Удалить" в интерфейсе системы. Система получает этот запрос и приступает к его обработке. Первым делом проверяется авторизация пользователя - система определяет, вошел ли пользователь в свой аккаунт. Если авторизация отсутствует, пользователь перенаправляется на страницу входа с соответствующим уведомлением.

После успешной проверки авторизации система переходит к проверке прав доступа. Здесь анализируется, имеет ли данный пользователь достаточно полномочий для удаления именно этой вакансии. Проверяется, является ли он создателем вакансии. Если проверка прав не пройдена, система возвращает ошибку.

На следующем этапе появляется запрос подтверждения действия. Если пользователь выбирает отмену, процесс останавливается, и интерфейс возвращается в исходное состояние. При подтверждении система переходит к непосредственному удалению данных.

На рисунке 2.16 представлена диаграмма последовательности, описывающая процесс отклика на вакансию.

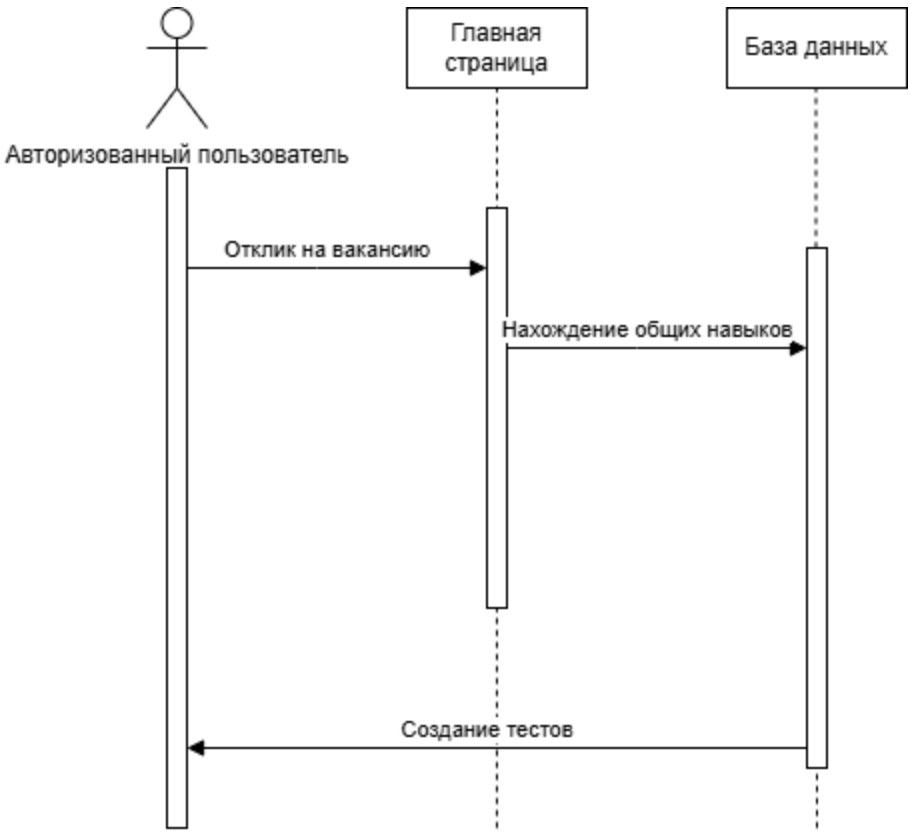

Рисунок 2.16 — Диаграмма последовательности отклика

Процесс отклика на вакансию - это сложный многоуровневый механизм взаимодействия кандидата с системой. Когда авторизованный пользователь нажимает кнопку отклика, система сначала проверяет полноту его профиля. Затем анализируется соответствие кандидата базовым требованиям вакансии. При обработке отклика система автоматически анализирует профиль кандидата и выделяет ключевые совпадения с требованиями вакансии. Эти данные структурируются и представляются работодателю в удобном формате. Кандидат получает подтверждение отклика и информацию о следующих шагах.

На рисунке 2.17 представлена диаграмма последовательности, описывающая процесс прохождения теста.

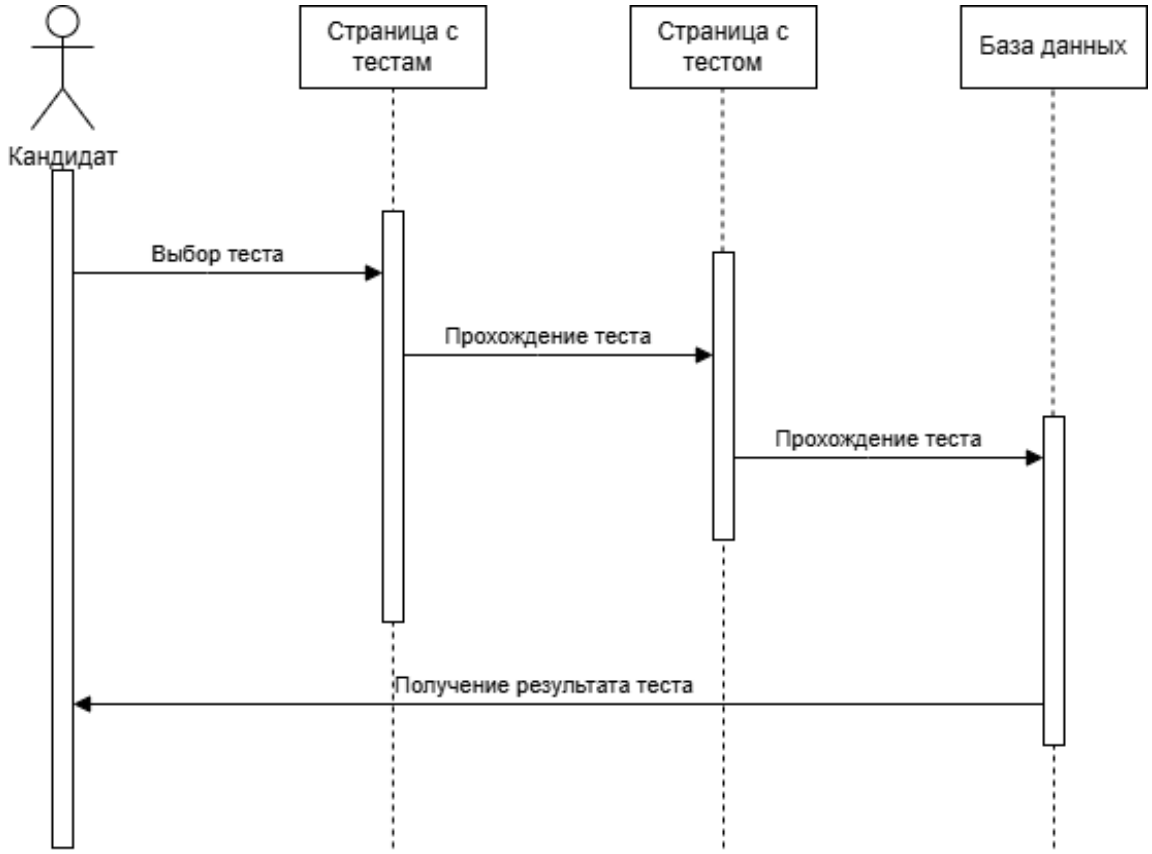

Рисунок 2.17 — Диаграмма последовательности создания вакансии

Процесс тестирования кандидатов представляет собой комплексную систему оценки компетенций. Когда пользователь начинает тест, система загружает вопросы в соответствии с заранее определенной схемой. Если пользователь ответил на все вопросы, то он отправляет свои ответы. Ответы проверяются. Результаты автоматически привязываются к профилю кандидата и отображаются работодателю в контексте конкретной вакансии.

На рисунке 2.18 представлена диаграмма последовательности, описывающая процесс формирования отчёта.

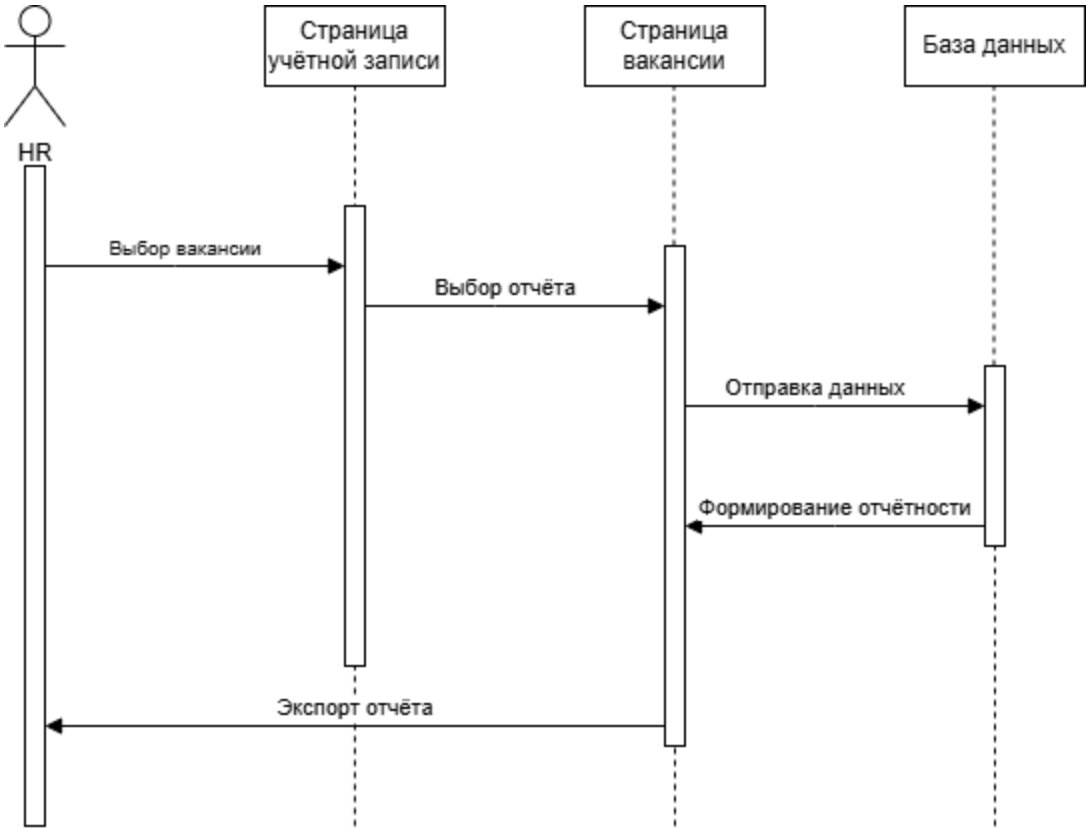

Рисунок 2.18 — Диаграмма последовательности формирования отчёта

Когда пользователь запрашивает отчет, система сначала проверяет его права доступа к запрашиваемым данным. Затем анализируются параметры отчета - временной период, категории вакансий, типы метрик. В процессе формирования отчета данные извлекаются из нескольких источников — основная база вакансий, таблица кандидатов, журнал действий. Особое внимание уделяется визуализации данных - автоматический подбор типов диаграмм, настройка уровней детализации. Для больших объемов данных реализованы механизмы фоновой генерации отчетов с уведомлением о завершении. Готовый отчет можно не только просматривать в системе, но и экспортировать в различных форматах с настройкой параметров экспорта.

На рисунке 2.19 представлена диаграмма последовательности, описывающая процесс редактирования профиля.

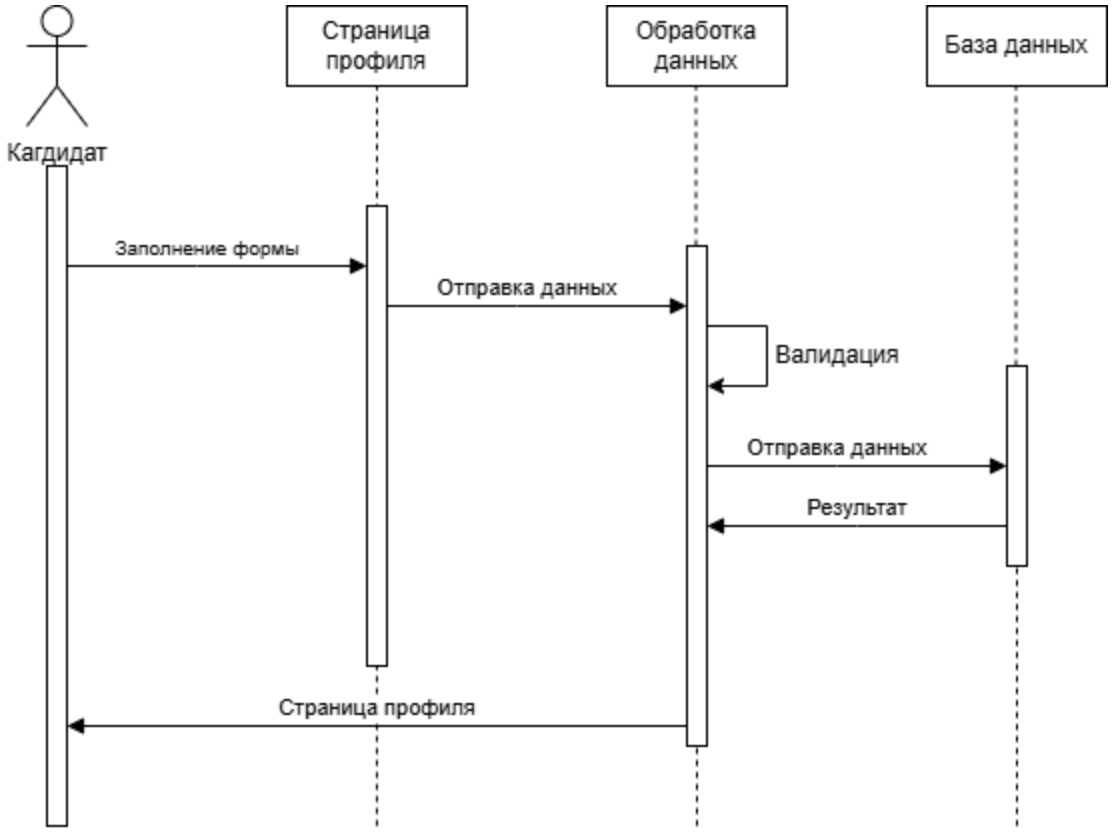

Рисунок 2.19 — Диаграмма последовательности редактирования профиля

Когда пользователь начинает редактирование, система загружает текущие данные профиля, выделяя обязательные и дополнительные поля. При сохранении изменений выполняется комплексная проверка данных - уникальность контактной информации, соответствие форматам, отсутствие противоречий. После сохранения профиль проходит автоматическую переиндексацию для поисковых механизмов системы.

2.3 Проектирование базы данных

Когда пользователь начинает редактирование, система загружает текущие данные профиля, выделяя обязательные и дополнительные поля. При сохранении изменений выполняется комплексная проверка данных - уникальность контактной информации, соответствие форматам, отсутствие противоречий. После сохранения профиль проходит автоматическую переиндексацию для поисковых механизмов системы.

Современные информационные системы играют ключевую роль в управлении данными, автоматизации бизнес-процессов и поддержке принятия

решений. Одним из важнейших компонентов таких систем является база данных, которая обеспечивает структурированное хранение, эффективный доступ и надежное управление информацией. Проектирование базы данных – это сложный и многоэтапный процесс, требующий глубокого анализа предметной области, выбора оптимальной модели данных, разработки структуры и последующей реализации с учетом требований производительности, безопасности и масштабируемости.

В условиях стремительного роста объемов данных и усложнения бизнес-процессов возрастает потребность в эффективных системах хранения и обработки информации. Грамотно спроектированная база данных позволяет минимизировать избыточность, обеспечить целостность данных, ускорить выполнение запросов и упростить сопровождение системы. Неправильный подход к проектированию, напротив, может привести к снижению производительности, возникновению аномалий при обновлении данных и увеличению затрат на поддержку системы.

Целью данной главы является рассмотрение основных этапов проектирования базы данных, включая концептуальное, логическое и физическое моделирование. В рамках работы будут изучены методологии проектирования, принципы нормализации, выбор системы управления базами данных и оптимизация структуры данных. Особое внимание уделяется анализу предметной области, построению ER-диаграмм, преобразованию концептуальной модели в реляционную схему, а также вопросам индексации, безопасности и резервного копирования.

Актуальность темы обусловлена тем, что качество проектирования базы данных напрямую влияет на эффективность работы всей информационной системы. Внедрение современных подходов к моделированию данных, использование передовых СУБД и соблюдение лучших практик проектирования позволяют создавать надежные, производительные и удобные в эксплуатации решения.

Проектирование базы данных является фундаментальным этапом создания информационной системы, определяющим ее функциональность, производительность и долгосрочную устойчивость. В последующих разделах главы будут детально рассмотрены ключевые аспекты этого процесса, а также приведены примеры реализации на основе современных технологий.

2.3.1 Концептуальная модель данных

Когда пользователь начинает редактирование, система загружает текущие данные профиля, выделяя обязательные и дополнительные поля. При сохранении изменений выполняется комплексная проверка данных — уникальность контактной информации, соответствие форматам, отсутствие противоречий. После сохранения профиль проходит автоматическую переиндексацию для поисковых механизмов системы.

Концептуальное проектирование базы данных представляет собой первый этап в процессе построения информационной модели, который направлен на формализацию предметной области без учёта особенностей конкретной СУБД или языка реализации. Главной задачей на этом этапе является выделение всех ключевых сущностей и определение взаимосвязей между ними. Результатом становится обобщённое логическое представление структуры данных, понятное как разработчикам, так и специалистам в области, не связанным с программированием.

В рамках данной выпускной квалификационной работы разрабатывается веб-приложение, предназначенное для автоматизированного подбора персонала. Предметная область включает три основных типа пользователей: кандидаты, HR-специалисты и администраторы. Каждый из них взаимодействует с системой по-разному, что отражается в структуре будущей базы данных.

Кандидаты — это пользователи, использующие систему для поиска работы. Они регистрируются в приложении, создают и заполняют свои профили, указывают навыки, прикрепляют резюме, проходят тестирование и откликаются на размещённые вакансии. Таким образом, для представления кандидата как

сущности необходимо предусмотреть хранение как личной информации, так и связанных данных: навыков, откликов и результатов тестов.

HR-специалисты — это пользователи, представляющие компании. Их основная задача заключается в публикации вакансий, просмотре откликов и анализе результатов тестов кандидатов. Система должна позволять каждому HR-специалисту создавать и редактировать множество вакансий, указывать требования к навыкам, а также иметь доступ к статистике по откликам и успешным прохождениям тестов.

Администраторы — это особая категория пользователей, которые обладают полными правами в системе. Они контролируют целостность данных, управляют всеми объектами базы данных и могут редактировать информацию обо всех пользователях. Администратор также может управлять списком доступных навыков, тестов и настроек системы.

Для отражения предметной области в концептуальной модели были выделены следующие сущности:

- пользователь — обобщённая сущность, описывающая всех участников системы. Содержит идентификатор, электронную почту, пароль и роль (кандидат, HR, администратор).
- кандидат — пользователь, ищущий работу. Имеет личные данные, список навыков, отклики на вакансии и результаты прохождения тестов.
- HR-специалист — пользователь, публикующий вакансии от имени компании. Имеет данные о работодателе и связан с размещёнными вакансиями.
- вакансия — описание рабочей позиции с требованиями к навыкам, уровнем заработной платы и описанием обязанностей.
- навык — отдельная компетенция, необходимая для успешного выполнения работы. Может быть связан как с кандидатом, так и с вакансией.

- отклик — действие кандидата, проявляющего интерес к конкретной вакансии. Отражает факт связи между кандидатом и вакансией.
- тест — инструмент для оценки знаний и навыков кандидата. Привязан к вакансии и используется HR-специалистом.
- результат теста — информация о прохождении кандидатом теста: набранный балл, дата прохождения и успешность.

Можно выделить следующие связи между сущностями:

- один пользователь может быть только одного типа: кандидат, HR или администратор.
- один HR может публиковать множество вакансий, но каждая вакансия создаётся только одним HR.
- один кандидат может откликнуться на несколько вакансий, и одна вакансия может получить отклики от множества кандидатов. Это отражает связь многие-ко-многим между сущностями Кандидат и Вакансия через сущность Отклик.
- один кандидат может обладать множеством навыков, и каждый навык может встречаться у различных кандидатов. Аналогичная связь существует между Вакансиями и Навыками.
- каждая вакансия может быть связана только с одним тестом, но один тест может быть пройден разными кандидатами, что фиксируется в сущности Результат теста.

Концептуальная модель отражает ключевые элементы предметной области и их взаимосвязи. Она служит основой для последующего логического и физического проектирования базы данных.

2.3.2 Логическая модель данных

Логическая модель данных представляет собой важный этап проектирования базы данных, на котором осуществляется формализация структуры информации и связей между данными. Этот этап следует за концептуальным моделированием и предшествует физической реализации,

выполняя роль связующего звена между бизнес-требованиями и технической реализацией.

Основная задача логической модели заключается в детализированном представлении данных в виде структурированной схемы, готовой для последующей реализации в системе управления базами данных. В отличие от концептуальной модели, которая описывает данные на высоком уровне абстракции, логическая модель применяет строгие правила нормализации, определяет типы данных и устанавливает связи между сущностями. При этом она сохраняет независимость от конкретной СУБД, что позволяет использовать её как универсальный инструмент проектирования.

Значение логического моделирования трудно переоценить, так как именно на этом этапе закладываются основы будущей базы данных. Правильно разработанная логическая модель обеспечивает целостность данных, минимизирует избыточность и создает предпосылки для эффективной работы информационной системы. Она служит общим языком для всех участников проекта - от бизнес-аналитиков до разработчиков, позволяя точно и однозначно описать структуру данных.

На рисунке 2.20 представлена ER-диаграмма.

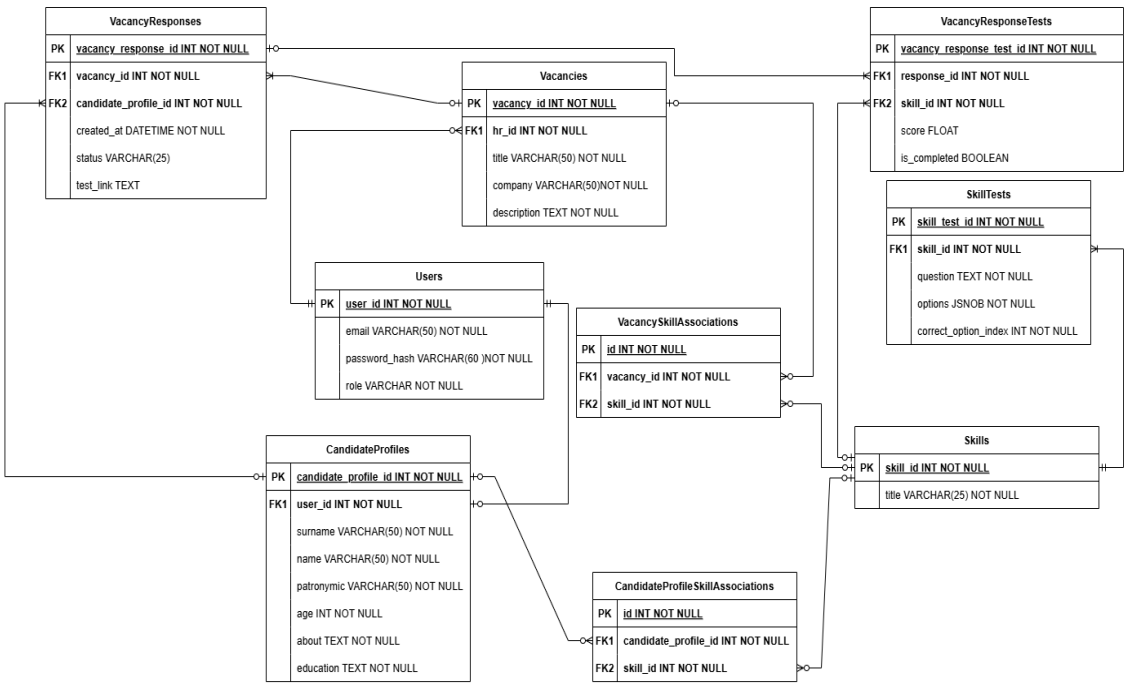

Рисунок 2.20 — ER-диаграмма

На представленной диаграмме изображена структура базы данных, предназначенной для системы подбора вакансий. Диаграмма представлена в виде ER-модели, в которой отражены основные сущности, их атрибуты и связи между ними.

Сущность Users представляет всех пользователей системы: кандидатов, HR-специалистов и администраторов. Каждому пользователю соответствует уникальный идентификатор user_id, который используется как первичный ключ. Для идентификации пользователя по email в таблице предусмотрено поле email, а для аутентификации — поле password_hash, в котором хранится хеш пароля. Также указывается роль пользователя — например, кандидат, HR или администратор — в текстовом поле role. Один пользователь может быть связан с одной записью в таблице CandidateProfiles, если он является кандидатом. Если пользователь — HR, он может быть связан с несколькими записями в таблице Vacancies, поскольку один HR может создавать множество вакансий. Таким образом, таблица Users участвует в связях один-к-одному с CandidateProfiles и один-ко-многим с Vacancies.

Сущность CandidateProfiles описывает профили пользователей, являющихся кандидатами. В каждом профиле содержится персональная информация о кандидате: фамилия, имя, отчество, возраст, описание, а также сведения об образовании. Каждому профилю соответствует уникальный идентификатор candidate_profile_id, который является первичным ключом. Через внешний ключ user_id обеспечивается связь с таблицей Users, то есть каждый профиль принадлежит конкретному пользователю. Один профиль кандидата может быть связан с множеством откликов на вакансии, которые хранятся в таблице VacancyResponses. Кроме того, каждый кандидат может обладать множеством навыков, которые отражены в связующей таблице CandidateProfileSkillAssociations. Таким образом, профиль кандидата участвует в связи один-ко-многим с откликами и во множественной связи многие-ко-многим с навыками.

Сущность Vacancies содержит сведения о вакансиях, размещённых HR-специалистами. Каждая вакансия имеет уникальный идентификатор vacancy_id, являющийся первичным ключом. Дополнительно указаны название вакансии, наименование компании и текстовое описание. Каждая вакансия принадлежит одному HR, что выражается через внешний ключ hr_id, ссылающийся на таблицу Users. Одна вакансия может быть связана с множеством откликов, которые хранятся в таблице VacancyResponses. Также одна вакансия может требовать множество навыков, связь с которыми осуществляется через таблицу VacancySkillAssociation, где реализована связь многие-ко-многим между вакансиями и навыками.

Сущность VacancyResponses предназначена для хранения информации об откликах кандидатов на вакансии. Каждому отклику присваивается уникальный идентификатор vacancy_response_id. Внешние ключи vacancy_id и candidate_profile_id указывают соответственно на вакансию и профиль кандидата, к которым относится данный отклик. Также сохраняются дата создания отклика, его статус и ссылка на тестовое задание. Таким образом, каждый отклик связан с одной вакансией и одним кандидатом, формируя связи многие-к-одному. Кроме того, к одному отклику могут быть привязаны результаты тестирования, которые отражаются в таблице VacancyResponseTests, реализуя связь один-ко-многим между откликами и результатами тестов.

Сущность VacancyResponseTests хранит результаты тестов, которые кандидат проходит в рамках отклика на вакансию. Каждая запись содержит уникальный идентификатор vacancy_response_test_id. Через внешний ключ response_id тест связывается с откликом, а skill_id — с конкретным навыком, к которому относится данный тест. В таблице также указаны балл за тест и флаг, показывающий, завершён ли тест. Таким образом, один отклик может содержать несколько результатов тестов, каждый из которых соответствует определённому навыку.

Сущность Skills представляет список навыков, используемых в системе. Каждому навыку соответствует уникальный идентификатор skill_id и текстовое

название title. Один навык может встречаться в нескольких вакансиях, если он требуется для выполнения работы — такая связь реализована через таблицу VacancySkillAssociation. Также навык может быть у нескольких кандидатов — эта связь отображена в таблице CandidateProfileSkillAssociations.

Кроме того, каждому навыку может соответствовать множество тестов в таблице SkillTests, а также множество тестов отклика в таблице VacancyResponseTests. Таким образом, Skills участвует сразу в нескольких связях многие-ко-многим и один-ко-многим с другими таблицами.

Сущность SkillTests содержит тестовые вопросы, связанные с конкретными навыками. Каждой записи присваивается уникальный идентификатор skill_test_id. Через внешний ключ skill_id определяется, к какому навыку относится тест. Также указывается текст вопроса, варианты ответа в формате JSONB и индекс правильного ответа. Один навык может быть связан с несколькими тестами, реализуя связь один-ко-многим между Skills и SkillTests.

Сущность VacancySkillAssociation представляет собой связующую таблицу, отражающую отношение многие-ко-многим между вакансиями и навыками. Каждая запись имеет уникальный идентификатор id. Через внешние ключи vacancy_id и skill_id устанавливается, какой навык требуется для какой вакансии. Это позволяет гибко задавать требования к разным вакансиям и переиспользовать одни и те же навыки в разных вакансиях.

Сущность CandidateProfileSkillAssociations также является связующей таблицей, предназначенной для отображения связи многие-ко-многим между профилями кандидатов и навыками. У каждой записи есть уникальный идентификатор id, а также внешние ключи candidate_profile_id и skill_id, указывающие, какой кандидат обладает каким навыком. Эта таблица позволяет учитывать индивидуальные особенности кандидатов и применять навык в нескольких профилях.

Вся модель данных строится на логике ролевого взаимодействия: пользователь в зависимости от своей роли может быть кандидатом или HR, размещать вакансии, откликаться на них, проходить тесты, демонстрировать

навыки, которые сопоставляются с требованиями вакансии. Благодаря множеству связей один-ко-многим и многие-ко-многим, структура базы данных позволяет гибко управлять процессом подбора персонала и учитывать множество индивидуальных характеристик как вакансий, так и кандидатов.

Выводы по главе

Во второй главе был выполнен подробный анализ требований к аппаратно-программной среде, необходимой для разработки и эксплуатации информационной системы по подбору кандидатов. Были рассмотрены вопросы обеспечения производительности, отказоустойчивости, безопасности и масштабируемости системы. Эти аспекты особенно важны в условиях обработки персональных данных, высоких требований к надёжности и быстродействию.

Разработка информационной системы предполагает не только проектирование программных модулей, но и продуманное архитектурное решение всей инфраструктуры. Указанные в главе технические параметры обеспечивают необходимый уровень работоспособности, гибкости и безопасности при эксплуатации системы как в рамках локального предприятия, так и в облачном окружении.

Проектирование информационной системы является ключевым этапом разработки, определяющим её архитектуру, функциональность и структуру данных. В процессе проектирования были рассмотрены основные аспекты, включая архитектурные решения, функциональные требования и организацию базы данных. Архитектура системы была проанализирована с использованием контекстной диаграммы, диаграммы компонентов и диаграммы развёртывания, что позволило чётко определить границы системы, её внутренние модули и физическое распределение компонентов. Это обеспечивает понимание взаимодействия системы с внешней средой и между её частями, что особенно важно для масштабируемости и поддержки в будущем.

Функциональные требования были детализированы через диаграмму вариантов использования, диаграмму активности и диаграмму последовательности. Эти модели помогли описать поведение системы, её

взаимодействие с пользователями и внутренние процессы. Такой подход способствует точному определению функциональности, минимизации ошибок на этапе реализации и созданию удобного интерфейса для конечных пользователей.

Особое внимание было уделено проектированию базы данных, включая разработку концептуальной и логической моделей. Концептуальная модель отражает основные сущности и их взаимосвязи, обеспечивая целостное представление о данных, а логическая модель детализирует структуру таблиц, атрибутов и связей. Это позволяет создать эффективную и надёжную базу данных, соответствующую требованиям системы и обеспечивающую быстрый доступ к информации.

Проведённое проектирование обеспечивает основу для дальнейшей реализации информационной системы. Использование современных методов моделирования и стандартов проектирования позволило создать чёткую и согласованную документацию, что упростит процесс разработки и внедрения. Результаты работы демонстрируют готовность системы к реализации, а также её соответствие поставленным техническим и функциональным требованиям.

Глава 3 Процесс разработки информационной системы

3.1 Серверная часть

Серверная часть информационной системы играет ключевую роль в обработке данных, обеспечении бизнес-логики и взаимодействии с клиентскими приложениями. Она выступает в качестве посредника между пользовательским интерфейсом, базами данных и внешними сервисами, гарантируя безопасность, производительность и надежность работы всей системы.

3.1.1 Конфигурация проекта

Конфигурация проекта является важным этапом разработки информационной системы, так как она определяет основные параметры работы приложения. Настройки, хранящиеся в конфигурационных файлах, позволяют гибко управлять поведением системы в разных окружениях. Централизованное

хранение конфигурации упрощает поддержку проекта и снижает риск ошибок при развертывании.

На листинге 3.1 представлен код config.py.

Листинг 3.1 — фрагмент кода config.py

```
class AuthJWT(BaseModel):
    private_key: Path = BASE_DIR / "certs" / "jwt-private.pem"
    public_key: Path = BASE_DIR / "certs" / "jwt-public.pem"
    algorithm: str = "RS256"
    access_token_expire_minutes: int = 15
    refresh_token_expire_days: int = 30

class DBSettings(BaseModel):
    url: str = "postgresql+asyncpg://postgres:postgres@localhost:5432/fastapi_db"
    echo: bool = False

class Settings(BaseSettings):
    auth: AuthJWT = AuthJWT()
    db: DBSettings = DBSettings()
    api_v1_prefix: str = "/api/v1"
```

В данном фрагменте кода есть три класса.

Класс “AuthJWT” отвечает за аутентификацию пользователей. В классе “DBSettnigs” находится адрес базы данных. Класс “Settings” соединяет в себе два предыдущих класса и является основным классом настроек для использования в проекте.

3.1.2 Работа с базой данных

Работа с базой данных является одним из важнейших процессов во время реализации проекта, поскольку именно через БД осуществляется хранение, обработка и управление критически важной информацией. На данном этапе были разработаны модели данных, реализованы запросы и оптимизировано взаимодействие между серверной частью приложения и системой управления базами данных. Взаимодействие с БД организовано через ORM-систему, что позволило абстрагироваться от конкретной СУБД и упростить дальнейшую поддержку проекта.

3.1.2.1 Соединение

Для создания соединения с базой данных был реализован класс “DataBaseHelper”.

Листинг 3.2 — db_helper.py

```
class DataBaseHelper:
    def __init__(self, url: str, echo: bool = False):
        self.engine = create_async_engine(url=settings.db.url, echo=settings.db.echo)
        self.session_factory = async_sessionmaker(
            bind=self.engine, autoflush=False, autocommit=False, expire_on_commit=False
        )
    def get_scoped_session(self):
        session = async_scoped_session(
            session_factory=self.session_factory, scopefunc=current_task
        )
        return session
    async def scoped_session_dependency(self):
        session = self.get_scoped_session()
        yield session
        await session.close()
```

3.1.2.2 Модели данных

Модели данных представляют собой фундаментальную часть архитектуры системы, описывающую структуру хранения и обработки информации.

На листинге 3.3 представлен пример модели данных таблицы “Вакансии”.

Листинг 3.3 — фрагмент класса “Vacancy”

```
class Vacancy(Base):
    __tablename__ = "vacancies"
    title: Mapped[str] = mapped_column(nullable=False, index=True)
    company: Mapped[str] = mapped_column(nullable=False)
    description: Mapped[str] = mapped_column(Text(), nullable=False)
    hr_id: Mapped[int] = mapped_column(
        ForeignKey("users.id"), unique=False, nullable=False
```

Продолжение листинга 3.3.

```
    )
    vacancy_skills: Mapped[list["VacancySkillAssociation"]] = relationship(
        back_populates="vacancy",
        cascade="all, delete-orphan",
        passive_deletes=True,
    )
```

В данном классе описаны поля таблицы и связи с другими таблицами.

3.1.2.3 Механизм миграций

Для преобразования моделей в таблицы базы данных служит инструмент для миграции базы данных Alembic. Он позволяет отслеживать изменения в моделях базы данных и автоматически генерировать соответствующие SQL-скрипты для применения изменений. Alembic работает в связке с SQLAlchemy, анализируя различия между текущим состоянием моделей и предыдущей версией схемы базы данных, создавая инкрементальные миграции. Эти миграции хранятся в виде версионированных Python-скриптов, что обеспечивает контроль над эволюцией структуры базы данных и возможность как применения новых изменений, так и отката к предыдущим версиям. Alembic поддерживает различные СУБД, что делает его универсальным инструментом для управления изменениями схемы базы данных в Python-приложениях.

3.1.2.4 Запросы к базе данных

Инструментом для выполнения запросов к базе данных служит библиотека SQLAlchemy. Она позволяет абстрагироваться от специфики конкретных СУБД, предоставляя единый интерфейс для работы с различными базами данных, включая PostgreSQL, MySQL и SQLite. Библиотека автоматически генерирует оптимизированные SQL-запросы, соответствующие выбранной системе управления базами данных. Основной особенностью SQLAlchemy является реализация ORM, которая преобразует таблицы базы данных в Python-классы, позволяя работать с данными как с объектами. SQLAlchemy автоматически управляет сессиями и транзакциями, обеспечивая целостность данных.

В проекте используются такие запросы, как:

- получение вакансии по id;
- получение списка вакансий;
- создание вакансии;
- получение списка вакансий по пользователю;
- обновление вакансий;
- удаление вакансий
- создание отклика на вакансию;
- получение списка пользователей с профилями, откликами и результатами тестов;
- получение навыков у вакансии;
- получение списка навыков;
- получение навыка по названию;
- получение навыка по id;
- обновление навыка;
- создание навыка;
- удаление навыка;
- создание вопроса по навыку;
- получение вопросов по навыку;
- создание результата по тесту;
- получение пользователя по токену;
- получение пользователя с профилем по токену;
- обновление профиля пользователя;
- обновление навыков в профиле;
- получение тестов для кандидата;
- создание пользователя;
- создание пользователя с профилем.

В листинге 3.1 представлен SQLAlchemy-запрос, который выполняет выборку кандидатов.

Листинг 3.4 — выборка кандидатов с помощью SQLAlchemy

```
select(User)
        .join(User.candidate_profile)
        .join(CandidateProfile.vacancy_responses)
        .options(
            selectinload(User.candidate_profile)
            .selectinload(CandidateProfile.vacancy_responses)
            .selectinload(VacancyResponse.tests),
            selectinload(User.candidate_profile).selectinload(
                CandidateProfile.profile_skills ), ).where(
            User.role == UserRole.CANDIDATE,
            VacancyResponse.vacancy_id == vacancy_id,
        )
```

Данный запрос на SQLAlchemy выполняет комплексную выборку данных о кандидатах, откликнувшихся на конкретную вакансию, с полной загрузкой всех связанных данных. Запрос начинается с выбора пользователей из таблицы users, которые имеют роль CANDIDATE, и соединяет их с профилями кандидатов через таблицу candidate_profiles по связи user_id. Затем происходит соединение с таблицей vacancy_responses, чтобы найти все отклики этих кандидатов, при этом добавляется условие фильтрации по конкретному vacancy_id. Для оптимизации загрузки связанных данных используется стратегия selectinload, которая эффективно подгружает связанные сущности отдельными запросами. В первую очередь загружаются профили кандидатов и связанные с ними отклики на вакансии вместе с результатами, что реализуется через каскадное соединение таблиц candidate_profiles, vacancy_responses и tests. Во вторую очередь отдельно загружаются навыки каждого кандидата из таблицы profile_skills. Такой подход с использованием selectinload вместо стандартного lazy loading или joinedload был выбран для повышения производительности — он генерирует несколько более простых SQL-запросов вместо одного сложного запроса с множеством соединений, что особенно важно при работе с большими объемами данных или сложными связями между таблицами. В результате

выполнения этого запроса формируется полная информация о кандидатах, откликнувшихся на указанную вакансию, включая их профили, все соответствующие отклики с тестами и перечень профессиональных навыков каждого соискателя, что обеспечивает комплексное представление данных для последующего анализа в системе подбора персонала.

3.1.3 Разработка API

API является важнейшим компонентом информационной системы, обеспечивающим взаимодействие между клиентской частью и сервером. В рамках данного проекта было разработано API на основе REST-архитектуры с использованием фреймворка FastAPI, что позволило достичь высокой производительности, простоты разработки и удобства интеграции. Основная задача разработанного API заключается в обработке клиентских запросов, включая операции аутентификации и CRUD-операции с данными. Для обеспечения надежности работы API реализована строгая валидация входных данных с использованием Pydantic-схем, а также система безопасности на основе JWT-аутентификации с проверкой прав доступа.

3.1.3.1 Приложения

Разделение API на приложения позволяет добиться модульности и масштабируемости системы, упрощая поддержку и дальнейшее развитие проекта. Такой подход дает несколько ключевых преимуществ. Во-первых, он обеспечивает четкое разделение ответственности между различными функциональными блоками системы — каждое приложение фокусируется на своей конкретной предметной области. Во-вторых, это позволяет независимо разрабатывать, тестировать и развертывать отдельные компоненты системы. Кроме того, модульная архитектура упрощает процесс отладки, так как проблемы можно локализовать в конкретном приложении, не затрагивая работу всей системы.

Разделение на приложения также способствует лучшей организации кода — каждый модуль содержит свои представления, схемы, зависимости, роутеры в одном логически связанном пространстве. Это делает код более читаемым и поддерживаемым. С точки зрения производительности, такой подход позволяет оптимизировать загрузку только необходимых компонентов при обработке запросов, уменьшая общий объем используемой памяти. При необходимости масштабирования системы можно распределить разные приложения между различными серверами или контейнерами, обеспечивая балансировку нагрузки.

Еще одним важным преимуществом является возможность повторного использования приложений в разных проектах — хорошо спроектированные модули могут быть легко интегрированы в новые системы с минимальными доработками. Это значительно ускоряет разработку новых продуктов. Кроме того, модульная структура упрощает процесс документирования API, так как каждый функциональный блок может быть описан отдельно, что делает документацию более структурированной и понятной для разработчиков.

В таблице 3.1 представлены приложения с содержанием.

Таблица 3.1 — содержание всех приложений

Название приложения	Содержание
Профили	schemas.py, crud.py, views.py, dependencies.py.
Тесты	schemas.py, crud.py, views.py.
Навыки	schemas.py, crud.py, views.py.
Вакансии	schemas.py, crud.py, views.py.
Аутентификация	auth_helpers.py, dependencies.py, schemas.py, utils.py, views.py.
Пользователи	schemas.py, crud.py, views.py.

Во всех приложениях есть представления, которые позволяют выполнять роутинг по эндпоинтам. Также во всех приложениях есть файл “schemas.py”, в котором описаны схемы для валидации входных и выходных данных. В файлах “crud.py” написана основная бизнес-логика. В dependencies.py расположены

нужные зависимости для приложения. В приложении аутентификации auth_helpers.py служит для организации бизнес-логики, а utils.py — для утилит.

А также в каждом файле с представлениями есть роутеры, которые подключаются к одному основному роутеру в main.py.

В листинге 3.5 представлено представление удаления вакансии.

Листинг 3.5 — представление удаления вакансии с аутентификацией

```
@router_with_auth.delete("/vacancy/delete/{vacancy_id}/")
async def delete_vacancy(
    vacancy_id: int,
    payload: dict = Depends(get_current_token_payload),
    session: AsyncSession = Depends(db_helper.scoped_session_dependency),
):
    return await crud.delete_vacancy(
        vacancy_id=vacancy_id, payload=payload, session=session
    )
```

Представление delete_vacancy представляет собой защищенный API-эндпоинт для удаления вакансий, который только для аутентифицированных пользователей. Этот DELETE-метод принимает идентификатор вакансии в URL-параметре и автоматически проверяет авторизацию пользователя через механизм зависимости get_current_token_payload, который извлекает и верифицирует JWT-токен из заголовков запроса, возвращая декодированные данные пользователя в виде словаря payload. Для работы с базой данных функция использует асинхронную сессию AsyncSession, которая автоматически создается и управляется через dependency injection. Основная логика удаления вакансии инкапсулирована в CRUD-слое, куда передаются полученные параметры: идентификатор вакансии, данные пользователя из токена и сессия базы данных. Такая архитектура обеспечивает четкое разделение ответственности между слоями приложения — роутер обрабатывает HTTP-запрос и проверяет авторизацию, в то время как вся бизнес-логика и работа с базой данных делегируется в соответствующий сервисный слой. Функция возвращает результат выполнения операции, который автоматически преобразуется в

соответствующий HTTP-ответ, включая обработку возможных ошибок Этот эндпоинт следует принципам RESTful-дизайна, используя соответствующий HTTP-метод и возвращая стандартные коды состояния HTTP для различных сценариев выполнения операции.

В листинге 3.6 представлена функция, реализующая удаление вакансии из базы данных.

Листинг 3.6 — функция удаления вакансии

```
async def delete_vacancy(vacancy_id: int, payload: dict, session: AsyncSession):
    user = await get_user_by_sub(payload=payload, session=session)
    check_access(user=user, role=UserRole.HR)
    vacancy = await session.get(Vacancy, vacancy_id)
    await session.delete(vacancy)
    await session.commit()
    return None
```

Функция delete_vacancy представляет собой асинхронный CRUD-метод для удаления вакансий из системы, который выполняет несколько важных проверок перед выполнением операции. В начале функции происходит получение объекта пользователя из базы данных на основе данных из JWT-токена, что обеспечивает идентификацию того, кто пытается удалить вакансию. Затем выполняется проверка прав доступа через функцию check_access, которая удостоверяется, что пользователь имеет роль HR-специалиста, так как только сотрудники отдела кадров должны иметь возможность удалять вакансии. После подтверждения прав функция получает объект вакансии по переданному идентификатору vacancy_id. Если вакансия существует, она удаляется из базы данных, изменения фиксируются.

3.1.3.2 Валидация данных

Для валидации данных используются Pydantic-схемы, которые обеспечивают строгую типизацию и автоматическую проверку структуры передаваемых данных. Эти схемы позволяют гибко настраивать правила валидации для входящих запросов и исходящих ответов, гарантируя

соответствие данных ожидаемым форматам на уровне API. Например, для вакансий могут быть определены отдельные схемы для создания, обновления и отображения, где каждая схема включает только соответствующие поля с указанием их типов, обязательности и дополнительных ограничений. Pydantic автоматически проверяет данные при их получении, преобразует типы согласно схеме и генерирует понятные ошибки валидации, если данные не соответствуют требованиям.

На листинге 3.7 представлен пример схемы профиля кандидата в личном кабинете.

Листинг 3.7 — схема профиля кандидата

```
class CandidateProfileUser(BaseModel):
    email: Annotated[EmailStr, MinLen(5), MaxLen(25)]
    name: Annotated[str, MinLen(2), MaxLen(50)]
    surname: Annotated[str, MinLen(2), MaxLen(50)]
    patronymic: Annotated[str, MinLen(2), MaxLen(50)]
    age: int
    about_candidate: str
    education: str
    skills: list[SkillBase]
```

3.1.3.3 Аутентификация

Система аутентификации реализована на основе JWT-токенов, обеспечивая надежную защиту API и простую интеграцию с клиентскими приложениями. В основе лежит стандартный OAuth2 Password Flow, адаптированный под специфику HR-платформы. При успешной аутентификации через эндпоинт “/login” система генерирует пару токенов — access token с коротким сроком жизни и refresh token с более длительным действием

Хранение паролей организовано с использованием современного алгоритма хеширования bcrypt, что исключает возможность их компрометации даже в случае утечки базы данных. Все токены подписываются

асимметричными RSA-ключами, что обеспечивает высокий уровень криптографической защиты.

Ролевая модель системы интегрирована непосредственно в процесс аутентификации. По указанной при логине электронной почте проверяются права доступа на уровне API-эндпоинтов.

Для веб-клиентов обеспечена правильная настройка CORS, предотвращающая межсайтовые атаки.

Аутентификация тесно интегрирована с системой документации API. В Swagger UI доступна удобная кнопка авторизации, позволяющая разработчикам легко тестировать защищенные эндпоинты. Все ошибки аутентификации возвращают стандартизированные коды состояния 401 или 403 с чёткими сообщениями об ошибках, что упрощает отладку клиентских приложений.

3.2 Клиентская часть

Фронтенд данного программного продукта представляет собой клиентскую часть информационной системы автоматизированного подбора персонала. Основная цель интерфейса заключается в обеспечении взаимодействия конечного пользователя с системой посредством визуальных элементов, форм, таблиц и других интерактивных компонентов. Фронтенд реализован с применением классического подхода: каждая страница представлена отдельным HTML-документом, стилизована с помощью CSS и использует JavaScript для динамического взаимодействия с серверной частью через REST API. Такой подход обеспечивает наглядную модульную структуру, удобную для понимания, тестирования и сопровождения.

Интерфейс ориентирован на три типа пользователей: кандидатов, HR-специалистов и администраторов. Каждый тип пользователя после аутентификации получает доступ к ограниченному набору страниц и функций, соответствующих его роли в системе. Фронтенд не использует фреймворки вроде React, Angular или Vue. Простота архитектуры позволяет сосредоточиться

на логике взаимодействия, структуре системы и реализации пользовательских сценариев.

Главная страница системы. Выполняет функцию точки входа для всех пользователей. На ней отображается приветственное сообщение и ссылки для перехода на страницы регистрации и авторизации. Эта страница не требует взаимодействия с сервером, так как предназначена исключительно для начальной навигации. После нажатия на кнопку входа или регистрации пользователь перенаправляется на соответствующую страницу.

Страница регистрации нового пользователя в системе. Содержит форму для ввода следующих данных: имя, адрес электронной почты, пароль и выбор роль. После заполнения формы данные отправляются на сервер методом POST. На клиентской стороне реализована валидация введённых значений, предотвращающая отправку пустых или некорректных данных. В случае успешной регистрации пользователь получает уведомление и перенаправляется на страницу входа.

Страница входа в систему. Пользователь вводит адрес электронной почты и пароль. После отправки формы выполняется запрос на сервер для проверки учётных данных. В случае успешной авторизации сервер возвращает токен, который сохраняется в локальном хранилище браузера. В зависимости от роли пользователя происходит автоматическое перенаправление: кандидат попадает на страницу профиля, HR — на страницу управления вакансиями, администратор — на страницу редактирования навыков и тестов.

Личный кабинет кандидата. Пользователь с ролью “кандидат” может редактировать информацию о себе, такую как ФИО, возраст, образование, контакты и список профессиональных навыков. Список навыков загружается с сервера, а также может быть изменён с помощью интерфейса. Вся информация сохраняется на сервере и используется при откликах на вакансии и просмотре профиля HR-специалистами. Также реализована возможность загрузки и отображения резюме.

Страница управления профессиональными навыками. Здесь отображается список всех доступных навыков в системе, которые могут быть выбраны кандидатами или привязаны к вакансиям. Администратор может добавлять новые навыки, редактировать или удалять существующие. Каждое действие сопровождается отправкой запроса на сервер и обновлением отображаемых данных. Навыки хранятся централизованно и доступны для выбора на других страницах.

Страница прохождения теста кандидатом. Загружает с сервера список вопросов, каждый из которых содержит варианты ответов. Кандидат выбирает ответы и отправляет их на проверку. После завершения теста отображается результат: количество правильных и неправильных ответов. Система сохраняет результаты на сервере, а также предоставляет доступ к ним HR-сотрудникам для оценки знаний кандидата.

Страница просмотра всех открытых вакансий, доступная кандидатам. Данные о вакансиях загружаются с сервера. Для каждой вакансии отображается название, описание, требования, список навыков и кнопка отклика. При нажатии на кнопку происходит отправка отклика кандидата, который сохраняется на сервере.

Страница создания или редактирования вакансии. Доступна HR-сотрудникам. При открытии страницы для редактирования уже существующей вакансии данные подгружаются с сервера и автоматически заполняют поля формы. HR может отредактировать название вакансии, описание, требования, добавить навыки и тесты, которые кандидат должен пройти. После завершения редактирования отправляется PUT-запрос на сервер. При создании новой вакансии используется POST-запрос.

Страница управления вакансиями, опубликованными текущим HR-сотрудником. Отображаются всех созданных вакансий с возможностью перехода к их редактированию, удалению или просмотру откликов. Также на этой странице есть форма для создания новой вакансии. Все действия инициируют

соответствующие запросы к серверу. Страница реализует основную административную функцию HR-пользователя.

Страница выбора и редактирования тестов, связанных с конкретной вакансией. HR-сотрудник может указать, какие тесты должны пройти кандидаты, откликнувшиеся на данную вакансию. Предусмотрен выбор из общего списка тестов, загружаемого с сервера. После выбора данные сохраняются и привязываются к вакансии. Это позволяет обеспечить автоматическую проверку знаний кандидатов после подачи заявки.

Фронтенд проекта построен по принципу разделения ответственности: каждая HTML-страница решает строго определённую задачу. Использование JavaScript позволяет реализовать динамическое взаимодействие с сервером и частичное обновление интерфейса без полной перезагрузки страниц. Обмен данными между клиентской и серверной частью осуществляется через REST API. Полученные с сервера данные отображаются на странице в виде таблиц, форм или списков, что обеспечивает удобство взаимодействия и восприятия.

Данная архитектура фронтенда делает систему понятной, расширяемой и удобной для поддержки. Благодаря использованию только базовых веб-технологий, реализация остаётся прозрачной, что особенно важно при учебной разработке и защите дипломного проекта.

3.3 Контейнеризация

Была реализована контейнеризация всех компонентов системы с использованием технологии Docker. Все сервисы, включая backend-приложение, frontend-интерфейс и базу данных PostgreSQL, были упакованы в отдельные Docker-контейнеры для обеспечения согласованной работы системы.

Backend-часть на основе FastAPI была размещена в контейнере на базе официального Python-образа. Внутри контейнера находится приложение со всеми необходимыми зависимостями, включая установленные через requirements.txt библиотеки. Контейнер настроен на работу с определенным портом и подключением к контейнеру с базой данных.

Frontend-часть, реализованная на чистом HTML и CSS без использования JavaScript-фреймворков, была упакована в отдельный контейнер на основе легковесного веб-сервера nginx. Статические файлы (HTML-страницы, CSS-стили и изображения) копируются в контейнер при его сборке. Nginx обеспечивает быструю и надежную отдачу статического контента конечным пользователям.

База данных PostgreSQL была развернута в собственном контейнере с настроенными томами для постоянного хранения информации. Это гарантирует сохранность данных между перезапусками контейнеров. Все компоненты системы связаны через внутреннюю Docker-сеть, что обеспечивает безопасное взаимодействие между сервисами.

Для управления инфраструктурой проекта использовался docker-compose, позволяющий описывать конфигурацию всех сервисов в едином YAML-файле. Такой подход значительно упрощает процесс развертывания системы на любом хосте с установленным Docker, обеспечивая идентичность сред разработки, тестирования и промышленной эксплуатации. Контейнеризация также облегчает масштабирование системы и развертывание дополнительных экземпляров сервисов при необходимости.

На листинге В.1 представлен код файла docker-compose, который запускает контейнеры с компонентами.

3.4 Тестирование

Тестирование разработанной информационной системы проводилось комплексно на всех уровнях архитектуры для обеспечения надежности и стабильности работы. Процесс тестирования включал несколько этапов, направленных на проверку корректности функционирования отдельных компонентов и системы в целом. На первом этапе выполнялось модульное тестирование backend-части приложения с использованием библиотеки pytest. Тесты проверяли работу отдельных функций и методов, включая обработку запросов к API, валидацию входных данных и взаимодействие с базой данных.

Особое внимание уделялось тестированию механизма аутентификации и авторизации пользователей.

Интеграционное тестирование позволило проверить взаимодействие между frontend и backend частями системы. С помощью инструментов Postman и curl тестировались все API-эндпоинты на корректность обработки запросов и возвращаемых ответов. Проверялись различные сценарии работы, включая обработку ошибочных данных и пограничные случаи. Для frontend-части проводилось ручное тестирование интерфейса, включая проверку отображения страниц в разных браузерах и на различных разрешениях экрана. Тестировалась корректность работы всех форм и их взаимодействие с backend-сервисами.

На листинге В.2 представлен тест регистрации кандидата.

Выводы по главе

Заключение

Выпускная квалификационная работа была посвящена разработке информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика, что является актуальной задачей в современном кадровом менеджменте и IT-индустрии в целом. В процессе выполнения работы была проведена глубокая аналитика предметной области, определены ключевые требования и проведено комплексное проектирование и разработка программного продукта, отвечающего современным стандартам и требованиям бизнеса.

Главной целью работы являлась оптимизация и автоматизация процесса подбора персонала, что позволило существенно повысить эффективность взаимодействия между HR-специалистами и техническими подразделениями компании, минимизировать человеческий фактор, а также сократить время и финансовые затраты, связанные с поиском и отбором квалифицированных специалистов. Внедрение автоматизированной системы, основанной на объективном сопоставлении навыков кандидатов с требованиями вакансий, а

также автоматической отправке и оценке тестовых заданий, позволяет обеспечить более качественный и прозрачный процесс подбора.

На этапе исследования была проведена всесторонняя аналитика современных методов и технологий подбора персонала, изучены лучшие практики в сфере автоматизации HR-процессов, а также проведены консультации с профильными экспертами, что позволило сформулировать полный набор функциональных и нефункциональных требований к системе. Особое внимание уделялось вопросам безопасности данных, удобства интерфейса и масштабируемости решения.

Архитектурное проектирование системы выполнено с использованием современных принципов разработки программного обеспечения, включая многослойную архитектуру, что способствует четкому разделению функциональных блоков, облегчает поддержку и развитие системы, а также обеспечивает высокую надежность и отказоустойчивость. Выбор технологического стека, включающего язык программирования Python, веб-фреймворк FastAPI, систему управления базами данных PostgreSQL и ORM SQLAlchemy, обеспечил гибкость, производительность и удобство сопровождения кода.

Особое значение в работе имело обеспечение безопасности и конфиденциальности персональных данных пользователей системы. Были реализованы надежные механизмы аутентификации и авторизации, использованы современные стандарты шифрования, а также проведены мероприятия по защите от типичных уязвимостей веб-приложений, что гарантирует сохранность данных и соблюдение нормативных требований в области информационной безопасности.

Разработка программного продукта сопровождалась применением передовых методик обеспечения качества, включая написание модульных тестов, проведение код-ревью и организацию процессов непрерывной интеграции, что значительно повысило качество конечного продукта и снизило вероятность возникновения ошибок при эксплуатации.

Тестирование и внедрение системы показали высокую производительность и удобство использования, возможность масштабирования под растущие потребности компании, а также положительное влияние на процессы подбора персонала. Сокращение времени отклика на вакансии и повышение объективности оценки кандидатов способствовали улучшению общей эффективности работы HR-отдела и повысили уровень удовлетворенности конечных пользователей.

В ходе выполнения работы были выявлены и проанализированы основные ограничения и возможные риски, связанные с автоматизацией кадровых процессов, что позволило разработать рекомендации по их минимизации и дальнейшему совершенствованию системы. Например, планируется интеграция методов машинного обучения и искусственного интеллекта для повышения точности оценки и прогнозирования успешности кандидатов, что откроет новые горизонты в области автоматизации HR.

Перспективы развития проекта включают расширение функционала за счет подключения внешних платформ и сервисов, внедрение многоуровневой аналитики и отчетности, улучшение пользовательского интерфейса с учетом обратной связи, а также разработку мобильных приложений для повышения доступности системы.

Практическая значимость работы заключается в создании надежного инструмента, способного значительно улучшить процессы подбора персонала, что особенно важно для IT-компаний, стремящихся к оперативному привлечению квалифицированных специалистов в условиях высокой конкуренции на рынке труда. Результаты работы могут быть использованы как основа для дальнейших исследований и разработок в области автоматизации управления человеческими ресурсами.

Таким образом, выполненная выпускная квалификационная работа полностью соответствует заявленным целям и задачам, продемонстрировала высокий уровень теоретической подготовки и практических навыков, и внесла

весомый вклад в развитие современных информационных систем, направленных на оптимизацию бизнес-процессов.

В заключение можно отметить, что достигнутые результаты не только подтвердили эффективность выбранного подхода и технологий, но и открыли новые возможности для совершенствования и масштабирования системы в будущем, что делает её перспективным решением для организаций, заинтересованных в цифровой трансформации кадрового менеджмента.

Список литературы

1. Иванов А.А., Петров К.Д. (2024). «Автоматизированные методы оценки компетенций IT-специалистов» // «Информационные технологии». - № 5. - С. 34–41.

2. Сидорова Е.Л. (2023). «Анализ современных HR-технологий в IT-рекрутинге» // «Управление персоналом». - № 8. - С. 56–63.

3. Кузнецов М.П. (2022). «Искусственный интеллект в подборе технических специалистов»: дис. … канд. экон. наук. - М.: МГУ.

4. Смирнов В.Г. (2023). «IT-рекрутинг: современные методы и инструменты». - СПб.: Питер. - 320 с.

5. Ткаченко И.Н. (2024). «Автоматизация HR-процессов: от теории к практике». - М.: Альпина Паблишер. - 250 с.

6. Рыжков А.Ю. (2023). «Технологии машинного обучения в управлении персоналом». - М.: ИНФРА-М. - 180 с.

7. Белова О.С. (2024). «Методы анализа больших данных в HR-аналитике» // «Бизнес-информатика». - № 1. - С. 22–29.

8. Громов П.К. (2022). «Цифровая трансформация рекрутинга». - М.: Эксмо. - 210 с.

9. Журнал «Директор по персоналу» (2024). Спецвыпуск «IT-рекрутинг 2024: тренды и кейсы».

10. Материалы конференции «HR Digital» (2023). Сборник докладов. - М.: Изд-во «HR-Практика».

11. Мартин Р. Чистый код: создание, анализ и рефакторинг. - М.: Питер, 2020. - 464 с.

12. Эванс Э. Предметно-ориентированное проектирование (DDD). - М.: Вильямс, 2019. - 448 с.

13. Ньюмен С. Создание микросервисов. - М.: Питер, 2021. - 304 с.

14. Фаулер М. Архитектура корпоративных программных приложений. - М.: Вильямс, 2019. - 544 с.

15. Клеппман М. Высоконагруженные приложения. - М.: Питер, 2020. - 608 с.

16. Ричардсон К. Шаблоны микросервисов. - М.: Питер, 2021. - 592 с.

17. Найгард М. Высокодоступные системы. - М.: Питер, 2020. - 368 с.

18. Официальная документация по Java [Электронный ресурс]. - URL: https://docs.oracle.com/javase/ru/

19. Документация Django на русском [Электронный ресурс]. - URL: https://djangodoc.ru/

20. Документация Spring Framework [Электронный ресурс]. - URL: https://spring-projects.ru/

21. Документация Docker на русском [Электронный ресурс]. - URL: https://docs.docker.ru/

22. Документация Kubernetes [Электронный ресурс]. - URL: https://kubernetes.io/ru/docs/

23. Документация PostgreSQL [Электронный ресурс]. - URL: https://postgrespro.ru/docs/

24. Документация MongoDB [Электронный ресурс]. - URL: https://mongodb.com/docs/ru-ru/

Приложение А

Техническое задание

1. Наименование и область применения

Наименование темы работы: «Разработка информационной системы для автоматизированного подбора кандидатов на должность backend-разработчика». Разрабатываемая система предназначена для упрощения работы HR-специалистов, связанной с анализом откликов кандидатов, оценкой их компетенций и автоматической отправкой тестовых заданий по совпадающим навыкам.

2. Основания для разработки

Основанием для разработки является задание на выпускную квалификационную работу, утверждённое кафедрой «Информатика и информационные технологии» Московского политехнического университета.

3. Исполнитель

Исполнителем является студент кафедры «Информатика и информационные технологии» Московского политехнического университета, группы 211-722, Махлышев Матвей Дмитриевич.

4. Назначение разработки

4.1. Назначение системы

Система предназначена для автоматизированного подбора кандидатов на вакансию backend-разработчика с учетом их профессиональных навыков, сопоставления их с требованиями вакансий и последующего тестирования.

4.2. Цели создания системы

Повышение эффективности подбора специалистов в сфере backend-разработки, снижение временных затрат HR-специалистов на первичный отбор, автоматизация оценки технических навыков кандидатов.

5. Технические требования к системе

5.1. Задачи, подлежащие решению

1) Анализ предметной области и исследование существующих решений;
2) Проектирование базы данных и структуры данных;
3) Разработка API и бизнес-логики backend-приложения;
4) Разработка интерфейса для HR и кандидатов;
5) Реализация механизма сравнения навыков кандидата с требованиями вакансии;
6) Автоматизированная отправка тестов;
7) Хранение и отображение результатов тестирования;
8) Тестирование и отладка;
9) Развёртывание системы.

5.2. Требования к функциональным характеристикам

Информационная система должна предоставлять следующие функции:

1) Регистрация и авторизация пользователей;
2) Разграничение прав доступа;

3) Добавление и редактирование вакансий HR-специалистом;
4) Возможность отклика на вакансию кандидатом;
5) Сравнение навыков кандидата с требованиями вакансии;
6) Автоматическая отправка тестов по совпавшим навыкам;
7) Проверка и хранение результатов тестов;
8) Просмотр откликов, навыков и оценок кандидатов HR-специалистами.

5.3. Требования к входным и выходным данным

Информационная система принимает на вход следующие данные:

1) Регистрационные данные
2) Список навыков кандидата
3) Навыки, указанные в вакансии
4) Ответы на тестовые задания

Информационная система предоставляет следующие выходные данные:

1) Уровень совпадения навыков
2) Результаты тестов
3) Список откликов на вакансию
4) Оценка кандидатов по каждому тесту

5.4. Требования к составу программных компонентов

Система должна содержать следующие программные компоненты:

1) Клиентское приложение
2) Серверное приложение
3) База данных

5.5. Требования к компонентам программного продукта.

Пользовательский интерфейс:

1) Простой и понятный дизайн
2) Поддержка адаптивной верстки
3) Быстрая навигация по вакансиям и откликам
4) Уведомления о действиях и ошибках

База данных:

1) Безопасность и целостность данных
2) Хранение информации о пользователях, навыках, вакансиях, откликах и тестах

Серверное приложение:

1) Обработка входящих запросов
2) Валидация данных
3) Сопоставление навыков
4) Отправка и проверка тестов
5) Документированный API

5.6. Требования к составу и параметрам технических средств.

Программное обеспечение:

- ОС: Windows 10+, macOS 11+, Linux
- Браузеры: Chrome 100+, Firefox 100+, Safari 15+, Edge 100+
- Установленные Python 3.10+, PostgreSQL

Аппаратное обеспечение:

- Устройство с доступом в интернет
- Оперативная память от 4 ГБ
- Разрешение экрана не менее 1280x720
- Клавиатура и мышь/тачпад

6. Стадии и этапы разработки

Название этапа	Даты реализации
Анализ предметной области	01.12.2024 – 15.12.2024
Анализ существующих решений	16.12.2024 – 31.12.2024
Анализ технической литературы	01.01.2025 – 15.01.2025
Разработка технического задания	16.01.2025 – 31.01.2025
Проектирование архитектуры системы	01.02.2025 – 15.02.2025
Выбор технологического стека	16.02.2025 – 28.02.2025
Разработка backend-приложения	01.03.2025 – 20.03.2025
Разработка frontend-интерфейса	21.03.2025 – 10.04.2025
Интеграция и тестирование	11.04.2025 – 30.04.2025
Оптимизация и доработка	01.05.2025 – 20.05.2025
Контейнеризация	21.05.2025 – 06.06.2025

7. Техническая документация, предоставляемая по окончанию работы

1) Схемы архитектуры системы в целом;
2) Схема структуры базы данных;
3) Описание API серверного приложения;
4) Описание классов серверного и клиентского приложений;
5) Информация о процессе отладки;
6) Исходный программный код.

Приложение Б

Приложение В

Листинг 1 — docker-compose.yml

```
services:
  db:
    image: postgres:14
    container_name: fastapi_postgres
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: fastapi_db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: fastapi_backend
    restart: always
```

```
    depends_on:
      - db
    ports:
      - "8000:8000"
    environment:
      DATABASE_URL: postgres://postgres:postgres@db:5432/fastapi_db
    command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
    volumes:
      - ./backend:/app

  frontend:
    image: python:3.12.3-slim
    container_name: frontend_http_server
    working_dir: /frontend
    volumes:
      - ./frontend:/frontend
    ports:
      - "8080:8080"
    command: python3 -m http.server 8080

volumes:
  postgres_data:
```

Листинг 2 — тест регистрации кандидата

```
@pytest.mark.asyncio
async def test_full_candidate_registration_flow(async_client: AsyncClient, async_session:
AsyncSession):
    test_data = {
        "user": {
            "email": "test.candidate@example.com",
            "password": "SecurePassword123!",
            "role": "CANDIDATE"
        },
```

```
    "profile": {
        "surname": "Иванов",
        "name": "Иван",
        "patronymic": "Иванович",
        "age": 30,
        "about_candidate": "Опытный разработчик",
        "education": "Высшее техническое"
    }
}
existing_user = await async_session.execute(
    select(User).where(User.email == test_data["user"]["email"])
)
assert existing_user.scalar_one_or_none() is None
response = await async_client.post(
    "/register/candidate/",
    json=test_data
)
assert response.status_code == status.HTTP_201_CREATED
response_data = response.json()
assert response_data["user"]["email"] == test_data["user"]["email"]
assert response_data["profile"]["name"] == test_data["profile"]["name"]
db_user = await async_session.execute(
    select(User).where(User.email == test_data["user"]["email"])
)
db_user = db_user.scalar_one()
assert db_user is not None
assert db_user.role == UserRole.CANDIDATE
assert verify_password(test_data["user"]["password"], db_user.password_hash)
db_profile = await async_session.execute(
    select(CandidateProfile).where(CandidateProfile.user_id == db_user.id)
)
db_profile = db_profile.scalar_one()
assert db_profile is not None
assert db_profile.surname == test_data["profile"]["surname"]
```

```
assert db_profile.age  == test_data["profile"]["age"]


duplicate_response  = await async_client.post(
    "/register/candidate/",
    json=test_data
)
assert duplicate_response.status_code  == status.HTTP_409_CONFLICT
assert "email  already  exists"  in duplicate_response.json()["detail"].lower()
```