

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Приложение по автоматизированному учету и
оценке персонала HR-специалистами**

Студент _____ Пудовкин Александр Павлович
(подпись, дата) (ФИО полностью)

Руководитель _____ Козлов А.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Пудовкин Александр Павлович гр. 21вп1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Приложение по автоматизированному учету и оценке
персонала HR-специалистами

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20____

2. Руководитель работы д.т.н профессор, зав. кафедры МОиПЭВМ Козлов А.Ю

3. Задание на работу (назначение разработки, исходные данные и т.п.) Разработать
веб-приложение для автоматизированного учёта и оценки персонала с использованием
Node.js, React и Microsoft SQL Server в среде разработки Visual Studio Code, обеспечивающее
ведение структуры организации, проведение процедур оценки, формирование отчётов и
уведомление пользователей, с возможностью многопользовательского доступа и интеграции
через REST API и Kafka

4. Перечень подлежащих разработке
вопросов 1.

Анализ предметной области и постановка задачи

2. Анализ требований на разработку

3. Разработка программных средств

4. Тестирование программных средств

5. Оформление ВКР

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 94 листа, 12 рисунков, 8 таблиц, 23 использованных источника, 2 приложения.

ВЕБ-ПРИЛОЖЕНИЕ, УЧЕТ ПЕРСОНАЛА, ОЦЕНКА СОТРУДНИКОВ, HRM, NODE.JS, REACT, MICROSOFT SQL SERVER, REST API, KAFKA, UML

Объектом разработки является веб-приложение для автоматизированного учета и оценки персонала, предназначенное для HR-специалистов.

Цель работы разработка и реализация информационной системы для автоматизированного учета и оценки персонала на основе микросервисной архитектуры, обеспечивающей ведение организационной структуры, проведение процедур оценки, формирование отчетов и уведомление пользователей.

Система представляет собой полнофункциональное приложение для автоматизированного учета и оценки персонала, реализованное на серверной части с использованием Node.js, Express.js и Microsoft SQL Server, а на клиентской — с помощью React, HTML5 и CSS3. Компоненты взаимодействуют через REST API и Apache Kafka, обеспечивая масштабируемость и устойчивость. Разработка велась в среде Visual Studio Code. В системе реализовано управление учетными записями с разграничением ролей (HR, руководители, сотрудники), проведение оценки персонала, автоматическая генерация отчетов, рассылка уведомлений участникам оценки, а также адаптивный интерфейс для работы на различных устройствах.

					ПГУ 09.03.04 – 08БР211.21 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Приложение по автоматизированному учету и оценке персонала HR-специалистами Пояснительная записка	Лит.	Лист	Листов
Разраб.		Пудовкин А.П.						
Пров.		Козлов А.Ю.					4	94
Н. контр.		Такташкин Д.В.				Группа 21ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ области исследования	9
1.1 Анализ существующих систем учета и оценки персонала	9
1.1.1 Обзор популярных HRM-систем	9
1.2 Методика оценки персонала	10
1.2.1 Метод 360 градусов.....	11
1.2.2 Метод KPI (Key Performance Indicators)	11
1.2.3 Метод 9-box grid (матрица талантов).....	12
1.2.4 Сравнительный анализ методик	12
1.3 Постановка задачи автоматизации	13
1.4 Проектирование бюджета и планирование сроков разработки	14
2 Проектирование приложения.....	16
2.1 Требования к приложению.....	16
2.1.1 Функциональные требования	16
2.1.2 Нефункциональные требования	17
2.2 Архитектура приложения.....	19
2.2.1 Микросервисная архитектура	19
2.2.2 Взаимодействие компонентов	21
2.2.3 Организация базы данных.....	22
2.3 Моделирование предметной области.....	24
2.4 Выбор технологий и обоснование	32
2.5 Проектирование интерфейса пользователя	33
2.5.1 Основные экраны приложения	33
2.5.2 Принципы UI/UX	35
3 Реализация приложения	37
3.1 Разработка базы данных	37
3.2 Реализация веб-приложения	39
3.2.1 Реализация интерфейса пользователя.....	39

3.2.2 Реализация серверной части приложения	42
3.2.3 Реализация взаимодействия между компонентами	43
4 Тестирование	46
4.1 Функциональное тестирование.....	46
4.1.1 Тестирование отправки уведомлений.....	46
4.1.2 Тестирование авторизации в системе	47
4.1.3 Тестирование получения анкеты для оценки.....	48
4.1.4 Тестирование создания оценки сотрудника.	49
4.1.5 Тестирование получения анкеты для оценки.....	49
4.2 Анализ метрик кода	50
4.3 Системное тестирование	50
Заключение	52
Список используемых источников.....	54
Приложение А (обязательное) Код приложения	57
Приложение Б (рекомендуемое) Изображения.....	92

Введение

В условиях цифровизации бизнеса и повышения конкуренции на рынке труда возрастает значение эффективного управления человеческими ресурсами [1]. Компании стремятся не только фиксировать кадровые данные, но и проводить регулярную оценку эффективности сотрудников, формировать кадровый резерв, поддерживать мотивацию персонала. Однако многие организации до сих пор используют ручные или фрагментарные методы, такие как Excel-таблицы и неинтегрированные решения, что ведёт к потерям времени, ошибкам и низкой прозрачности HR-процессов.

Разработка современного автоматизированного веб-приложения для учёта и оценки персонала позволяет повысить управляемость персоналом, улучшить внутреннюю коммуникацию и ускорить принятие кадровых решений. Это определяет актуальность выбранной темы.

Цель работы – разработать и реализовать информационную систему для автоматизированного учёта и оценки персонала на основе микросервисной архитектуры.

Для достижения цели необходимо решить следующие задачи:

- проанализировать существующие HRM-системы и методики оценки персонала;
- сформировать требования к разрабатываемому приложению;
- спроектировать архитектуру и структуру базы данных;
- реализовать микросервисы, интерфейс и механизмы взаимодействия компонентов;
- провести тестирование, оценить надёжность и функциональность системы.

Объектом исследования является процесс цифровизации кадрового учёта и оценки персонала в организациях.

Предметом исследования выступает разработка архитектуры и программной реализации веб-приложения, предназначенного для

автоматизации HR-процессов, включая хранение информации о сотрудниках, проведение процедур оценки и формирование аналитических отчётов.

В процессе выполнения работы применялись методы системного анализа, объектно-ориентированного проектирования, прототипирования пользовательского интерфейса, а также методы функционального и модульного тестирования. Использовались технологии веб-разработки (Node.js, React, Express.js, Kafka), методы моделирования (UML, ER-диаграммы) и инструменты визуализации.

Дипломная работа состоит из введения, четырёх глав, заключения и списка использованных источников:

- в первой главе рассматриваются существующие HR-системы и методики оценки персонала, формулируется задача автоматизации;
- вторая глава посвящена проектированию архитектуры приложения, описанию требований, моделированию базы данных и интерфейса;
- в третьей главе представлена реализация компонентов системы, включая интерфейс, серверную часть и взаимодействие между сервисами;
- в четвёртой главе проводится тестирование, анализируются результаты и метрики качества;
- работа сопровождается иллюстрациями, диаграммами, листингами кода и результатами тестов.

Разработанное приложение может быть использовано в организациях малого и среднего бизнеса для упрощения процедур кадрового учёта и оценки сотрудников. Система легко масштабируется, поддерживает расширение методик оценки и может быть развёрнута как внутрикорпоративное решение. Практическая значимость заключается в снижении трудозатрат на проведение оценки, повышении прозрачности и системности HR-деятельности.

1 Анализ области исследования

1.1 Анализ существующих систем учета и оценки персонала

1.1.1 Обзор популярных HRM-систем

На современном рынке информационных систем автоматизации управления персоналом (HRM-систем) существует широкий спектр решений, предлагающих различные функциональные возможности и ориентированных на бизнесы разного масштаба [2]. Ниже рассмотрены наиболее распространённые программные продукты, применяемые в российских и зарубежных организациях.

SAP SuccessFactors – корпоративная облачная HR-платформа, предоставляющая комплексный набор инструментов для управления жизненным циклом сотрудника. Система поддерживает процессы найма, адаптации, оценки, обучения и развития, а также планирования кадрового резерва и управления эффективностью. Отличительной чертой является высокая степень интеграции с другими модулями SAP и поддержка международных HR-стандартов. Тем не менее, внедрение системы требует значительных затрат и высококвалифицированного персонала.

1С:Зарплата и управление персоналом (1С:ЗУП) – один из лидеров российского рынка, предлагающий широкие возможности по учёту кадров, расчёту заработной платы, оформлению кадровых документов, ведению штатного расписания и формированию регламентированной отчётности. В последних версиях присутствуют функции по управлению компетенциями и оценке персонала. Однако система в основном ориентирована на учётные процессы, а возможности стратегического управления персоналом и гибкой оценки в ней ограничены.

BambooHR – облачная HRM-система, ориентированная на малые и средние компании. Обеспечивает ведение базы сотрудников, управление отпусками, отслеживание времени, проведение опросов и обратной связи. Поддерживает базовые процедуры оценки эффективности. Основным

недостатком для российских организаций является англоязычный интерфейс и отсутствие адаптации под местные нормативные требования.

OrangeHRM – система с открытым исходным кодом, предназначенная для автоматизации HR-процессов. Включает модули управления персоналом, отпусками, посещаемостью и производительностью. Поддерживает настройку процессов оценки и отчётности. Достоинством является гибкость и возможность кастомизации, однако функциональность базовой версии ограничена, а интеграция требует дополнительных ресурсов.

Битрикс24 – многофункциональная корпоративная платформа, включающая инструменты CRM, управления проектами и коммуникациями. HR-блок включает элементы кадрового документооборота, учета рабочего времени и базовую оценку сотрудников. Основным преимуществом является интеграция с другими бизнес-процессами компании. Недостатком может стать ограниченная глубина специализированных HR-функций.

WebSoft HRM – отечественные решения, ориентированные на комплексное управление персоналом, в том числе на процедуру оценки. Предоставляют инструменты для формирования анкет, организации 360-оценки, анализа компетенций и подготовки отчетов. Однако системы доступны только по коммерческим лицензиям и часто требуют долгой настройки под бизнес-процессы заказчика.

Изучив современные HRM-системы, можно заметить, что они обычно предлагают базовый набор функций (ведут кадровый учёт, помогают оценивать сотрудников, управляют обучением и следят за KPI). Однако существующие решения дорогие, сложные в настройке. Особенно это заметно в малом и среднем бизнесе, где нужна гибкость в оценке персонала.

1.2 Методика оценки персонала

Для эффективного управления персоналом важно не только вести кадровый учёт, но и понимать, как каждый сотрудник справляется со своими задачами. Оценка персонала помогает выявить сильные и слабые стороны,

наметить направления развития, принять обоснованные кадровые решения. Существует несколько методик оценки, которые применяются в организациях. Ниже рассмотрим наиболее популярные из них [3].

1.2.1 Метод 360 градусов

Метод оценки «360 градусов» основан на сборе обратной связи от разных участников рабочего процесса: коллег, подчинённых, руководителя и самого сотрудника. В результате формируется более объективная картина, так как учитываются мнения с разных точек зрения.

Каждому участнику предлагается заполнить анкету с вопросами, касающимися профессионального поведения, компетенций и взаимодействия. Оценка часто проводится анонимно, чтобы повысить откровенность.

Преимущества этого метода – глубина и многогранность оценки. Он позволяет сотруднику понять, как его воспринимают окружающие, а руководителю – увидеть поведенческие особенности, которые иначе могли бы остаться незамеченными. Недостаток – высокая трудоёмкость в организации и необходимость чёткой настройки анкет.

1.2.2 Метод KPI (Key Performance Indicators)

Метод оценки по ключевым показателям эффективности (KPI) строится на основе заранее заданных количественных и качественных целей, которые ставятся перед сотрудником. Примеры KPI – выполнение плана продаж, количество обработанных заявок, срок выполнения задач.

Этот способ позволяет легко отследить результаты деятельности и оценить вклад в общие цели компании. Он особенно хорошо работает в отделах с чёткими измеримыми результатами – продажах, производстве, обслуживании клиентов.

Плюс метода – чёткие и прозрачные критерии оценки. Минус – ограниченность: он не учитывает поведение, командную работу и развитие компетенций, а только конечный результат.

1.2.3 Метод 9-box grid (матрица талантов)

Метод «9-box grid» используется в основном для оценки потенциала и эффективности сотрудников в контексте карьерного планирования и формирования кадрового резерва. Сотрудников размещают в матрице 3×3, где по одной оси оценивается текущая эффективность, а по другой – потенциал развития.

Например, сотрудник с высокой эффективностью и высоким потенциалом попадает в верхнюю правую ячейку – это кандидат на продвижение. А человек с низкой эффективностью и низким потенциалом – в нижнюю левую, что может сигнализировать о необходимости переобучения или замены.

Метод удобен для стратегического планирования, но требует качественной предварительной оценки – часто с использованием других методов, таких как 360 или KPI.

1.2.4 Сравнительный анализ методик

Каждая из рассмотренных методик имеет свои особенности и применяется в зависимости от целей оценки. В таблице 1 приведено краткое сравнение.

Таблица 1 – Сравнение методик

Метод	Оценка поведения	Оценка результатов	Подходит для	Недостатки
360 градусов	Да	Частично	Развитие, обратная связь	Сложность, возможная субъективность
KPI	Нет	Да	Бизнес-цели, контроль результатов	Не учитывает «мягкие» навыки
9-box grid	Да	Да	Планирование карьеры, резервы	Требует комплексной информации

Таким образом, для комплексной оценки персонала рекомендуется комбинировать методы. Например, использовать KPI для оценки текущей эффективности, 360 градусов – для понимания поведенческих характеристик, а 9-box – для стратегических кадровых решений. Именно такой подход позволяет учитывать и результат, и потенциал, и качество взаимодействия сотрудников в команде.

1.3 Постановка задачи автоматизации

Современные организации стремятся эффективно управлять своими кадрами, чтобы развивать бизнес и повышать производительность. Однако на практике многие HR-процессы до сих пор выполняются вручную или с помощью устаревших инструментов – Excel-таблиц, электронной почты или разрозненных программ. Это снижает прозрачность, увеличивает трудозатраты и создает риски ошибок.

Особенно остро стоит вопрос автоматизации оценки персонала. Такие процедуры, как метод 360 градусов, требуют сбора данных от большого количества участников, подготовки анкет, контроля сроков и обработки результатов. Без автоматизированного решения эти процессы становятся долгими и неудобными как для HR-менеджеров, так и для сотрудников.

Кроме того, руководителям и специалистам по персоналу нужны удобные отчеты и аналитика, чтобы быстро принимать решения по развитию сотрудников, формированию кадрового резерва и реорганизации подразделений. Поэтому важна не только фиксация данных, но и их структурированный анализ.

В рамках данной дипломной работы ставится задача разработать веб-приложение, которое автоматизирует процесс учета и оценки персонала, включая:

- хранение информации о сотрудниках, их должностях и структурных подразделениях;
- организацию процедур оценки;

- автоматическую обработку результатов с возможностью генерации отчетов;
- удобный интерфейс для разных ролей – HR, руководителей, сотрудников и администраторов;
- обеспечение безопасности и разграничения доступа, чтобы каждый пользователь видел только то, что ему положено по роли.

Таким образом, автоматизация направлена на повышение эффективности HR-процессов, сокращение времени на обработку данных, улучшение качества обратной связи и усиление управляемости персоналом в организации. Предлагаемая система должна быть доступной, расширяемой и удобной как для небольших, так и для крупных компаний.

1.4 Проектирование бюджета и планирование сроков разработки

Для успешной реализации программного продукта необходимо заранее определить основные этапы разработки (таблица 2), оценить трудозатраты и временные ресурсы, а также спланировать бюджет [4]. Это позволяет контролировать выполнение проекта и эффективно распределять задачи между участниками.

Проект был реализован в индивидуальном порядке в рамках дипломного исследования, поэтому прямые финансовые затраты отсутствуют. Вместо этого акцент сделан на оценке трудозатрат по этапам разработки в человеко-часах и планировании календарных сроков. В качестве инструмента планирования применена диаграмма Ганта (рисунок 1), отражающая последовательность выполнения этапов, их длительность и взаимозависимость.

Таблица 2 – Основные этапы проекта

№	Этап разработки	Сроки выполнения	Трудозатраты (чел.-ч)
1	Анализ предметной области и ТЗ	01.03 – 05.03	16
2	Проектирование архитектуры	06.03 – 10.03	20

Продолжение таблицы 2

3	Разработка базы данных	11.03 – 17.03	24
4	Разработка микросервисов (backend)	18.03 – 05.04	50
5	Разработка frontend (React)	06.04 – 15.04	30
6	Интеграция и отладка компонентов	16.04 – 20.04	20
7	Тестирование, исправление ошибок	21.04 – 25.04	16
8	Подготовка отчётной документации	26.04 – 01.05	24

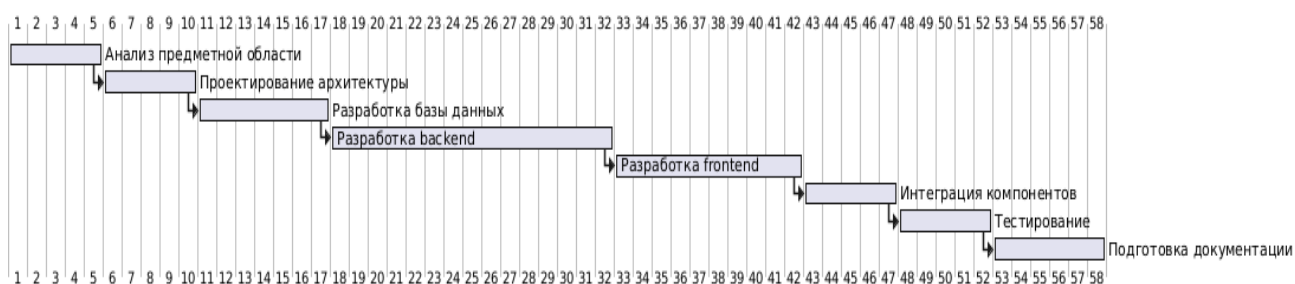


Рисунок 1 – Диаграмма Ганта

Таким образом, предложенный план-график охватывает все ключевые этапы проектирования и реализации информационной системы. Использование диаграммы Ганта позволило наглядно отразить последовательность работ, определить критические точки, а также оценить общий объём трудозатрат. Такой подход способствует эффективной организации индивидуальной разработки, повышает управляемость процессом и позволяет своевременно контролировать выполнение задач.

На основании приведённого графика можно сделать вывод о сбалансированности временного плана и его реалистичности в рамках дипломного проекта.

2 Проектирование приложения

2.1 Требования к приложению

2.1.1 Функциональные требования

Разрабатываемое программное средство предназначено для автоматизации процессов учета и оценки персонал. Оно должно обеспечивать выполнение функций, направленных на помощь в управлении персоналом, повышение эффективности процессов оценки и предоставление актуальной аналитической информации.

Приложение должно предоставлять механизмы создания и управления учетными записями пользователей с разграничением ролей доступа. Предусматривается наличие следующих категорий пользователей: специалисты по управлению персоналом (HR), руководители, сотрудники. Каждая учетная запись должна быть защищена аутентификацией по паролю.

Пользователю с ролью администратора (HR) должно быть доступно редактирование организационной структуры, включая создание, удаление и модификацию записей об отделах, должностях и сотрудниках. Должна быть обеспечена возможность привязки сотрудников к соответствующим подразделениям и позициям, а также отображение иерархии организации в виде древовидной структуры. Это необходимо для дальнейшего построения цепочек подчиненности, используемых при проведении оценки.

Система должна реализовывать механизм назначения процедур оценки. Ответственный пользователь (HR или руководитель) инициирует проведение оценки, определяет перечень оцениваемых лиц и назначает респондентов в различных ролях: самооценка, оценка коллег (равных), оценка со стороны руководителя и подчиненных.

Приложение должно автоматически уведомлять участников оценки о начале, сроках и этапах заполнения анкет. При наступлении контрольных сроков должны формироваться напоминания. Такой функционал позволяет

повысить вовлеченность респондентов и соблюдение временных рамок процедуры.

Каждый респондент заполняет индивидуальную анкету, данные которой сохраняются в базе данных с привязкой к конкретной процедуре, идентификатору оцениваемого лица и дате прохождения.

После завершения процедуры оценки система автоматически формирует отчетную документацию, включающую:

- индивидуальные отчеты для сотрудников с визуализацией результатов по каждой компетенции;
- групповые отчеты по подразделениям или должностям, содержащие обобщённые показатели;
- итоговый аналитический отчет для руководства, содержащий средние значения, сравнительный анализ и рекомендации.

Отчёты должны быть представлены в табличной и графической форме, поддерживать экспорт в распространённые офисные форматы.

2.1.2 Нефункциональные требования

Наряду с функциональными возможностями программного продукта важнейшее значение имеют его нефункциональные характеристики, которые определяют удобство использования, производственную пригодность и способность к развитию в условиях изменяющихся требований [5].

Программа должна быть удобной для пользователя. Интерфейс должен быть понятным с первого взгляда, без лишних деталей. Все элементы должны быть расположены логично и выглядеть одинаково по стилю. Пользователь должен легко находить нужные функции и выполнять действия с минимальным количеством шагов. Вся работа с программой должна быть простой и приятной.

Особое внимание следует уделить адаптивности дизайна, позволяющей автоматически изменять внешний вид интерфейса в зависимости от характеристик устройства пользователя, разрешения экрана и ориентации. Это

необходимо для обеспечения корректного отображения и функционирования системы как на настольных компьютерах, так и на мобильных устройствах.

В целях обеспечения сохранности информации, связанной с персоналом и результатами оценок, система должна использовать надежную реляционную систему управления базами данных (СУБД). В качестве хранилища информации используется Microsoft SQL Server, обеспечивающий поддержку транзакционности, целостности данных, а также возможность реализации резервного копирования и восстановления.

Применение транзакций помогает избежать ошибок, когда несколько пользователей одновременно меняют данные. Это делает работу с данными безопаснее. Также нужно защищать данные от поломок компьютера. Для этого важно регулярно делать резервные копии и использовать надёжные системы, которые продолжают работать даже при сбоях.

Кроме того, требуется реализация шифрования паролей с использованием криптографических хеш-функций (например, алгоритмов SHA-256 или bcrypt), защита от SQL-инъекций и иных типов атак на уровне клиент-серверного взаимодействия. Для обеспечения безопасности данных при передаче может использоваться протокол HTTPS с TLS.

Архитектура программного продукта должна обеспечивать возможность масштабирования и расширения функционала без необходимости полной переработки системы. В частности, предусмотрена возможность добавления новых методик оценки персонала (например, KPI-анализ, модель 9-box grid, ассессмент-центры и др.) за счёт модульной структуры и слабой связанности компонентов.

Такой подход позволяет адаптировать систему под различные сценарии оценки в зависимости от текущих потребностей организации, а также упрощает внесение изменений в логику и интерфейс отдельных микросервисов без затрагивания остальных.

Система должна быть реализована в виде веб-приложения с доступом через браузер, что обеспечивает независимость от операционной системы и

типа устройства конечного пользователя. Приложение должно корректно функционировать в современных версиях браузеров (Google Chrome, Mozilla Firefox, Microsoft Edge и др.), поддерживать стандарты HTML5/CSS3 и не требовать установки дополнительных плагинов или компонентов.

Таким образом, достигается высокая степень доступности системы, что особенно актуально при наличии удалённых сотрудников и распределённых офисов.

2.2 Архитектура приложения

2.2.1 Микросервисная архитектура

Для реализации приложения выбрана микросервисная архитектурная модель, представляющая собой совокупность слабо связанных между собой компонентов, каждый из которых выполняет строго определённый набор функций и может разрабатываться, масштабироваться и сопровождаться независимо от других [6]. Такой подход полностью соответствует современным требованиям к разработке высоконагруженных и гибких корпоративных информационных систем.

К числу ключевых преимуществ микросервисной архитектуры относятся:

- масштабируемость – возможность горизонтального масштабирования отдельных компонентов системы в зависимости от текущей нагрузки;
- изоляция ответственности – чёткое разграничение функциональных зон, что снижает риски, связанные с ошибками в отдельных модулях;
- гибкость сопровождения – возможность обновления, переиспользования или замены отдельных микросервисов без остановки всей системы;
- технологическая независимость – выбор оптимальных технологий и инструментов разработки для каждого микросервиса с учётом его специфики и требований.

В таблице 3 представлены микросервисы, которые используются в разрабатываемом приложении.

Таблица 3 – Назначения микросервисов

Микросервис	Назначение
Auth-service	Обеспечивает функции регистрации, аутентификации и авторизации пользователей с учётом ролевой модели доступа. Реализует шифрование паролей, выдачу токенов доступа и верификацию.
User-service	Осуществляет управление данными пользователей, включая сотрудников, руководителей, HR-менеджеров. Хранит информацию о подразделениях, должностях, ролях и связях между сущностями.
Evaluation-service	Реализует основную бизнес-логику процедуры оценки персонала по методике 360 градусов. Управляет анкетами, участниками оценки, результатами и сроками прохождения.
Report-service	Формирует отчётность различного уровня (по сотрудникам, отделам, всей организации). Поддерживает экспорт данных в форматы Excel и Word, а также визуализацию аналитики.
Notification-service	Отвечает за отправку уведомлений (внутрисистемных и по электронной почте), включая напоминания, приглашения и уведомления об окончании процедуры оценки.
Frontend	Web-приложение, реализующее пользовательский интерфейс. Взаимодействует с микросервисами через REST API, обеспечивает интуитивный UX-дизайн.

Все микросервисы взаимодействуют между собой через стандартизированные HTTP/REST-интерфейсы с использованием JSON в качестве формата обмена данными. Для хранения данных в каждом микросервисе используется независимая схема в общей базе данных Microsoft SQL Server, что обеспечивает логическую изоляцию и упрощает контроль над миграциями данных.

Коммуникация между микросервисами может быть дополнительно реализована с использованием асинхронного обмена сообщениями через

брокеры сообщений (например, Apache Kafka) при необходимости масштабирования и обработки событий в реальном времени.

Выбор микросервисной архитектуры – правильное решение для современной HR-системы. Такой подход дает нам гибкость и надежность, которые так важны для корпоративного софта. Мы не только сможем реализовать все нужные функции прямо сейчас, но и без проблем добавим новые возможности в будущем. А еще это позволит легко подключать разные внешние системы, будь то кадровые базы или аналитические инструменты.

2.2.2 Взаимодействие компонентов

Взаимодействие между компонентами разрабатываемой информационной системы построено на основе принципов слабо связанной архитектуры, характерной для микросервисных решений. Основное внимание при проектировании взаимодействий уделяется масштабируемости, отказоустойчивости и минимизации связности между сервисами, что позволяет эффективно управлять жизненным циклом каждого компонента, обеспечивая при этом целостность системы в целом.

Коммуникация между микросервисами осуществляется преимущественно посредством синхронного обмена сообщениями через интерфейсы REST (Representational State Transfer) с использованием формата JSON. REST-протокол является де-факто стандартом взаимодействия в распределённых системах и обеспечивает простоту интеграции, читаемость данных и поддержку HTTP-методов (GET, POST, PUT, DELETE и др.), что соответствует рекомендациям по разработке API в распределённых системах.

Для реализации асинхронного обмена сообщениями между сервисами, обработки событий и задач, не критичных к времени выполнения, возможно применение брокеров сообщений, таких как Apache Kafka [7]. Данный подход обеспечивает буферизацию, масштабирование по потреблению и устойчивость к временной недоступности отдельных компонентов, что особенно актуально при обработке уведомлений и генерации отчётов.

В соответствии с принципом изоляции данных, каждый микросервис взаимодействует с собственной схемой базы данных, развернутой на едином экземпляре СУБД Microsoft SQL Server. Такой подход обеспечивает логическое разделение хранимых данных и предотвращает прямую зависимость между сервисами на уровне хранилища, что повышает устойчивость системы к изменениям и упрощает управление схемами.

Каждая схема включает собственный набор таблиц, индексов, представлений и процедур, необходимых исключительно для работы соответствующего микросервиса. Это способствует соблюдению принципов инкапсуляции и минимального доступа, а также упрощает реализацию миграций, резервного копирования и контроля версий данных.

Таким образом, реализованная модель взаимодействия компонентов обеспечивает высокую гибкость, масштабируемость и надёжность работы системы в условиях реальной эксплуатации. Выбор смешанного подхода к коммуникации (синхронного и асинхронного), централизованного API-шлюза и распределённого хранилища соответствует лучшим практикам проектирования микросервисных архитектур и подтверждает обоснованность архитектурного решения в рамках автоматизированной системы учёта и оценки персонала.

2.2.3 Организация базы данных

Одним из ключевых элементов архитектуры проектируемого программного обеспечения является подсистема хранения и обработки данных [8]. В разрабатываемом приложении применяется Microsoft SQL Server [9]. Данная платформа обладает поддержкой транзакционной обработки, разграничения прав доступа, механизмов резервного копирования и восстановления, а также широкими возможностями для анализа и агрегирования данных, что делает её целесообразным выбором для корпоративного уровня приложений.

Организация хранения данных реализована на основе архитектурного подхода Database-per-Service, характерного для микросервисных систем.

Согласно данному принципу, каждый микросервис использует собственную, изолированную от других сервисов схему базы данных, а при необходимости - полностью отдельную БД. Такой подход обеспечивает слабую связанность компонентов системы, способствует автономности их развития, тестирования и масштабирования, а также минимизирует риски, связанные с отказами отдельных сервисов.

Прямой обмен данными между базами исключён. Взаимодействие между микросервисами осуществляется исключительно через стандартизированные интерфейсы прикладного программирования (API), что соответствует современным практикам построения распределённых приложений. Это повышает надёжность архитектуры и упрощает управление её компонентами.

Каждому микросервису в рамках системы сопоставляется определённая схема данных, содержащая соответствующие предметной области таблицы. Пример распределения объектов хранения данных по микросервисам представлен в таблице 4.

Таблица 4 – Микросервисы и схемы баз данных

Микросервис	Схема в БД / Таблицы
User-service	employees, departments, positions
Evaluation-service	evaluations, forms, responses
Report-service	reports, metrics
Auth-service	users, roles, permissions

Сервис управления пользователями (User-service) реализует функции управления персоналом и организационной структурой. В его схему входят таблицы employees (сотрудники), departments (структурные подразделения) и positions (должности). Сервис оценки персонала (Evaluation-service) отвечает за реализацию методики оценки 360 градусов. Он использует таблицы evaluations (сессии оценки), forms (оценочные анкеты) и responses (ответы участников). Сервис отчетности (Report-service) агрегирует и анализирует результаты оценочных мероприятий. Основными таблицами являются reports (сформированные отчёты) и metrics (ключевые показатели оценки). Сервис

аутентификации и авторизации (Auth-service) обеспечивает контроль доступа, идентификацию и разграничение прав пользователей. Для этого используются таблицы users (учётные записи), roles (роли пользователей) и permissions (права доступа).

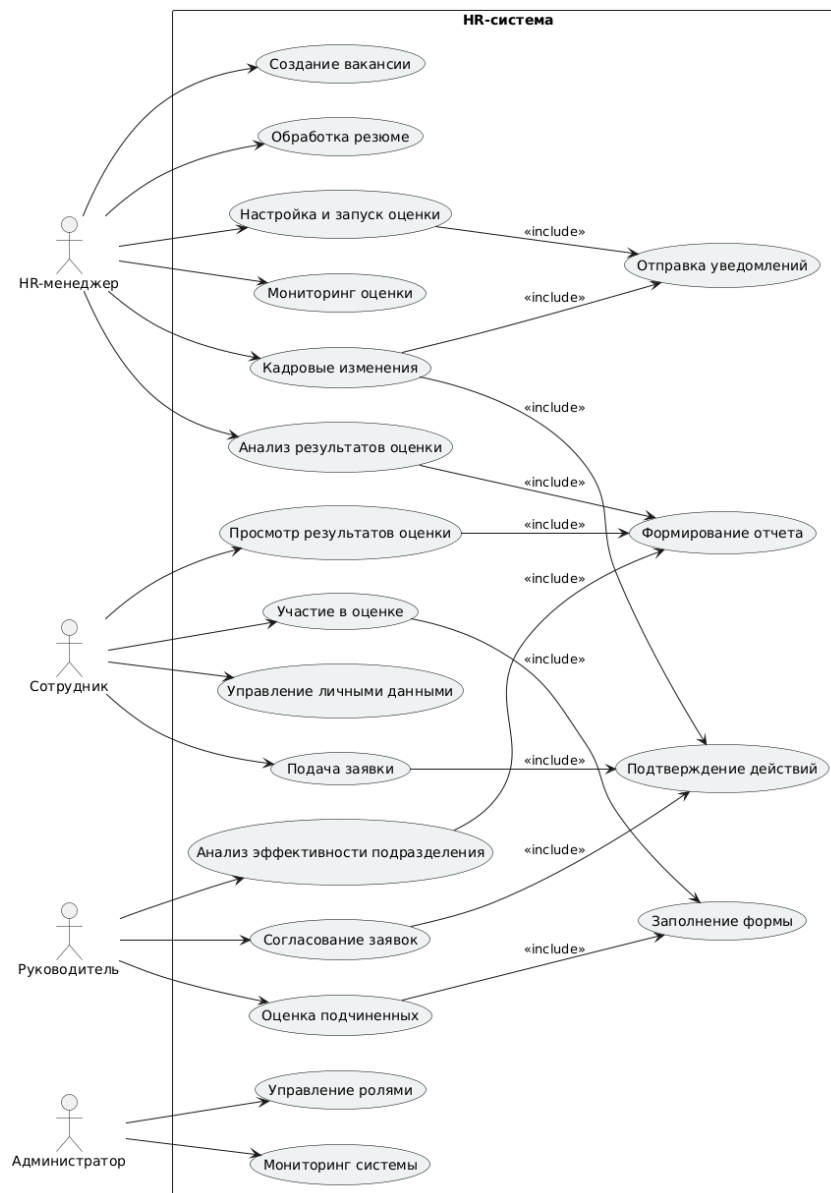
Каждая схема разрабатывается автономно, с учётом требований конкретного микросервиса. Такая декомпозиция способствует упрощению обслуживания системы, повышает устойчивость к ошибкам, возникающим в отдельных сервисах.

Предложенная организация базы данных отвечает требованиям к современным корпоративным информационным системам. Использование Microsoft SQL Server совместно с архитектурным подходом Database-per-Service обеспечивает устойчивость, масштабируемость и гибкость решения. Изоляция микросервисов на уровне базы данных способствует упрощению сопровождения системы, повышает отказоустойчивость и облегчает интеграцию дополнительных компонентов в будущем. Такое проектное решение можно считать обоснованным и соответствующим как практическим задачам разработки, так и научно-техническим рекомендациям в области построения распределённых систем.

2.3 Моделирование предметной области

Моделирование предметной области представляет собой фундаментальный этап анализа и проектирования информационной системы, направленный на формализацию ключевых сущностей, их взаимосвязей и сценариев взаимодействия пользователей с системой. В рамках рассматриваемой HR-системы моделирование осуществляется посредством построения диаграммы прецедентов использования (Use Case) [10, 11]. Данный подход способствует точному определению требований к программному обеспечению, облегчает коммуникацию между заказчиками и разработчиками, а также обеспечивает прозрачность. В последующих подразделах приводится детализированное описание ролей пользователей и основных сценариев

использования, реализующих процессы подбора, оценки и администрирования персонала.



В результате анализа требований определены четыре основные роли пользователей системы:

- HR-менеджер – ключевой пользователь системы, ответственный за создание вакансий, обработку резюме, настройку и запуск оценки персонала, мониторинг хода оценки, анализ ее результатов, а также кадровые изменения;
- сотрудник – участник кадровой системы, который управляет личными данными, подает заявки на вакансии, принимает участие в оценочных процедурах и просматривает результаты оценки;
- руководитель – пользователь, осуществляющий согласование заявок, оценку подчиненных и анализ эффективности подразделения;
- администратор – пользователь, отвечающий за управление ролями и мониторинг технического состояния системы.

На основе анализа требований и ролей пользователей определены следующие ключевые сценарии использования системы:

- создание вакансии – HR-менеджер формирует новые вакансии в системе, задает требования и параметры;
- обработка резюме – HR обрабатывает поступившие резюме для дальнейшего подбора персонала;
- настройка и запуск оценки – HR запускает процедуру оценки сотрудников, включая подготовку и распределение анкет;
- мониторинг оценки – контроль хода заполнения анкет и участия в оценке;
- анализ результатов оценки – анализ полученных данных оценки для принятия кадровых решений;
- кадровые изменения – внесение изменений в организационную структуру и штатное расписание;
- управление личными данными – сотрудники редактируют и обновляют свои персональные данные;
- подача заявки – сотрудники подают заявки на вакансии;
- просмотр результатов оценки – просмотр итоговых отчетов и аналитики;

– оценка подчиненных – проведение оценки сотрудниками их непосредственных подчиненных;

– управление ролями – администратор настраивает права доступа и роли пользователей;

На рисунке 3 представлена диаграмма классов (Class Diagram), отражающая ключевые сущности предметной области системы управления персоналом, реализованной на микросервисной архитектуре [12].

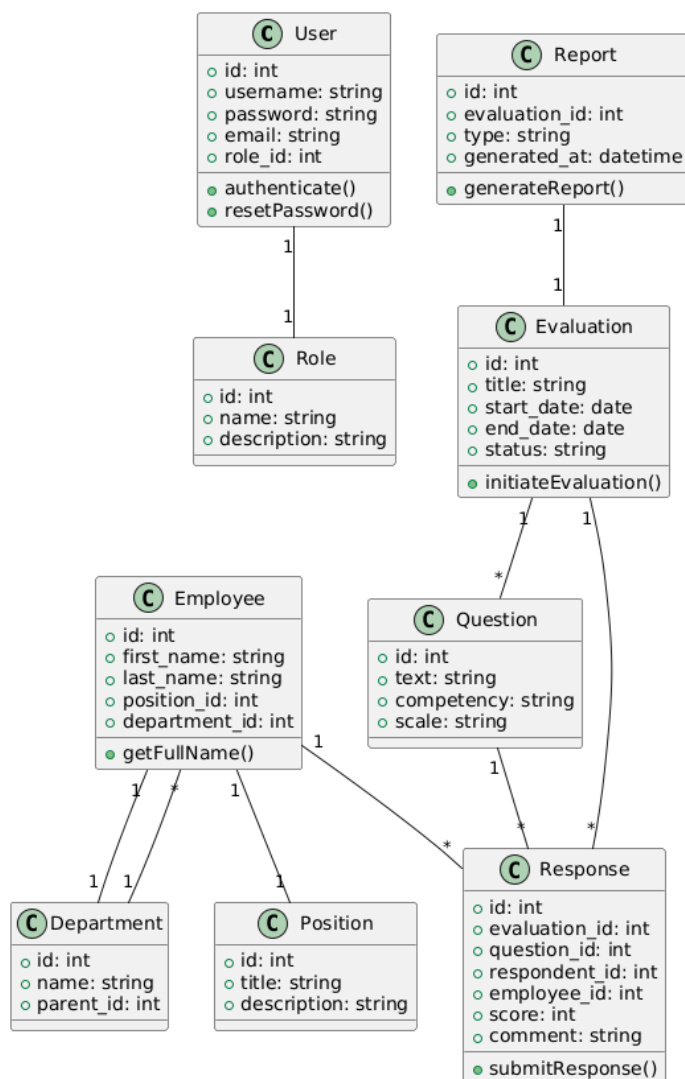


Рисунок 3 – Диаграмма классов

Диаграмма классов представляет основные сущности системы и их взаимосвязи. Ключевые классы: **User** – представляет учетные записи пользователей с атрибутами: id, username, password, email, role_id. Методы: authenticate() и resetPassword(); **Role** – роли пользователей (HR, руководитель,

сотрудник) с атрибутами: id, name, description; Employee – сотрудники организации с атрибутами: id, first_name, last_name, position_id, department_id. Метод getFullName() возвращает полное имя; Department – структурные подразделения с атрибутами: id, name, parent_id (для иерархии); Position – должности сотрудников с атрибутами: id, title, description. Evaluation – процедуры оценки с атрибутами: id, title, start_date, end_date, status; Метод initiateEvaluation() запускает оценку; Question – вопросы анкет с атрибутами: id, text, competency, scale; Response – ответы респондентов с атрибутами: id, evaluation_id, question_id, respondent_id, employee_id, score, comment. Метод submitResponse() сохраняет ответ; Report – отчеты с атрибутами: id, evaluation_id, type, generated_at. Метод generateReport() формирует отчет.

Между классами определены следующие связи:

- «Пользователь» (User) связан с одной ролью (Role);
- «Сотрудник» (Employee) связан с одной должностью (Position) и одним подразделением (Department);
- «Подразделение» (Department) может содержать множество сотрудников (Employee);
- «Оценка» (Evaluation) включает множество вопросов (Question) и ответов (Response);
- «Вопрос» (Question) может иметь множество ответов (Response);
- «Сотрудник» (Employee) может быть участником множества оценок через ответы (Response);
- «Отчет» (Report) связан с одной оценкой (Evaluation).

Диаграмма последовательности, представленная на рисунке 4, описывающая процесс проведения оценки, демонстрирует временную и логическую последовательность взаимодействий между участниками системы и основными микросервисами в рамках одного бизнес-сценария [13, 14]. Этот процесс инициируется HR-менеджером, который через пользовательский интерфейс формирует параметры оценки, включая выбор оцениваемых

сотрудников, назначение ролей участников (оценщик, руководитель, коллега, подчинённый) и выбор анкет.

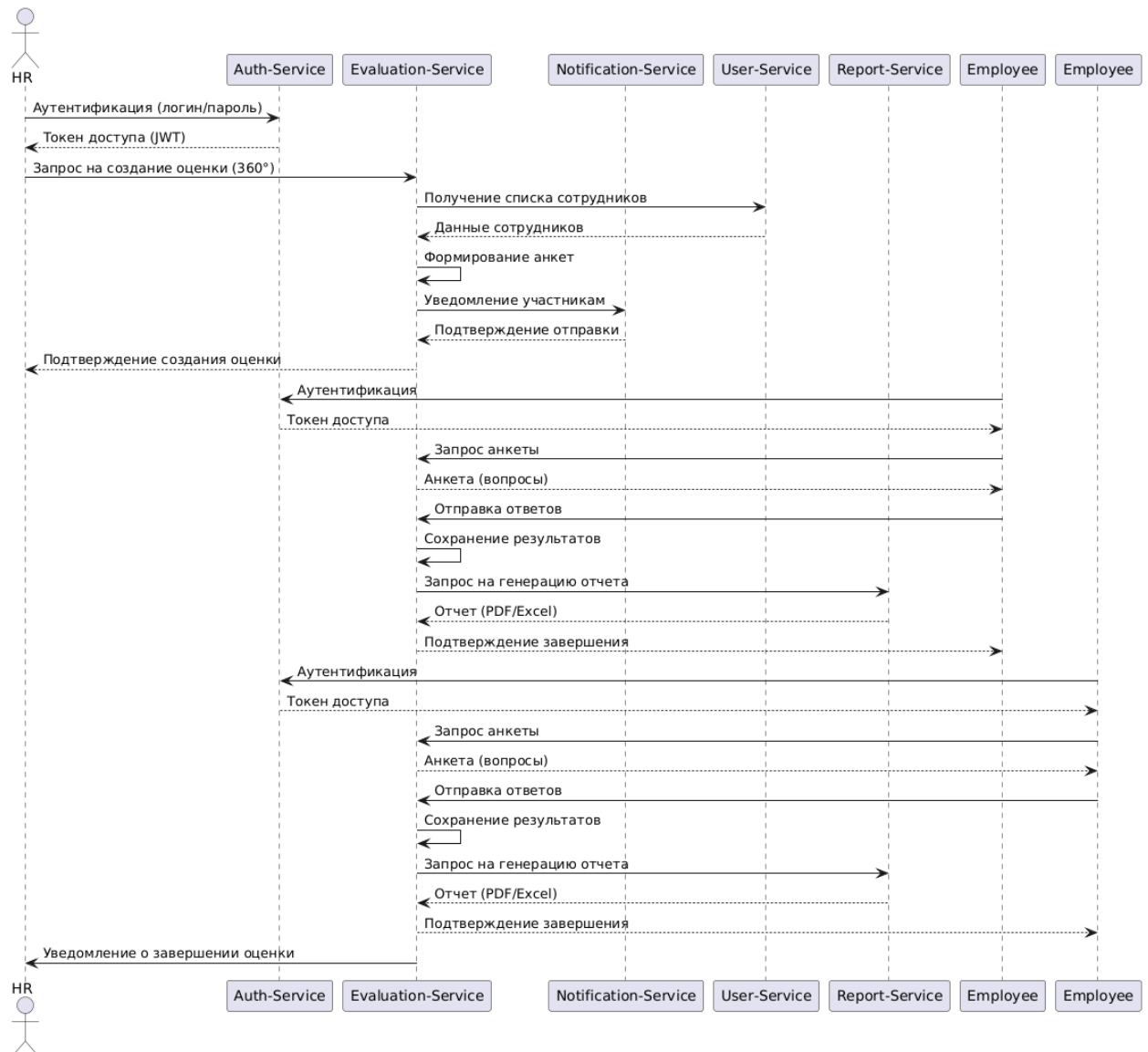


Рисунок 4 – Диаграмма последовательности

После задания параметров система направляет запросы к микросервисам, обеспечивающим управление пользователями и проведением оценок. На этапе инициализации создаются соответствующие сущности в базе данных, генерируются индивидуальные формы оценки, а затем активируется механизм рассылки уведомлений. Notification-сервис информирует участников о необходимости прохождения процедуры в установленный срок.

Участники оценки получают доступ к персонализированным анкетам и поочередно заполняют формы, данные из которых сохраняются в базе и в

дальнейшем используются для формирования отчётности. Завершается сценарий анализом полученных ответов и генерацией итогового отчёта, который доступен для просмотра HR-менеджером и другими заинтересованными сторонами, в зависимости от ролей и прав доступа.

Диаграмма последовательности иллюстрирует процесс проведения оценки с участием HR-менеджера, сотрудников и микросервисной архитектуры системы. Процесс начинается с аутентификации HR-менеджера: он входит в систему через Auth-Service и получает JWT-токен для дальнейшего доступа к защищённым ресурсам. Далее HR-менеджер инициирует процедуру оценки, отправляя запрос на её создание в Evaluation-Service. Этот сервис, в свою очередь, обращается к User-Service для получения актуального списка сотрудников, после чего формирует персонализированные анкеты и передает данные в Notification-Service для автоматической рассылки уведомлений участникам оценки. По завершении всех этих шагов HR получает подтверждение об успешном создании процедуры.

На следующем этапе сотрудники системы (например, Employee1 и Employee2) проходят аналогичную процедуру аутентификации и обращаются к Evaluation-Service за получением своих индивидуальных анкет. После получения вопросов участники заполняют формы и отправляют ответы через тот же сервис. Evaluation-Service сохраняет полученные данные, а затем инициирует генерацию отчетов, отправляя соответствующий запрос в Report-Service. Сотрудники получают подтверждение о завершении этапа оценки.

Финальной стадией процесса является уведомление HR-менеджера со стороны Evaluation-Service о том, что все участники завершили прохождение процедуры, и результаты готовы к анализу. В архитектуре системы используются синхронные HTTP-запросы между микросервисами по REST API, аутентификация осуществляется с применением JWT-токенов, уведомления рассылаются асинхронно через Notification-Service, а отчеты формируются по запросу в режиме реального времени. Такой подход

обеспечивают согласованную работу всех компонентов системы и повышает эффективность оценки персонала.

Далее рассмотрим ER-модель базы данных (рисунок 5) [15, 16]. Она строится на основе анализа предметной области и требований к системе.

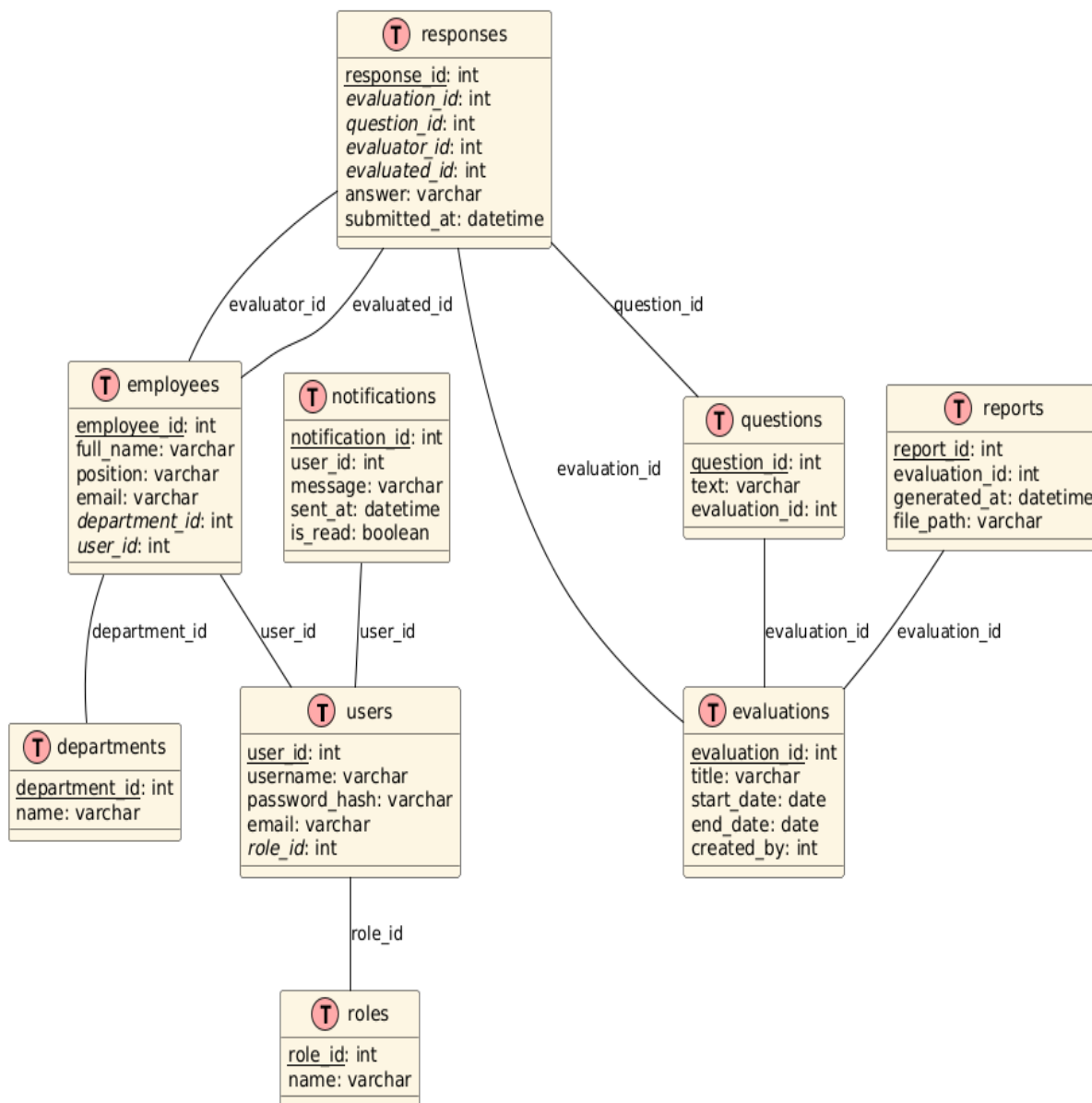


Рисунок 5 – ER-модель

В модели представлены основные таблицы:

- users – учетные записи пользователей системы;
- roles – роли, определяющие права доступа;
- employees – информация о сотрудниках организации;
- departments – структурные подразделения;

- evaluations – записи процедур оценки;
- questions – анкеты с вопросами;
- responses – ответы участников;
- reports – сгенерированные отчёты;
- notifications – сообщения и напоминания.

Связи между таблицами учитывают специфику предметной области: например, отношение многие-ко-многим между сотрудниками и оценками, что отражает участие одного сотрудника в нескольких оценках и участие нескольких сотрудников в одной оценке. Отношения один-ко-многим применяются, например, между отделами и сотрудниками.

Проведенное моделирование обеспечило четкое понимание структуры системы, взаимодействия ее компонентов и логики работы. Это позволило перейти к этапу реализации с полным набором требований и продуманной архитектурой, что в итоге способствовало созданию решения для автоматизации HR-процессов.

2.4 Выбор технологий и обоснование

Выбор технологического стека обусловлен требованиями к масштабируемости, расширяемости и безопасности системы. Применяемые программные средства соответствуют современным стандартам разработки информационных систем, обеспечивают модульность, поддержку микросервисной архитектуры и удобство сопровождения. Выбранные технологии и их обоснование представлены в таблице 5.

Таблица 5 – Технология и обоснование ее выбора

Компонент	Выбранная технология	Обоснование выбора
Серверный язык программирования	Node.js [17, 18]	Обеспечивает асинхронную неблокирующую модель ввода-вывода, что критично для обработки большого

Продолжение таблицы 5

Компонент	Выбранная технология	Обоснование выбора
		количества одновременных запросов. Имеет развитую экосистему и поддержку модульной архитектуры.
Веб-интерфейс	HTML5, CSS3, JavaScript	Использование стандартных веб-технологий обеспечивает быстрое создание адаптивного и кроссбраузерного интерфейса пользователя.
Фреймворк для сервера	Express.js	Лёгкий и гибкий фреймворк для создания REST API на платформе Node.js. Упрощает маршрутизацию, валидацию и обработку запросов.
Система управления базами данных (СУБД)	Microsoft SQL Server	Корпоративная реляционная СУБД, обеспечивающая высокую надёжность, поддержку транзакций, масштабируемость и мощные средства аналитики. Хорошо интегрируется с .NET и обладает развитой системой безопасности.

Выбранный набор технологий представляет собой сбалансированное решение, которое не только удовлетворяет текущим требованиям проекта, но и предоставляет хороший задел для будущего масштабирования системы. Сочетание производительности, надежности и удобства разработки делает этот стек оптимальным для создания современных корпоративных HR-приложений.

2.5 Проектирование интерфейса пользователя

2.5.1 Основные экраны приложения

Пользовательский интерфейс системы представляет собой совокупность экранных форм, обеспечивающих доступ к доступному функционалу в соответствии с ролью субъекта. При проектировании интерфейсов учитывались принципы удобства использования продукта, адаптивности для различных

устройств, что соответствует требованиям, предъявляемым к современным корпоративным информационным системам.

Начальной точкой взаимодействия пользователя с системой является экран входа. Он предоставляет базовые функции, такие как ввод логина и пароля для аутентификации, а также доступ к форме регистрации нового пользователя. В случае успешной авторизации интерфейс адаптируется в зависимости от роли пользователя.

Для пользователей с расширенными правами – HR-менеджеров и администраторов – предусмотрена панель администратора/HR. Она включает функции управления организационной структурой (отделами, должностями, списком сотрудников), создание и запуск процедур оценки, мониторинг процесса заполнения анкет, а также возможность выгрузки аналитических отчетов в формате PDF или Excel. Такой подход позволяет специалистам по персоналу комплексно управлять оценочными и кадровыми процессами.

Форма оценки реализована как основной инструмент для прохождения процедур оценки сотрудниками. Она включает оценочные шкалы (например, от 1 до 5 баллов), группы вопросов, разделенные по категориям компетенций, поведения, эффективности и другим критериям. Управление элементами выполнено с учетом принципов минималистичного дизайна, а сам интерфейс адаптируется под различные устройства, включая мобильные.

Просмотр результатов осуществляется через экран аналитики, позволяющий пользователю видеть индивидуальные и групповые показатели оценки в виде таблиц и диаграмм (в том числе круговых и линейных). Реализована также функция фильтрации по различным параметрам: период, отдел, должность и т. д. Пользователи могут скачивать итоговые отчеты в удобных форматах и получать рекомендации для дальнейшего профессионального развития.

Для обычных сотрудников предусмотрена персональная панель, из которой доступен список активных процедур оценки, история предыдущих оценок, результаты и аналитика. Также пользователи получают персональные

уведомления и могут редактировать свои личные данные при необходимости. Интерфейс поддерживает адаптивность, включая мобильные версии, и позволяет получать уведомления о событиях в системе в режиме реального времени.

2.5.2 Принципы UI/UX

Разработка пользовательского интерфейса (UI) и опыта взаимодействия (UX) в рамках системы управления персоналом осуществляется с учетом современных требований эргономики, доступности и ролевой дифференциации [19, 20]. Основной целью проектирования является обеспечение высокой удобства использования, снижения когнитивной нагрузки на пользователя и повышения эффективности выполнения бизнес-операций.

Одним из ключевых принципов проектирования интерфейса является простота. Пользовательский путь оптимизирован до минимального количества действий: от 1 до 3 кликов до основной функции. Формы построены по принципу минимализма – только необходимые поля, лаконичные подписи и подсказки.

Интуитивность навигации достигается за счёт логичной структуры экранов, последовательного расположения элементов управления и наличия повторяющихся интерфейсных паттернов. Пользователь с базовыми цифровыми навыками может легко адаптироваться в системе без предварительного обучения.

Для обеспечения визуальной согласованности всех компонентов интерфейса применяется единый графический стиль, основанный на использовании стандартизированных UI-компонентов. В рамках проекта реализована интеграция с CSS-фреймворком Bootstrap, который обеспечивает следующие преимущества:

- унифицированная цветовая палитра и типографика;
- повторное использование компонентов (кнопки, формы, модальные окна);

– визуальная иерархия, акцентирующая внимание на ключевых действиях.

Такая унификация повышает воспринимаемость интерфейса, снижает риск пользовательских ошибок и упрощает дальнейшее масштабирование системы.

Особое внимание в проектировании уделено адаптивной вёрстке, обеспечивающей корректное отображение интерфейса на различных устройствах (настольные ПК, планшеты, смартфоны). Используются современные технологии CSS – Flexbox и Grid Layout, позволяющие динамически адаптировать компоновку компонентов под доступную ширину экрана.

Реализовано масштабирование и изменение плотности интерфейса без потери функциональности и доступности. Это позволяет обеспечить непрерывный пользовательский опыт в различных условиях использования, в том числе при удалённой работе.

3 Реализация приложения

3.1 Разработка базы данных

Разработка базы данных выполнена с применением реляционной модели, развернутой в среде управления базами данных Microsoft SQL Server. Каждому микросервису соответствует изолированная схема данных, реализующая подход Database-per-Service. Это повышает модульность, отказоустойчивость и масштабируемость системы.

В качестве средства взаимодействия между прикладной логикой и СУБД используется объектно-реляционное отображение (ORM) Sequelize, что обеспечивает типизированный доступ к данным и упрощает миграцию схем.

Ниже представлен SQL-скрипт (листинг 1) для создания таблиц по описанной ER-модели в Microsoft SQL Server (смотри рисунок 4). Все внешние ключи определены, типы данных подобраны с учетом назначения, включены первичные ключи и основные связи.

Листинг 1 – SQL-скрипты для создания таблиц

```
CREATE TABLE Roles (  
    role_id INT PRIMARY KEY IDENTITY(1,1),  
    name NVARCHAR(100) NOT NULL  
);  
  
CREATE TABLE Users (  
    user_id INT PRIMARY KEY IDENTITY(1,1),  
    username NVARCHAR(100) NOT NULL UNIQUE,  
    password_hash NVARCHAR(255) NOT NULL,  
    email NVARCHAR(255) NOT NULL UNIQUE,  
    role_id INT NOT NULL,  
    FOREIGN KEY (role_id) REFERENCES Roles(role_id)  
);  
  
CREATE TABLE Departments (  
    department_id INT PRIMARY KEY IDENTITY(1,1),  
    name NVARCHAR(100) NOT NULL  
);  
  
CREATE TABLE Employees (  
    employee_id INT PRIMARY KEY IDENTITY(1,1),
```

```

        full_name NVARCHAR(255) NOT NULL,
        position NVARCHAR(100),
        email NVARCHAR(255) UNIQUE,
        department_id INT,
        FOREIGN KEY (department_id) REFERENCES
Departments(department_id)
);

CREATE TABLE Evaluations (
    evaluation_id INT PRIMARY KEY IDENTITY(1,1),
    title NVARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    created_by INT NOT NULL,
    FOREIGN KEY (created_by) REFERENCES Users(user_id)
);

CREATE TABLE Questions (
    question_id INT PRIMARY KEY IDENTITY(1,1),
    text NVARCHAR(1000) NOT NULL,
    evaluation_id INT NOT NULL,
    FOREIGN KEY (evaluation_id) REFERENCES
Evaluations(evaluation_id)
);

CREATE TABLE Response (
    response_id INT PRIMARY KEY IDENTITY(1,1),
    evaluation_id INT NOT NULL,
    question_id INT NOT NULL,
    evaluator_id INT NOT NULL,      -- Кто оценивает
    evaluated_id INT NOT NULL,      -- Кого оценивают
    answer INT NOT NULL,            -- Балл (например, от 1 до 5)
    submitted_at DATETIME DEFAULT GETDATE(),
    FOREIGN KEY (evaluation_id) REFERENCES
Evaluations(evaluation_id),
    FOREIGN KEY (question_id) REFERENCES Questions(question_id),
    FOREIGN KEY (evaluator_id) REFERENCES Employees(employee_id),
    FOREIGN KEY (evaluated_id) REFERENCES Employees(employee_id)
);

CREATE TABLE Notifications (
    notification_id INT PRIMARY KEY IDENTITY(1,1),
    user_id INT NOT NULL,
    message NVARCHAR(500) NOT NULL,
    sent_at DATETIME DEFAULT GETDATE(),
    is_read BIT DEFAULT 0,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

```

```
CREATE TABLE Reports (
    report_id INT PRIMARY KEY IDENTITY(1,1),
    evaluation_id INT NOT NULL,
    generated_at DATETIME DEFAULT GETDATE(),
    file_path NVARCHAR(255) NOT NULL,
    FOREIGN KEY (evaluation_id) REFERENCES
Evaluations(evaluation_id)
);
```

Основные сущности и связи:

- «User-service» (employees – сотрудники организации, departments – отделы, positions – должности, job_vacancies –открытые вакансии);
- «Evaluation-service» (evaluations – результаты оценок, evaluation_procedures – процедуры запуска оценки, participants – участники оценки);
- «Auth-service» (users – учетные записи, roles – роли доступа);
- «Report-service» (нет собственных таблиц, взаимодействует через API и Kafka).

Разработанная структура базы данных полностью соответствует требованиям предметной области и обеспечивает эффективное хранение и обработку информации, а так же позволяет легко масштабировать систему при необходимости.

3.2 Реализация веб-приложения

Разработка веб-приложения выполнена с применением архитектуры клиент-сервер, где фронтенд реализован в виде одностраничного приложения (SPA), а бэкенд – в виде распределённого набора микросервисов. Коммуникация между компонентами осуществляется через REST API и Apache Kafka. Такой подход обеспечивает модульность, гибкость, а также удобство поддержки и масштабирования системы.

3.2.1 Реализация интерфейса пользователя

Пользовательский интерфейс реализован с использованием библиотеки React, что позволяет эффективно управлять состоянием приложения и

динамически обновлять содержимое страниц без полной перезагрузки. Благодаря применённым технологиям интерфейс остается отзывчивым, интуитивным и кроссбраузерным, а адаптивная верстка обеспечивает комфортную работу с приложением как на десктопных, так и на мобильных устройствах. Основной интерфейс приложения представлен на рисунке 6.

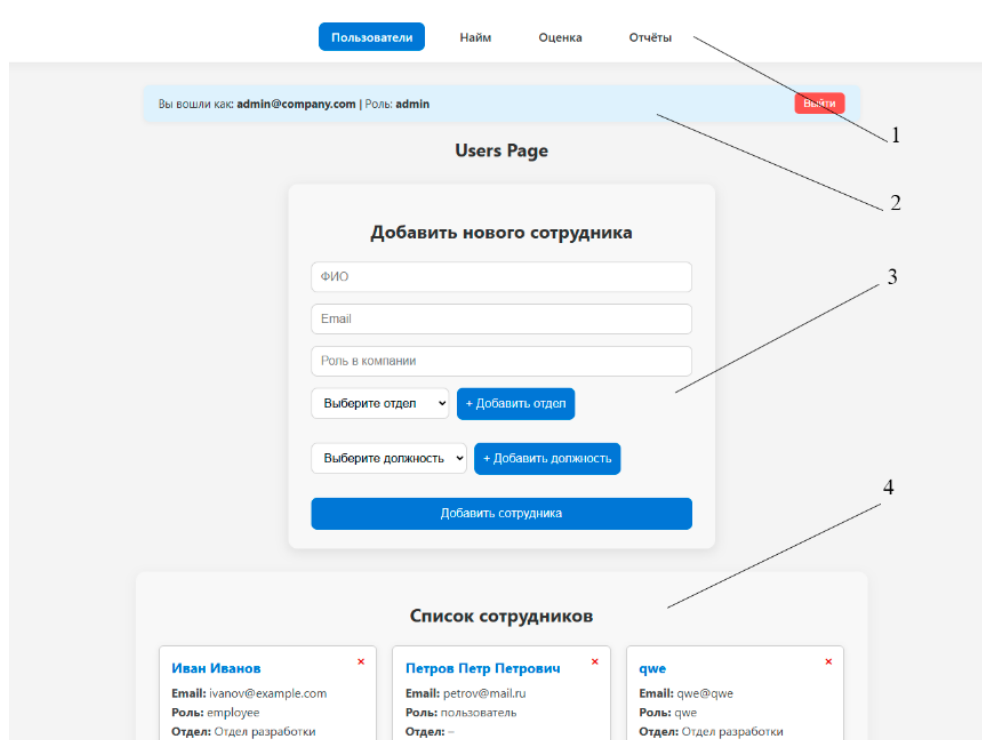


Рисунок 6 – Основной интерфейс

Рассмотрим основные элементы интерфейса главного экрана приложения, представленные на рисунке 6:

- 1 – верхняя панель, которая содержит основное меню навигации;
- 2 – блок с авторизованным пользователем;
- 3 – форма добавления сотрудника;
- 4 – список сотрудников.

Основные экраны приложения включают:

- форму входа и регистрации (листинг 2) (рисунок Б.1);
- основное окно учета сотрудников (смотри рисунок 6);
- формы оценки (рисунок Б.2);
- просмотр отчётов (листинг 3) (рисунок 7);

— форма управления вакансиями (рисунок Б.3).

Пользователи Найм Оценка **Отчёты**

Вы вошли как: admin@company.com | Роль: admin Выйти

Генерация отчётов

По сотруднику

Выберите сотрудника ▾

Сформировать отчет

По подразделению

Выберите подразделение ▾

Сформировать отчет

Рисунок 7 – Окно отчетов

Отчеты формируются в PDF-файле по выбранному сотруднику или подразделению (рисунок Б.4).

Листинг 2 – Компонент формы входа LoginForm

```
// components/LoginForm.js
function LoginForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = async () => {
    const response = await axios.post('/api/auth/login', { email,
password });

    localStorage.setItem('token', response.data.token);
  };

  return (
    <form onSubmit={handleLogin}>
      <input type="email" value={email} onChange={e =>
setEmail(e.target.value)} />
      <input type="password" value={password} onChange={e =>
setPassword(e.target.value)} />
      <button type="submit">Войти</button>
    </form>
  );
}
```

Листинг 3 – Компонент отображение отчёта ReportViewer

```
// components/ReportViewer.js
function ReportViewer({ employeeId }) {
  useEffect(() => {
    axios.get(`/api/reports/employee/${employeeId}`)
      .then(response => setReport(response.data));
  }, [employeeId]);

  return <div dangerouslySetInnerHTML={{ __html: report.content }}
  />;
}
```

Реализованный пользовательский интерфейс полностью соответствует современным стандартам веб-разработки и обеспечивает комфортную работу с системой для всех категорий пользователей. Продуманная архитектура компонентов и использование лучших практик React-разработки создают хорошую основу для дальнейшего расширения функциональности системы.

3.2.2 Реализация серверной части приложения

Серверная часть построена по принципу микросервисной архитектуры, каждый сервис отвечает за строго ограниченную функциональность. Реализация выполнена на Node.js с использованием Express.js.

Основные микросервисы:

- auth-service – авторизация, проверка токенов (листинг 4);
- user-service – управление пользователями, подразделениями, вакансиями;
- evaluation-service – запуск и обработка процедур оценки (листинг 5);
- report-service – генерация PDF- и Excel-отчетов;
- notification-service – отправка уведомлений.

Листинг 4 – Маршрут авторизации

```
router.post('/login', async (req, res) => {
  const user = await User.findOne({ where: { email: req.body.email } });
  const isValid = await bcrypt.compare(req.body.password, user.password);
});
```

```

    if (!isPasswordValid) return res.status(401).send('Ошибка
авторизации');

    const token = jwt.sign({ id: user.id, role: user.role },
process.env.JWT_SECRET);
    res.send({ token });
  });

```

Листинг 5 – Маршрут запуска оценки

```

router.post('/evaluations/start', async (req, res) => {
  const evaluation = await Evaluation.create({ ...req.body });
  await kafkaProducer.send({
    topic: 'evaluation.started',
    messages: [{ value: JSON.stringify(evaluation) }]
  });
  res.status(201).json(evaluation);
});

```

Серверная часть приложения полностью соответствует поставленным требованиям, обеспечивая стабильную работу системы и готовность к дальнейшему расширению функционала.

3.2.3 Реализация взаимодействия между компонентами

Взаимодействие между микросервисами реализовано двумя способами:

- REST API – для синхронных запросов;
- Apache Kafka – для событийного взаимодействия.

REST API используется для запросов, от которых требуется мгновенный отклик. Например, при входе пользователя в систему frontend отправляет запрос на /auth/login, а при отображении анкеты на /evaluations/:id. Серверная логика при этом реализована с использованием Express.js в каждом микросервисе. Одной из наиболее распространённых операций во взаимодействии между клиентской частью и сервером является получение списка оценок, относящихся к конкретному сотруднику. Этот запрос инициируется пользовательским интерфейсом (frontend) при переходе на страницу результатов сотрудника. Запрос отправляется по следующему адресу: /evaluations/employee/5. На стороне сервера реализован маршрут в Express.js, который выполняет выборку всех записей из таблицы evaluations, относящихся

к конкретному сотруднику. В случае отсутствия данных возвращается код 404 (листинг 6).

Листинг 6 – Получение всех оценок сотрудника

```
router.get('/employee/:id', async (req, res) => {
  const employeeId = req.params.id;
  try {
    const evaluations = await Evaluation.findAll({ where: {
employeeId } });
    if (!evaluations.length) {
      return res.status(404).json({ error: 'Оценки не найдены' });
    }
    res.json(evaluations);
  } catch (err) {
    console.error('Ошибка получения оценок:', err.message);
    res.status(500).json({ error: 'Ошибка сервера при получении
оценок' });
  }
});
```

На стороне клиентского приложения реализован вызов API через библиотеку axios. Запрос отправляется при загрузке компонента (листинг 7).

Листинг 7 – Клиентский запрос

```
useEffect(() => {
  axios.get(`/api/evaluations/employee/${employeeId}`)
    .then(response => setData(response.data))
    .catch(err => alert('Ошибка загрузки'));
}, [employeeId]);
```

Асинхронная модель применяется для операций, выполнение которых может быть отложено во времени или которые требуют одновременной реакции от нескольких компонентов. Основным механизмом является публикация сообщений в определённый Kafka-топик, после чего подписанные микросервисы получают и обрабатывают эти сообщения независимо друг от друга.

После запуска процедуры оценки evaluation-service создаёт запись в базе и формирует список участников. Затем он публикует сообщение в Kafka-топик evaluation.created (листинг 8). Каждый сервис получает событие независимо и выполняет свою логику.

Листинг 8 – Публикация события в Kafka

```
await kafkaProducer.send({
  topic: 'evaluation.created',
  messages: [
    {
      value: JSON.stringify({
        evaluationId: evaluation.id,
        participants: participantIds,
        title: evaluation.title
      })
    }
  ]
});
```

Подписавшись на топик `evaluation.created`, сервис уведомлений формирует и сохраняет индивидуальные уведомления для участников оценки. Уведомления отображаются в личном кабинете, либо направляются по электронной почте, в зависимости от конфигурации.

4 Тестирование

Тестирование является необходимым этапом разработки программного обеспечения и направлено на выявление дефектов, проверку корректности реализации требований, а также оценку устойчивости и совместимости системы. В рамках дипломного проекта были проведены функциональное и системное тестирование, а также выполнен анализ метрик программного кода.

4.1 Функциональное тестирование

Для проверки соответствия реализованных функций требованиям, описанным в разделе 2.1, применялась стратегия тестирования методом «черного ящика» [21, 22]. Проверка проводилась без знания внутренней логики системы, путём подачи входных данных и анализа ожидаемого результата.

4.1.1 Тестирование отправки уведомлений

Объект тестирования: функциональность отправки уведомления при запуске оценки (рисунок 8).

Ожидаемое поведение: при публикации события `evaluation.created` сервис уведомлений (`notification-service`) должен сформировать уведомления для всех участников оценки.

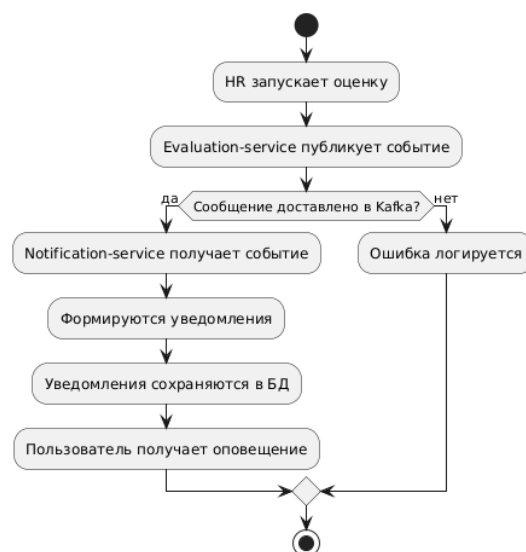


Рисунок 8 – Функциональная диаграмма (п-схема)

Результат теста: сценарий успешно отрабатывает при корректной настройке Kafka и наличии подписчиков, письмо сотруднику отправлено (рисунок 9). В случае сбоя подключения система корректно логирует ошибку без остановки остальных сервисов.

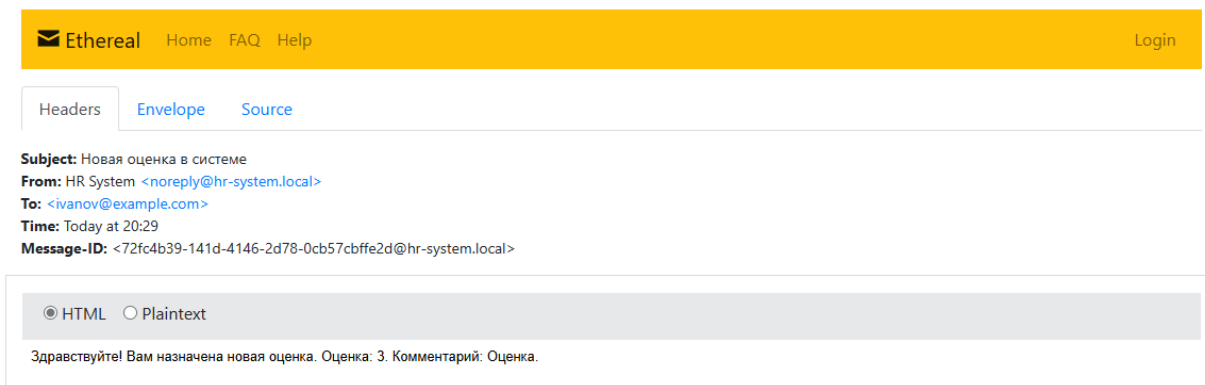


Рисунок 9 – Отправленное письмо

Проведенные тесты подтвердили корректность работы механизма отправки уведомлений через Kafka. Система успешно обрабатывает события создания оценки, формирует соответствующие уведомления и доставляет их участникам процедуры. В случае временной недоступности сервиса уведомлений система сохраняет устойчивость за счет асинхронной модели взаимодействия и механизма повторных попыток доставки. Логирование всех этапов процесса позволяет оперативно выявлять и устранять возможные сбои.

4.1.2 Тестирование авторизации в системе

Целью теста являлась проверка возможности входа в систему зарегистрированного пользователя. Входные данные включали электронную почту (ivan1@test.com) и пароль (pas123). Функция входа работает в соответствии с требованиями – тест пройден (рисунок 10).

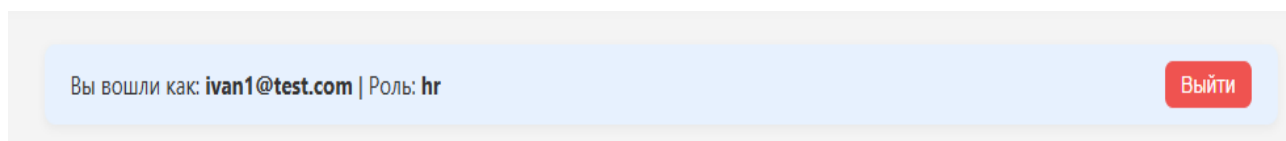


Рисунок 10 – Тестирование авторизации в системе

В результате успешного выполнения запроса система корректно выдала JWT-токен и выполнила автоматическое перенаправление на домашнюю страницу.

4.1.3 Тестирование получения анкеты для оценки

Проверялась корректность работы маршрута, предназначенного для получения списка оценок, назначенных конкретному сотруднику. Тестовый HTTP-запрос GET /evaluations/employee/5 возвращал ожидаемый JSON-ответ с перечнем оценок и статусом HTTP 200. Тест успешно подтверждает работоспособность данного функционала (рисунок 11).

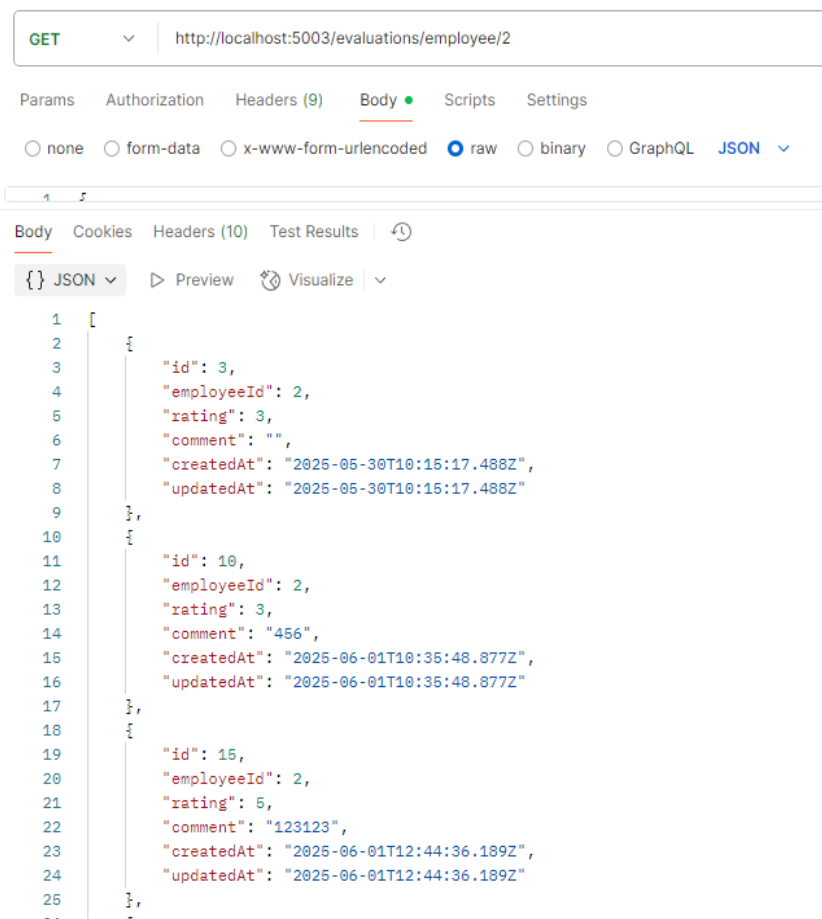


Рисунок 11 – Тестирование получения анкеты для оценки

Тестирование подтвердило корректность работы предоставления оценочных анкет. Система успешно обрабатывает запросы на получение анкет, учитывая ролевую модель доступа и связи между участниками оценки.

4.1.4 Тестирование создания оценки сотрудника

В рамках теста была проведена оценка сотрудника через форму на странице оценки. Результатом является созданная оценка и отправленное письмо участнику (рисунок 12).

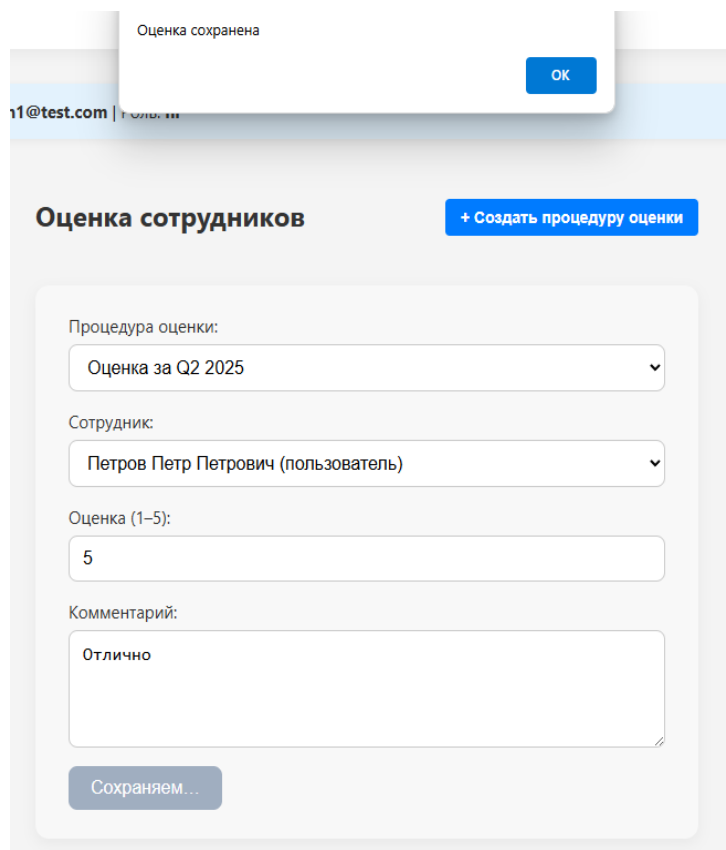


Рисунок 12 – Тестирование создания оценки сотрудника

Проведенные тесты подтвердили корректность работы модуля формирования отчетов.

4.1.5 Тестирование получения анкеты для оценки

Проверялась реакция report-service на событие в топике report.topic, содержащее идентификатор процедуры оценки. После получения события сервис сгенерировал корректный PDF-отчёт и сохранил путь к файлу в базе данных. Файл стал доступен пользователю для просмотра и загрузки, что подтверждает успешную реализацию логики асинхронной генерации отчётов (рисунок Б.4).

4.2 Анализ метрик кода

Для оценки сложности и качества программного кода использовалась метрика Холстеда (Halstead Complexity Measures), применённая к функции отправки уведомлений в Kafka (листинг 9), результаты отображены в таблице 6.

Листинг 9 – Анализ метрик кода

```
async function sendNotification({ to, subject, body }) {  
  await producer.connect();  
  await producer.send({  
    topic: 'notification-topic',  
    messages: [{ value: JSON.stringify({ to, subject, body }) }]  
  });  
  await producer.disconnect();  
}
```

Таблица 6 – Результаты метрик

Показатель	Значение
Кол-во различных операторов (n_1)	8
Кол-во различных операндов (n_2)	5
Общее число операторов (N_1)	11
Общее число операндов (N_2)	10
Словарь программы (n)	13 ($n_1 + n_2$)
Длина программы (N)	21 ($N_1 + N_2$)
Объём (V) = $N \times \log_2(n)$	$\approx 21 \times 3.7 = 77.7$
Трудоёмкость (E) = $V \times n_1/n_2$	$\approx 77.7 \times 8 / 5 \approx 124.3$

Можно сделать вывод, функция имеет среднюю когнитивную сложность, компактна и читаема. Уровень трудоёмкости допустим для промышленной эксплуатации.

4.3 Системное тестирование

Системное тестирование охватывало следующие аспекты: визуальное отображение интерфейса, взаимодействие компонентов, производительность и кроссбраузерную совместимость [23]. Для его проведения использовались как ручные проверки для различных браузеров (таблица 7), так и инструменты разработчика для проверки адаптивности (таблица 8).

Таблица 7 – Тестирование в различных браузерах

Браузер	Версия	Результат
Google Chrome	125.0	успешно
Mozilla Firefox	124.0	успешно
Microsoft Edge	123.0	успешно
Safari (macOS)	16.3	успешно
Opera	108.0	успешно

Таблица 8 – Тестирование адаптивности на экранах

Устройство	Разрешение экрана	Статус отображения
Смартфон (360×640)	360×640	корректно
Планшет (768×1024)	768×1024	корректно
Ноутбук (1366×768)	1366×768	корректно
Монитор FullHD	1920×1080	корректно

Исходя из результатов сделаем вывод, интерфейс приложения адаптирован к различным устройствам, все ключевые функции корректно работают во всех актуальных браузерах.

Проведённое функциональное и системное тестирование подтвердило соответствие приложения заявленным требованиям. Применение асинхронного взаимодействия с использованием Kafka показало надёжность событийной архитектуры. Оценка метрик качества кода продемонстрировала приемлемую сложность и эффективность реализации. Все выявленные незначительные ошибки были устранены на этапе отладки.

Заключение

В ходе выполнения дипломной работы была успешно разработана и реализована система автоматизированного учёта и оценки персонала для HR-специалистов. На начальном этапе проведён детальный анализ существующих HRM-решений, который выявил их ключевые недостатки: высокую стоимость, сложность настройки. Это позволило обосновать необходимость создания специализированного решения, сочетающего простоту использования.

Проектирование системы осуществлялось с применением современных подходов к разработке программного обеспечения. Была выбрана микросервисная архитектура, обеспечивающая высокую масштабируемость и отказоустойчивость решения. В качестве технологического стека использовались Node.js для серверной части, React для клиентского интерфейса и Microsoft SQL Server в качестве системы управления базами данных.

Реализация веб-приложения подтвердила эффективность выбранных технологических решений. Пользовательский интерфейс, разработанный на React, обеспечил интуитивно понятное взаимодействие с системой, в то время как бэкенд на Node.js в сочетании с Apache Kafka продемонстрировал стабильную работу распределённых компонентов. Особенностью реализации стало использование событийно-ориентированной архитектуры для обработки уведомлений и генерации отчётов.

Тестирование системы включало проверку функциональных требований, анализ производительности и оценку безопасности. Результаты тестирования подтвердили полное соответствие системы заявленным требованиям: все компоненты работают корректно, интерфейс адаптируется под различные устройства, а механизмы авторизации обеспечивают необходимый уровень защиты данных.

Практическая значимость разработанного решения заключается в существенном упрощении процессов оценки персонала, сокращении временных затрат на обработку данных и повышении прозрачности HR-деятельности. Система ориентирована на организации малого бизнеса, а её

модульная архитектура позволяет легко адаптировать функционал под конкретные потребности предприятия.

Для дальнейшего развития проекта рекомендуется реализовать интеграцию с популярными корпоративными мессенджерами (Slack, Teams) для оперативного оповещения сотрудников, а также рассмотреть возможность внедрения методов машинного обучения для углублённого анализа результатов оценки и прогнозирования кадровых решений.

Список используемых источников

1. Иванов, А.В. Цифровизация бизнеса: тенденции и перспективы / А.В. Иванов. – М.: Альпина Паблишер, 2020. – 256 с.
2. Кузнецов, С.П. Управление персоналом в современных организациях / С.П. Кузнецов. – СПб.: Питер, 2019. – 320 с.
3. Петрова, Л.А. Методы оценки персонала: практическое руководство / Л.А. Петрова. – М.: Эксмо, 2021. – 180 с.
4. Рамазанов, И.А. Финансовое планирование в проектах : практическое руководство / И.А. Рамазанов. – Москва : Издательство «инансы и кредит», 2021. – 215 с.
5. Вейл, Д.Ф. Разработка требований к программному обеспечению: практическое руководство для бизнес-аналитиков / Д.Ф. Вейл, К.Л. Джонсон ; пер. с англ. А.П. Смирнова. – 3-е изд., перераб. и доп. – Москва: ДМК Пресс, 2020. – 418 с.
6. Ньюмен, С. Создание микросервисов / С. Ньюмен. – СПб.: Питер, 2020. – 400 с.
7. Шапиро, М.В. Распределенные системы: Apache Kafka в действии / М.В. Шапиро. – Москва: ДМК Пресс, 2021. – 342 с.
8. Тихонов, А.Ю. Базы данных: проектирование и реализация / А.Ю. Тихонов. – М.: Лань, 2021. – 352 с.
9. Лукин, С.В. Microsoft SQL Server 2019: администрирование и настройка / С.В. Лукин. – Санкт-Петербург: БХВ-Петербург, 2021. – 768 с.
10. Леоненков, А.В. Диаграммы вариантов использования UML 2.0: практическое пособие / А.В. Леоненков. – Санкт-Петербург : БХВ-Петербург, 2019. – 320 с.
11. Розенберг, Д. Use Case Driven Object Modeling with UML / Д. Розенберг, К. Скотт; пер. с англ. П.Р. Волкова. – Москва: Вильямс, 2021. – 384 с.

12. Петров, В.С. Совершенствование диаграмм классов при моделировании предметных областей / В.С. Петров // Программная инженерия. – 2023. – № 1. – С. 34 – 42.
13. Зиндер, Е.З. Диаграммы последовательности UML: практикум / Е.З. Зиндер. – Москва: ДМК Пресс, 2022. – 256 с.
14. Object Management Group. UML Sequence Diagrams [Электронный ресурс] // OMG Documentation. – 2022. – URL: <https://www.omg.org/spec/UML/2.5.1/> (дата обращения: 25.04.2025).
15. Смирнов, А.В. Современные подходы к проектированию ER-моделей для распределенных систем / А.В. Смирнов // Системы управления базами данных. – 2023. – № 1. – С. 23 – 31.
16. Дейт, К.Дж. Введение в системы баз данных / К.Дж. Дейт ; пер. с англ. С.М. Дигио. – 8-е изд. – Москва : Вильямс, 2021. – 1072 с.
17. Херрон, Д. Node.js в действии / Д. Херрон, М. Кантелон ; пер. с англ. А.Н. Киселёва. – 2-е изд. – Москва : Питер, 2021. – 480 с.
18. Node.js для начинающих [Электронный ресурс] / А.В. Петров. – 2023. – URL: <https://habr.com/ru/articles/nodejs-guide/> (дата обращения: 29.04.2025).
19. Нилсен, Я. Юзабилити: как сделать сайт удобным / Я. Нилсен ; пер. с англ. В.В. Головача. – 2-е изд. – Москва : Питер, 2021. – 448 с.
20. Тидвелл, Д. Разработка пользовательских интерфейсов / Д. Тидвелл ; пер. с англ. А.В. Славникова. – Москва : Вильямс, 2022. – 528 с.
21. Куликов, С.В. Функциональное тестирование программного обеспечения: учебное пособие / С.В. Куликов. – Москва : ДМК Пресс, 2022. – 318 с.
22. Иванова, Е.П. Методы функционального тестирования веб-приложений / Е.П. Иванова // Программная инженерия. – 2023. – № 1. – С. 23 – 31.

23. Петров, А.В. Современные подходы к системному тестированию распределенных систем / А.В. Петров // Программная инженерия. – 2023. – № 2. – С. 34 – 42.

Приложение А
(обязательное)
Код приложения

Листинг А.1 – Файл user-service/server.js

```
const expressUser = require('express');
const corsUser = require('cors');
const dotenvUser = require('dotenv');
const { Sequelize: SequelizeUser, DataTypes: DataTypesUser } =
require('sequelize');
const sendNotificationMessage = require('./kafka/producer');

dotenvUser.config();

const userApp = expressUser();
userApp.use(corsUser());
userApp.use(expressUser.json());

const sequelizeUser = new SequelizeUser(process.env.DB_URL);

const Department = sequelizeUser.define('Department', {
  name: { type: DataTypesUser.STRING, allowNull: false }
});

const Position = sequelizeUser.define('Position', {
  title: { type: DataTypesUser.STRING, allowNull: false },
});

const Employee = sequelizeUser.define('Employee', {
  name: { type: DataTypesUser.STRING, allowNull: false },
  email: { type: DataTypesUser.STRING, unique: true, allowNull:
false },
  role: { type: DataTypesUser.STRING, allowNull: false },
});

Department.hasMany(Employee);
Employee.belongsTo(Department);

Position.hasMany(Employee);
Employee.belongsTo(Position);

const JobVacancy = sequelizeUser.define('JobVacancy', {
  title: { type: DataTypesUser.STRING, allowNull: false },
  description: { type: DataTypesUser.TEXT, allowNull: true },
  status: { type: DataTypesUser.STRING, defaultValue: 'open' }, //
open/closed
});

userApp.get('/vacancies', async (req, res) => {
  const vacancies = await JobVacancy.findAll();
  res.json(vacancies);
});

userApp.post('/vacancies', async (req, res) => {
  try {
    const { title, description, status } = req.body;
```

```

        const vacancy = await JobVacancy.create({ title, description,
status });
        res.status(201).json(vacancy);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

userApp.get('/departments', async (req, res) => {
    try {
        const departments = await Department.findAll();
        res.json(departments);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

userApp.post('/departments', async (req, res) => {
    try {
        const department = await Department.create({ name:
req.body.name });
        res.status(201).json(department);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

userApp.get('/positions', async (req, res) => {
    try {
        const positions = await Position.findAll();
        res.json(positions);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

userApp.post('/positions', async (req, res) => {
    try {
        const position = await Position.create({ title: req.body.title
});
        res.status(201).json(position);
    } catch (err) {
        res.status(400).json({ error: err.message });
    }
});

userApp.get('/employees', async (req, res) => {
    try {
        const employees = await Employee.findAll({ include:
[Department, Position] });
        res.json(employees);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

```

```

    }
  });

  userApp.get('/employees/:id', async (req, res) => {
    const id = req.params.id;
    try {
      const employee = await Employee.findByPk(id);
      if (!employee) return res.status(404).json({ error: 'Сотрудник не найден' });
      res.json(employee);
    } catch (err) {
      res.status(500).json({ error: err.message });
    }
  });

  userApp.post('/employees', async (req, res) => {
    const { name, email, role, departmentId, positionId } = req.body;
    try {
      const employee = await Employee.create({ name, email, role, DepartmentId: departmentId, PositionId: positionId });
      await sendNotificationMessage({
        type: 'assignment',
        to: email,
        data: {
          name,
          link: 'http://client.hr-system.local/login', // ссылка на SPA, если есть
          subject: 'Добро пожаловать в HR-систему',
        },
      });
      res.status(201).json(employee);
    } catch (err) {
      res.status(400).json({ error: err.message });
    }
  });

  const USER_PORT = process.env.USER_PORT || 5002;
  async function startUserService() {
    try {
      await sequelizeUser.authenticate();
      await sequelizeUser.sync();
      userApp.listen(USER_PORT, () => console.log(`User-service on port ${USER_PORT}`));
    } catch (e) {
      console.error('User-service failed:', e);
    }
  }

  userApp.delete('/employees/:id', async (req, res) => {
    const id = req.params.id;
    try {

```

```

    const deletedCount = await Employee.destroy({ where: { id }
});
    if (deletedCount === 0) {
        return res.status(404).json({ error: 'Сотрудник не найден'
});
    }
    res.json({ message: 'Сотрудник удалён' });
} catch (err) {
    res.status(500).json({ error: err.message });
}
});

startUserService();

```

Листинг А.2 – Файл user-service/.ENV

```

DB_URL=mssql://dp:pass1@localhost:1433/DiplomDB
USER_PORT=5002
JWT_SECRET=supersecretkey360
JWT_EXPIRES_IN=1d
KAFKA_BROKER=localhost:9092

```

Листинг А.3 – Файл user-service\kafka\producer.js

```

const { Kafka } = require('kafkajs');
require('dotenv').config();

const kafka = new Kafka({
    clientId: 'user-service',
    brokers: [process.env.KAFKA_BROKER],
});
const producer = kafka.producer();

async function sendNotificationMessage(message) {
    await producer.connect();
    await producer.send({
        topic: 'notification.queue',
        messages: [{ value: JSON.stringify(message) }],
    });
    await producer.disconnect();
}

module.exports = sendNotificationMessage;

```

Листинг А.4 – Файл user-service\middleware\middlewareverifyToken.js

```

const jwt = require('jsonwebtoken');

const verifyToken = (roles = []) => {
    return (req, res, next) => {
        try {

```

```

    const header = req.headers.authorization;
    if (!header || !header.startsWith('Bearer ')) {
        return res.status(401).json({ error: 'Токен отсутствует'
    });
    }

    const token = header.split(' ')[1];
    const user = jwt.verify(token, process.env.JWT_SECRET);

    if (roles.length && !roles.includes(user.role)) {
        return res.status(403).json({ error: 'Недостаточно прав'
    });
    }

    req.user = user;
    next();
} catch (err) {
    return res.status(401).json({ error: 'Неверный токен' });
}
};
};

module.exports = verifyToken;

```

Листинг A.5 – Файл report-service\server.js

```

const express = require('express');
const dotenv = require('dotenv');
const { consumer } = require('./kafka');
const { generateReport } = require('./reportGenerator');
const cors = require('cors');

dotenv.config();
const app = express();

app.use(cors({
    origin: 'http://localhost:3000',
    methods: ['GET', 'POST'],
}));

app.use('/reports', require('./routes/reportRoutes'));

async function startReportService() {
    await consumer.connect();
    await consumer.subscribe({ topic: 'report-topic', fromBeginning:
false });

    await consumer.run({
        eachMessage: async ({ topic, partition, message }) => {
            try {
                const { type, data } =
JSON.parse(message.value.toString());

```

```

        if (type === 'generate-report') {
            console.log(`Генерация отчета для:
${data.employeeEmail}`);
            generateReport(data,
`report_${data.employeeId}_${Date.now()}.pdf`);
        } else {
            console.log('Необработанный тип сообщения:', type);
        }
    } catch (err) {
        console.error('Ошибка генерации отчета:', err.message);
    }
},
});

const PORT = process.env.REPORT_PORT || 5005;
app.listen(PORT, () => {
    console.log(`Report-service работает на порту ${PORT}`);
});
}

startReportService().catch(console.error);

```

Листинг А.6 – Файл report-service\reportGenerator.js

```

const PDFDocument = require('pdfkit');
const fs = require('fs');
const path = require('path');

function generateReport(data, fileName = 'report.pdf') {
    return new Promise((resolve, reject) => {
        const doc = new PDFDocument();
        const filePath = path.join(__dirname, 'reports', fileName);
        fs.mkdirSync(path.dirname(filePath), { recursive: true });

        const stream = fs.createWriteStream(filePath);
        doc.pipe(stream);

        const fontPath = path.join(__dirname, 'fonts',
'DejaVuSans.ttf');
        doc.font(fontPath);

        doc.fontSize(20).text('Отчёт об оценке', { align: 'center' });
        doc.moveDown();

        doc.fontSize(14).text(`Сотрудник: ${data.name}`);
        doc.text(`Email: ${data.employeeEmail}`);
        doc.text(`Оценка: ${data.rating}`);
        doc.text(`Комментарий: ${data.comment}`);
        doc.text(`Дата: ${new Date().toLocaleString()}`);

        doc.end();
    });
}

```

```

    stream.on('finish', () => {
      console.log('Отчёт сохранён в:', filePath);
      resolve(filePath);
    });

    stream.on('error', (err) => {
      reject(err);
    });
  });
}

module.exports = { generateReport };

```

Листинг А.7 – Файл report-service\kafka.js

```

const { Kafka } = require('kafkajs');
require('dotenv').config();
const kafka = new Kafka({
  clientId: 'report-service',
  brokers: [process.env.KAFKA_BROKER],
});

const consumer = kafka.consumer({ groupId: 'report-group' });
module.exports = { kafka, consumer };

```

Листинг А.8 – Файл report-service\.env

```

KAFKA_BROKER=localhost:9092
REPORT_PORT=5005

```

Листинг А.9 – Файл report-service\services\reportService.js

```

const axios = require('axios');

async function getAllEmployeesWithResults() {
  try {
    const { data: employees } = await
    axios.get('http://localhost:5002/employees'); // список всех
    сотрудников

    const results = [];

    for (const emp of employees) {
      const { data: evaluations } = await
      axios.get(`http://localhost:5003/evaluations/employee/${emp.id}`);

      if (evaluations.length > 0) {

```

```

        const lastEvaluation = evaluations.sort((a, b) => new
Date(b.createdAt) - new Date(a.createdAt))[0];

        results.push({
            id: emp.id,
            fullName: emp.name,
            departmentName: emp.department?.name || 'Не указано',
            positionName: emp.position?.name || 'Не указано',
            evaluationScore: lastEvaluation.rating,
            comment: lastEvaluation.comment || ''
        });
    }
}

return results;
} catch (error) {
    console.error('Ошибка при получении данных сотрудников и
оценок:', error.message);
    return [];
}
}

module.exports = {
    getAllEmployeesWithResults
};

```

Листинг А.10 – Файл report-service\routes\reportRoutes.js

```

const express = require('express');
const router = express.Router();
const { sendKafkaReportRequest } = require('../kafka/producer');
const {
    generateReportController,
    generateDepartmentReportController
} = require('../controllers/reportController');

router.get('/employee/:id', generateReportController);
router.get('/department/:id', generateDepartmentReportController);
// <-- добавлено

router.post('/', async (req, res) => {
    const { employeeId, employeeEmail, rating, comment } = req.body;

    if (!employeeId || !employeeEmail) {
        return res.status(400).json({ error: 'employeeId и
employeeEmail обязательны' });
    }

    try {
        await sendKafkaReportRequest({ employeeId, employeeEmail,
rating, comment });
    }
}

```

```

    res.status(200).json({ message: 'Запрос на генерацию отчета
отправлен в Kafka' });
  } catch (err) {
    console.error('Ошибка отправки в Kafka:', err);
    res.status(500).json({ error: 'Ошибка генерации отчета через
Kafka' });
  }
});
module.exports = router;

```

Листинг A.11 – Файл report-service\kafka\producer.js

```

const { Kafka } = require('kafkajs');

const kafka = new Kafka({
  clientId: 'report-service',
  brokers: ['localhost:9092'],
});

const producer = kafka.producer();

async function sendKafkaReportRequest(payload) {
  await producer.connect();
  await producer.send({
    topic: 'report-topic',
    messages: [
      {
        value: JSON.stringify({
          type: 'generate-report',
          data: payload
        }),
      },
    ],
  });
  await producer.disconnect();
}

module.exports = { sendKafkaReportRequest };

```

Листинг A.12 – Файл report-service\controllers\reportController.js

```

const axios = require('axios');
const path = require('path');
const fs = require('fs');
const ExcelJS = require('exceljs');
const { generateReport } = require('../reportGenerator');
const { getAllEmployeesWithResults } =
require('../services/reportService');

async function generateReportController(req, res) {
  const employeeId = req.params.id;

```

```

    try {
      const employeeUrl =
`http://localhost:5002/employees/${employeeId}`;
      console.log(`Запрос данных сотрудника по URL:
${employeeUrl}`);
      const { data: employee } = await axios.get(employeeUrl);

      const evaluationsUrl =
`http://localhost:5003/evaluations/employee/${employeeId}`;
      console.log(`Запрос данных оценок по URL: ${evaluationsUrl}`);
      const { data: evaluations } = await axios.get(evaluationsUrl);

      if (!evaluations.length) {
        return res.status(404).json({ error: 'Оценки не найдены' });
      }

      const lastEvaluation = evaluations.sort((a, b) => new
Date(b.createdAt) - new Date(a.createdAt))[0];

      const reportData = {
        name: employee.name,
        employeeEmail: employee.email,
        rating: lastEvaluation.rating,
        comment: lastEvaluation.comment || '-',
      };

      const fileName = `report_${employeeId}.pdf`;
      const filePath = await generateReport(reportData, fileName);

      res.download(filePath, fileName, (err) => {
        if (err) {
          console.error('Ошибка отправки файла:', err);
          res.status(500).send('Ошибка отправки файла');
        }
      });
    } catch (error) {
      console.error('Ошибка генерации отчета:', error.message);
      if (error.response && error.response.status === 404) {
        return res.status(404).json({ error: 'Сотрудник или оценки
не найдены' });
      }
      res.status(500).json({ error: 'Ошибка сервера' });
    }
  }

  async function generateDepartmentReportController(req, res) {
    try {
      const employees = await getAllEmployeesWithResults();

      if (!employees || employees.length === 0) {
        return res.status(404).json({ message: 'Нет данных для
отчета' });
      }
    }
  }

```

```

    }

    const workbook = new ExcelJS.Workbook();
    const worksheet = workbook.addWorksheet('Отчет по
сотрудникам');

    worksheet.columns = [
        { header: 'ID', key: 'id', width: 10 },
        { header: 'ФИО', key: 'name', width: 30 },
        { header: 'Подразделение', key: 'department', width: 25 },
        { header: 'Должность', key: 'position', width: 25 },
        { header: 'Оценка', key: 'evaluation', width: 15 },
        { header: 'Комментарий', key: 'comment', width: 40 }
    ];

    employees.forEach(emp => {
        worksheet.addRow({
            id: emp.id,
            name: emp.fullName,
            department: emp.departmentName,
            position: emp.positionName,
            evaluation: emp.evaluationScore,
            comment: emp.comment || ''
        });
    });

    worksheet.getRow(1).font = { bold: true };

    res.setHeader(
        'Content-Type',
        'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet'
    );
    res.setHeader(
        'Content-Disposition',
        'attachment; filename=employee_report.xlsx'
    );

    await workbook.xlsx.write(res);
    res.end();
} catch (error) {
    console.error('Ошибка при генерации отчета:', error);
    res.status(500).json({ message: 'Ошибка при генерации отчета'
});
}
};

module.exports = {
    generateReportController,
    generateDepartmentReportController
};

```

Листинг A.13 – Файл notification-service\server.js

```
const express = require('express');
const dotenv = require('dotenv');
const { consumer } = require('./kafka');
const { sendMail, initMailer } = require('./mailer');

dotenv.config();
const app = express();

async function startNotificationService() {
  await initMailer(); // <-- ВАЖНО: инициализация Ethereum

  await consumer.connect();
  await consumer.subscribe({ topic: 'notification-topic',
    fromBeginning: false });

  await consumer.run({
    eachMessage: async ({ topic, partition, message }) => {
      const { type, to, subject, body } =
        JSON.parse(message.value.toString());
      console.log(`Получено сообщение: ${type}, получатель:
        ${to}`);

      try {
        await sendMail(to, subject, body);
        console.log('Письмо отправлено:', to);
      } catch (err) {
        console.error('Ошибка отправки письма:', err.message);
      }
    },
  });

  const PORT = process.env.NOTIFICATION_PORT || 5004;
  app.listen(PORT, () => {
    console.log(`Notification-service работает на порту ${PORT}`);
  });
}

startNotificationService().catch(console.error);
```

Листинг A.14 – Файл notification-service\notificationTemplates.js

```
const { Kafka } = require('kafkajs');
const { renderNotification } = require('./notificationTemplates');
const { logNotification } = require('./utils/logger');

const kafka = new Kafka({
  clientId: 'notification-service',
  brokers: ['localhost:9092'],
});
```

```

const consumer = kafka.consumer({ groupId: 'notification-group'
});

async function startConsumer() {
  await consumer.connect();
  await consumer.subscribe({ topic: 'notification-topic',
fromBeginning: true });

  await consumer.run({
    eachMessage: async ({ message }) => {
      const rawData = message.value.toString();
      const event = JSON.parse(rawData);

      const rendered = renderNotification(event.type, event.data);

      if (rendered) {
        const fullNotification = {
          to: event.to,
          subject: rendered.subject,
          body: rendered.body,
          type: event.type
        };

        console.log('Уведомление:', fullNotification);
        await logNotification(fullNotification);
      } else {
        console.warn('Неизвестный тип уведомления:', event.type);
      }
    }
  });
}

module.exports = { startConsumer };

```

Листинг A.15 – Файл notification-service\mailer.js

```

const nodemailer = require('nodemailer');

let transporter;
let testAccount;

async function initMailer() {
  testAccount = await nodemailer.createTestAccount();

  transporter = nodemailer.createTransport({
    host: testAccount.smtp.host,
    port: testAccount.smtp.port,
    secure: testAccount.smtp.secure,
    auth: {
      user: testAccount.user,
      pass: testAccount.pass
    }
  });
}

```

```

    console.log('Ethereal mailer готов!');
    console.log('Вход на Ethereum:', testAccount.user);
    console.log('Пароль:', testAccount.pass);
  }

  async function sendMail(to, subject, text) {
    if (!transporter) {
      throw new Error('Mailer не инициализирован. Вызови
initMailer() перед отправкой.');
```

Листинг A.16 – Файл notification-service\kafka.js

```

const { Kafka } = require('kafkajs');
require('dotenv').config();

const kafka = new Kafka({
  clientId: 'notification-service',
  brokers: [process.env.KAFKA_BROKER],
});

const consumer = kafka.consumer({ groupId: 'notification-group'
});

module.exports = { kafka, consumer };
```

Листинг A.17 – Файл notification-service\.env

```

KAFKA_BROKER=localhost:9092
EMAIL_FROM=hr-system@example.com
SMTP_HOST=smtp.example.com
SMTP_PORT=587
SMTP_USER=your_smtp_user
SMTP_PASS=your_smtp_password
NOTIFICATION_PORT=5004
```

Листинг A.18 – Файл notification-service\utils\logger.js

```
const fs = require('fs-extra');
const path = require('path');

const logFile = path.join(__dirname, '..', 'logs',
'notifications.log');

async function logNotification(notification) {
  const line = `[${new Date().toISOString()}]
${JSON.stringify(notification)}\n`;
  await fs.ensureFile(logFile);
  await fs.appendFile(logFile, line);
}

module.exports = { logNotification };
```

Листинг A.19 – Файл evaluation-service\server.js

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors({
  origin: 'http://localhost:3000',
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  credentials: true
}));

app.use(express.json());

app.use('/evaluations', require('./routes/evaluationRoutes'));

const sequelize = require('./database');
sequelize.sync().then(() => {
  console.log("DB connected");
  app.listen(5003, () => {
    console.log("Evaluation service running on port 5003");
  });
}).catch(err => {
  console.error("DB connection error:", err);
});
```

Листинг A.20 – Файл evaluation-service\kafkaProducer.js

```
const { Kafka } = require('kafkajs');

const kafka = new Kafka({
  clientId: 'evaluation-service',
  brokers: ['localhost:9092'],
```

```

});

const producer = kafka.producer();

async function sendNotification({ to, subject, body }) {
  await producer.connect();
  await producer.send({
    topic: 'notification-topic',
    messages: [
      {
        value: JSON.stringify({
          type: 'evaluation-submitted',
          to,
          subject,
          body,
        }),
      },
    ],
  });
  await producer.disconnect();
}

module.exports = { sendNotification };

```

Листинг A.21 – Файл evaluation-service\database.js

```

require('dotenv').config();
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize(process.env.DB_URL, {
  dialect: 'mssql',
  dialectOptions: {
    options: {
      encrypt: false, // отключаем SSL (можно
      trustServerCertificate: true // для локальных подключений
    }
  },
  logging: false // включи true, если хочешь
  // видеть SQL в логах
});

module.exports = sequelize;

```

Листинг A.22 – Файл evaluation-service.env

```

PORT=5003
DB_URL=mssql://dp:pass1@localhost:1433/DiplomDB
JWT_SECRET=supersecretkey360
JWT_EXPIRES_IN=1d

```

Листинг A.23 – Файл evaluation-service\services\evaluationService.js

```
const EvaluationProcedure =
require('../models/EvaluationProcedure');
const Participant = require('../models/Participant');

exports.createProcedure = async (data) => {
  // Валидация
  if (!data.title || !data.startDate || !data.endDate ||
!data.createdBy) {
    throw new Error('Недостаточно данных для создания процедуры');
  }

  return await EvaluationProcedure.create(data);
};

exports.addParticipants = async (evaluationId, participants) => {
  if (!evaluationId || !Array.isArray(participants)) {
    throw new Error('Неверные параметры участников');
  }
  const inserts = participants.map(p => ({
    evaluationId,
    userId: p.userId,
    role: p.role
  }));

  return await Participant.bulkCreate(inserts);
};
```

Листинг A.24 – Файл evaluation-service\routes\evaluationRoutes.js

```
const express = require('express');
const axios = require('axios');
const router = express.Router();
const Evaluation = require('../models/Evaluation');
const { sendNotification } = require('../kafkaProducer');

router.post('/', async (req, res) => {
  const { employeeId, rating, comment } = req.body;

  if (!employeeId || !rating) {
    return res.status(400).json({ error: 'employeeId и rating
обязательны' });
  }

  try {
    // Получить email сотрудника
    const userResponse = await
axios.get(`http://localhost:5002/employees/${employeeId}`);
```

```

    const employeeEmail = userResponse.data.email;

    if (!employeeEmail) {
        return res.status(400).json({ error: 'Email сотрудника не найден' });
    }

    const newEvaluation = await Evaluation.create({
        employeeId,
        rating,
        comment,
        employeeEmail,
    });

    await sendNotification({
        to: employeeEmail,
        subject: 'Новая оценка в системе',
        body: `Здравствуйте! Вам назначена новая оценка. Оценка: ${rating}. Комментарий: ${comment || 'отсутствует'}.`,
    });

    res.status(201).json({ message: 'Оценка сохранена',
        evaluation: newEvaluation });
    } catch (err) {
        console.error('Ошибка при сохранении оценки:', err.message);
        if (err.response && err.response.status === 404) {
            res.status(404).json({ error: 'Сотрудник не найден в user-service' });
        } else {
            res.status(500).json({ error: 'Ошибка сервера при сохранении оценки' });
        }
    }
});

router.get('/employee/:id', async (req, res) => {
    const employeeId = req.params.id;

    try {
        const evaluations = await Evaluation.findAll({ where: {
            employeeId } });
        if (!evaluations.length) {
            return res.status(404).json({ error: 'Оценки не найдены' });
        }
        res.json(evaluations);
    } catch (err) {
        console.error('Ошибка получения оценок:', err.message);
        res.status(500).json({ error: 'Ошибка сервера при получении оценок' });
    }
});

module.exports = router;

```

Листинг A.25 – Файл evaluation-service\models\Evaluation.js

```
const { DataTypes } = require('sequelize');
const sequelize = require('../database');

const Evaluation = sequelize.define('Evaluation', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true
  },
  employeeId: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  rating: {
    type: DataTypes.INTEGER,
    allowNull: false,
    validate: {
      min: 1,
      max: 5
    }
  },
  comment: {
    type: DataTypes.TEXT,
    allowNull: true
  }
}, {
  tableName: 'evaluations'
});

module.exports = Evaluation;
```

Листинг A.26 – Файл evaluation-service\models\Evaluation.js

```
const { DataTypes } = require('sequelize');
const sequelize = require('../database');

const EvaluationProcedure =
sequelize.define('EvaluationProcedure', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  title: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
```

```

    },
    startDate: {
      type: DataTypes.DATE,
      allowNull: false
    },
    endDate: {
      type: DataTypes.DATE,
      allowNull: false
    },
    status: {
      type: DataTypes.ENUM('draft', 'active', 'completed'),
      defaultValue: 'draft'
    },
    createdBy: {
      type: DataTypes.INTEGER,
      allowNull: false
    }
  }
});

module.exports = EvaluationProcedure;

```

Листинг А.27 – Файл evaluation-service\models\Participant.js

```

const { DataTypes } = require('sequelize');
const sequelize = require('../database');

const Participant = sequelize.define('Participant', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  evaluationId: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  userId: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  role: {
    type: DataTypes.ENUM('self', 'manager', 'peer',
'subordinate'),
    allowNull: false
  },
  isCompleted: {
    type: DataTypes.BOOLEAN,
    defaultValue: false
  }
});

module.exports = Participant;

```

Листинг А.28 – Файл evaluation-service\controllers\evaluationController.js

```
const EvaluationService =
require('../services/evaluationService');

exports.createProcedure = async (req, res) => {
  try {
    const data = await
EvaluationService.createProcedure(req.body);
    res.status(201).json(data);
  } catch (e) {
    res.status(500).json({ message: e.message });
  }
};

exports.addParticipants = async (req, res) => {
  try {
    const data = await
EvaluationService.addParticipants(req.body.evaluationId,
req.body.participants);
    res.status(201).json(data);
  } catch (e) {
    res.status(500).json({ message: e.message });
  }
};
```

Листинг А.29 – Файл auth-service\server.js

```
const express = require('express');
const cors = require('cors');
require('dotenv').config();

const app = express();

app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true
}));
app.use(express.json());

const authRoutes = require('./routes/authRoutes');
app.use('/auth', authRoutes);

const PORT = process.env.PORT || 5001;
app.listen(PORT, () => {
  console.log(`Auth service работает на порту ${PORT}`);
});
```

Листинг А.30 – Файл auth-service\.env

```
PORT=5001
```

```
DB_URL=mssql://dp:pass1@localhost:1433/DiplomDB
JWT_SECRET=supersecretkey360
JWT_EXPIRES_IN=1d
```

Листинг А.31 – Файл auth-service\routes\authRoutes.js

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/register', authController.register);
router.post('/login', authController.login);

module.exports = router;
```

Листинг А.32 – Файл auth-service\models\index.js

```
const sequelize = require('../config/db');
const User = require('./user');

const initModels = async () => {
  await sequelize.sync();
};

module.exports = {
  sequelize,
  User,
  initModels
};
```

Листинг А.33 – Файл auth-service\models\user.js

```
const { DataTypes } = require('sequelize');
const sequelize = require('../config/db');

const User = sequelize.define('User', {
  name: { type: DataTypes.STRING, allowNull: false },
  email: { type: DataTypes.STRING, allowNull: false, unique:
true },
  password: { type: DataTypes.STRING, allowNull: false },
  role: { type: DataTypes.ENUM('employee', 'manager', 'hr',
'admin'), defaultValue: 'employee' }
});

module.exports = User;
```

Листинг А.34 – Файл auth-service\middleware\authMiddleware.js

```

const jwt = require('jsonwebtoken');
const { User } = require('../models');

const authMiddleware = (roles = []) => {
  return async (req, res, next) => {
    try {
      const token = req.headers.authorization?.split(" ")[1];
      if (!token) return res.status(401).json({ error: "Нет токена" });

      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = decoded;

      if (roles.length && !roles.includes(decoded.role)) {
        return res.status(403).json({ error: "Недостаточно прав" });
      }

      next();
    } catch (err) {
      return res.status(401).json({ error: "Неверный токен" });
    }
  };
};

module.exports = authMiddleware;

```

Листинг А.35 – Файл auth-service\controllers\authController.js

```

const { User } = require('../models');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

exports.register = async (req, res) => {
  try {
    const { name, email, password, role } = req.body;

    console.log("Введённый пароль:", password);
    console.log("Совпадает?", user.password);

    const exists = await User.findOne({ where: { email } });
    if (exists) return res.status(400).json({ error: 'Пользователь уже существует' });

    const hash = await bcrypt.hash(password, 10);
    const user = await User.create({ name, email, password: hash, role });

    res.json({ message: 'Пользователь зарегистрирован', user: {
      id: user.id, name: user.name, role: user.role } });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}

```

```

};

exports.login = async (req, res) => {
  try {
    console.log("JWT_SECRET:", process.env.JWT_SECRET);
    const { email, password } = req.body;
    const user = await User.findOne({ where: { email } });

    console.log("Введённый пароль:", password);
    console.log("Хеш из БД:", user?.password);
    const passwordMatch = await bcrypt.compare(password,
user.password);
    console.log("Совпадает?", passwordMatch);

    if (!user || !(await bcrypt.compare(password, user.password)))
    {
      return res.status(401).json({ error: 'Неверные учетные
данные' });
    }

    const token = jwt.sign(
      { id: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: process.env.JWT_EXPIRES_IN }
    );

    res.json({ token, user: { id: user.id, name: user.name, email:
user.email, role: user.role } });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

```

Листинг А.36 – Файл auth-service\config\db.js

```

const { Sequelize } = require('sequelize');

const sequelize = new Sequelize(process.env.DB_URL, {
  dialect: 'mssql',
  dialectOptions: {
    options: {
      encrypt: false,
    }
  }
});

module.exports = sequelize;

```

Листинг А.37 – Файл App.js

```

import { AuthContext } from '../context/AuthContext';
import { useContext } from 'react';
import { BrowserRouter as Router, Routes, Route, NavLink } from
'react-router-dom';
import UsersPage from '../pages/UsersPage';
import RecruitmentPage from '../pages/RecruitmentPage';
import EvaluationPage from '../pages/EvaluationPage';
import LoginPage from '../pages/LoginPage';
import PrivateRoute from '../components/PrivateRoute';
import UserBar from '../components/UserBar';
import ReportPage from '../pages/ReportPage';
import './App.css'; // Подключаем стили

function App() {
  const { user } = useContext(AuthContext);

  return (
    <Router>
      <div className="app-container">
        {user && (
          <>
            <nav className="nav-bar">
              <NavLink to="/" end className="nav-
link">Пользователи</NavLink>
              <NavLink to="/recruitment" className="nav-
link">Найм</NavLink>
              <NavLink to="/evaluation" className="nav-
link">Оценка</NavLink>
              <NavLink to="/reports" className="nav-
link">Отчёты</NavLink>
            </nav>

            <UserBar />
          </>
        )}

        <Routes>
          <Route path="/login" element={<LoginPage />} />

          {/* Защищенные маршруты */}
          <Route path="/" element={
            <PrivateRoute>
              <UsersPage />
            </PrivateRoute>
          } />

          <Route path="/recruitment" element={
            <PrivateRoute>
              <RecruitmentPage />
            </PrivateRoute>
          } />

          <Route path="/evaluation" element={

```

```

        <PrivateRoute>
          <EvaluationPage />
        </PrivateRoute>
      </>

      <Route path="/reports" element={
        <PrivateRoute>
          <ReportPage />
        </PrivateRoute>
      } />
    </Routes>
  </div>
</Router>
);
}

export default App;

```

Листинг А.38 – Файл pages/EvaluationPage.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import '../css/EvaluationPage.css';

const EvaluationPage = () => {
  const [employees, setEmployees] = useState([]);
  const [selectedEmployeeId, setSelectedEmployeeId] =
    useState('');
  const [rating, setRating] = useState(3);
  const [comment, setComment] = useState('');
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    async function fetchEmployees() {
      try {
        const res = await
        axios.get('http://localhost:5002/employees');
        setEmployees(res.data);
      } catch (err) {
        alert('Ошибка загрузки сотрудников: ' +
        (err.response?.data?.error || err.message));
      }
    }
    fetchEmployees();
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!selectedEmployeeId) {
      alert('Выберите сотрудника для оценки');
      return;
    }
  }

```

```

    setLoading(true);
    try {
      await axios.post('http://localhost:5003/evaluations', {
        employeeId: selectedEmployeeId,
        rating,
        comment
      });
      alert('Оценка сохранена');
      setRating(3);
      setComment('');
      setSelectedEmployeeId('');
    } catch (err) {
      alert('Ошибка при сохранении оценки: ' +
        (err.response?.data?.error || err.message));
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="evaluation-container">
      <h2 className="page-title">Оценка сотрудников</h2>

      <form className="evaluation-form" onSubmit={handleSubmit}>
        <label>
          Сотрудник:
          <select
            value={selectedEmployeeId}
            onChange={e =>
              setSelectedEmployeeId(e.target.value)}
            required
          >
            <option value="" disabled>Выберите сотрудника</option>
            {employees.map(emp => (
              <option key={emp.id} value={emp.id}>
                {emp.name} ({emp.role})
              </option>
            ))}
          </select>
        </label>

        <label>
          Оценка (1-5):
          <input
            type="number"
            min="1"
            max="5"
            value={rating}
            onChange={e => setRating(Number(e.target.value))}
            required
          />
        </label>
      </form>
    </div>
  );

```

```

        <label>
            Комментарий:
            <textarea
                value={comment}
                onChange={ (e) => setComment(e.target.value) }
                placeholder="Ваш комментарий"
            />
        </label>

        <button type="submit" disabled={loading}>
            {loading ? 'Сохраняем...' : 'Отправить оценку'}
        </button>
    </form>
</div>
);
};

export default EvaluationPage;

```

Листинг А.39 – Файл pages\RecruitmentPage.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import '../css/RecruitmentPage.css';

const baseUrl = 'http://localhost:5002';

function RecruitmentPage() {
    const [vacancies, setVacancies] = useState([]);
    const [form, setForm] = useState({ title: '', description: '',
status: 'open' });
    const [loading, setLoading] = useState(false);

    useEffect(() => {
        fetchVacancies();
    }, []);

    async function fetchVacancies() {
        setLoading(true);
        try {
            const res = await axios.get(`${baseUrl}/vacancies`);
            setVacancies(res.data);
        } catch (err) {
            alert('Ошибка при загрузке вакансий');
        } finally {
            setLoading(false);
        }
    }

    async function handleSubmit(e) {
        e.preventDefault();
        try {

```

```

    const res = await axios.post(`${baseUrl}/vacancies`, form);
    setVacancies(prev => [...prev, res.data]);
    setForm({ title: '', description: '', status: 'open' });
  } catch (err) {
    alert('Ошибка при добавлении вакансии');
  }
}

return (
  <div className="recruitment-container">
    <h2 className="page-title">Управление вакансиями</h2>

    <form className="vacancy-form" onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Название вакансии"
        value={form.title}
        onChange={e => setForm({ ...form, title: e.target.value
    }}}
      </input>

      <textarea
        placeholder="Описание вакансии"
        value={form.description}
        onChange={e => setForm({ ...form, description:
e.target.value })}
      </textarea>

      <select
        value={form.status}
        onChange={e => setForm({ ...form, status: e.target.value
    }}}
      >
        <option value="open">Открыта</option>
        <option value="closed">Закрыта</option>
      </select>

      <button type="submit">Добавить вакансию</button>
    </form>

    {loading ? (
      <p className="loading-text">Загрузка вакансий...</p>
    ) : (
      <div className="vacancy-list">
        {vacancies.map(vac => (
          <div key={vac.id} className="vacancy-card">
            <h4>{vac.title}</h4>
            <span className={`status-label
${vac.status}`}>{vac.status === 'open' ? 'Открыта' :
'Закрыта'}</span>
            <p>{vac.description}</p>
          </div>
        ))}
      </div>
    )}
  </div>
);

```

```

        )))
      </div>
    )}
  </div>
);
}

export default RecruitmentPage;
import React, { useState, useContext } from 'react';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import '../css/LoginPage.css'; // Стили создадим позже

const LoginPage = () => {
  const { login } = useContext(AuthContext);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    try {
      await login(email, password);
      navigate('/'); // после входа — на основную страницу
    } catch (err) {
      console.error('Ошибка при входе:', err.response?.data || err.message);
      setError('Неверный логин или пароль');
    }
  };

  return (
    <div className="login-container">
      <h2>Вход в систему</h2>
      <form onSubmit={handleSubmit} className="login-form">
        <input
          type="email"
          placeholder="Email"
          required
          value={email}
          onChange={e => setEmail(e.target.value)}
        />
        <input
          type="password"
          placeholder="Пароль"
          required
          value={password}
          onChange={e => setPassword(e.target.value)}
        />
        <button type="submit">Войти</button>
        {error && <p className="error-text">{error}</p>}
      </form>
    </div>
  );
};

```

```

        </form>
      </div>
    );
  };

export default LoginPage;

```

Листинг А.40 – Файл pages/ReportPage.jsx

```

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import '../css/ReportPage.css';

const ReportPage = () => {
  const [employees, setEmployees] = useState([]);
  const [departments, setDepartments] = useState([]);
  const [selectedEmployeeId, setSelectedEmployeeId] =
    useState('');
  const [selectedDepartmentId, setSelectedDepartmentId] =
    useState('');
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    async function fetchEmployees() {
      try {
        const res = await
axios.get('http://localhost:5002/employees');
        setEmployees(res.data);
      } catch (err) {
        alert('Ошибка загрузки сотрудников: ' + err.message);
      }
    }

    async function fetchDepartments() {
      try {
        const res = await
axios.get('http://localhost:5002/departments'); // Заменить при
необходимости
        setDepartments(res.data);
      } catch (err) {
        console.warn('Не удалось загрузить подразделения
(опционально) ');
      }
    }

    fetchEmployees();
    fetchDepartments();
  }, []);

  const generateEmployeeReport = async () => {
    if (!selectedEmployeeId) {
      alert('Выберите сотрудника');
      return;
    }
  };

```

```

    }
    setLoading(true);
    try {
        const res = await
axios.get(`http://localhost:5005/reports/employee/${selectedEmployeeId}`);
        alert('Отчет сформирован: ' + res.data.message);
    } catch (err) {
        alert('Ошибка генерации отчета: ' + err.message);
    } finally {
        setLoading(false);
    }
};

const generateDepartmentReport = async () => {
    if (!selectedDepartmentId) {
        alert('Выберите подразделение');
        return;
    }
    setLoading(true);
    try {
        const res = await
axios.get(`http://localhost:5005/reports/department/${selectedDepartmentId}`);
        alert('Отчет сформирован: ' + res.data.message);
    } catch (err) {
        alert('Ошибка генерации отчета: ' + err.message);
    } finally {
        setLoading(false);
    }
};

return (
    <div className="report-container">
        <h2 className="page-title">Генерация отчётов</h2>

        <div className="report-section">
            <h3>По сотруднику</h3>
            <select value={selectedEmployeeId} onChange={(e) =>
setSelectedEmployeeId(e.target.value)}>
                <option value="">Выберите сотрудника</option>
                {employees.map(emp => (
                    <option key={emp.id}
value={emp.id}>{emp.name}</option>
                ))}
            </select>
            <button onClick={generateEmployeeReport}
disabled={loading}>
                {loading ? 'Генерируется...' : 'Сформировать отчет'}
            </button>
        </div>

        <div className="report-section">

```

```

        <h3>По подразделению</h3>
        <select value={selectedDepartmentId} onChange={(e) =>
setSelectedDepartmentId(e.target.value)}>
            <option value="">Выберите подразделение</option>
            {departments.map(dep => (
                <option key={dep.id}
value={dep.id}>{dep.name}</option>
            ))}
        </select>
        <button onClick={generateDepartmentReport}
disabled={loading}>
            {loading ? 'Генерируется...' : 'Сформировать отчет'}
        </button>
    </div>
</div>
);
};

export default ReportPage;

```

Листинг А.41 – Файл pages\UsersPage.jsx

```

import React from 'react';
import AddEmployeeForm from '../components/AddEmployeeForm';
import EmployeeList from '../components/EmployeeList';

const UsersPage = () => {
    return (
        <div>
            <h2>Users Page</h2>
            <AddEmployeeForm />
            <EmployeeList />
        </div>
    );
};

export default UsersPage;

```

Листинг А.42 – Файл routes\PrivateRoute.js

```

import { Navigate } from 'react-router-dom';
import { useContext } from 'react';
import { AuthContext } from '../context/AuthContext';

const PrivateRoute = ({ children, roles }) => {
    const { user } = useContext(AuthContext);

    if (!user) return <Navigate to="/login" />;
    if (roles && !roles.includes(user.role)) return <h2>Доступ
запрещен</h2>;

```

```
    return children;
};

export default PrivateRoute;
```

Приложение Б
(рекомендуемое)
Изображения

Вход в систему

admin@company.com

.....

Войти

или

Зарегистрироваться

Рисунок Б.1 – Форма авторизации

Пользователи Найм **Оценка** Отчёты

Вы вошли как: admin@company.com | Роль: admin Выйти

Оценка сотрудников + Создать процедуру оценки

Процедура оценки:
Выберите процедуру

Сотрудник:
Выберите сотрудника

Оценка (1–5):
3

Комментарий:
Ваш комментарий

Отправить оценку

Рисунок Б.2 – Форма оценки

Пользователи

Найм

Оценка

Отчёты

Вы вошли как: **admin@company.com** | Роль: **admin**

Выйти

Управление вакансиями

Название вакансии

Описание вакансии

Открыта

Добавить вакансию

Директор

Открыта

Директор

Рисунок Б.3 – Форма управления вакансиями

Отчёт об оценке

Сотрудник: Петров Петр Петрович
Email: petrov@mail.ru
Оценка: 3
Комментарий: 456456
Дата: 08.06.2025, 19:19:59

Рисунок Б.4 – PDF-файл отчета об оценке