

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Информационная система карьерного развития
и оценки сотрудников ИТ-компаний**

Студент	_____	Кареев Андрей Андреевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

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, 2026

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

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Кареев Андрей Андреевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система карьерного развития и оценки сотрудников ИТ-компаний

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы доцент кафедры «МОиПЭВМ», доцент, к.т.н., Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т. п.)

1) *Назначение: информационная система карьерного развития и оценки сотрудников ИТ-компаний, обеспечивающая возможность отслеживать профессиональный рост работников и принимать кадровые решения.*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: JavaScript, Java, CSS, SQL, UML.

Инструментальные средства: WebStorm 2025, IntelliJ IDEA 2025, DBeaver, Visual Paradigm 17.3, Docker Desktop.

3) *Основные функции приложения:*

а) создание и ведение индивидуальных планов развития;

б) осуществление кадровых решений;

в) оценка профессионального роста и работы сотрудников;

г) управление должностями, квалификационными уровнями и компетенциями.

5. Перечень подлежащих разработке вопросов

1) *Анализ предметной области и постановка задачи;*

2) *Проектирование системы;*

3) *Реализация программных средств;*

4) *Контроль качества программного обеспечения;*

5) *Оформление пояснительной записки.*

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

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 113 страниц, 25 рисунков, 12 таблиц, 8 листингов, 23 использованных источника и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, КАРЬЕРНОЕ РАЗВИТИЕ СОТРУДНИКОВ, ИНДИВИДУАЛЬНЫЙ ПЛАН РАЗВИТИЯ, КВАЛИФИКАЦИОННЫЙ УРОВЕНЬ, КОМПЕТЕНЦИИ, КАДРОВОЕ РЕШЕНИЕ, JAVA, SPRING BOOT, REACT, POSTGRESQL, UML

Объект исследования – информационная система карьерного развития сотрудников.

Цель выпускной квалификационной работы – спроектировать и разработать информационную систему карьерного развития сотрудников в ИТ-компании.

Технологии разработки – язык программирования Java и фреймворк Spring Boot для реализации серверной части, язык программирования JavaScript и технология React для реализации пользовательского интерфейса, система управления базами данных PostgreSQL, средство миграции базы данных Flyway, брокер сообщений Apache Kafka, объектное хранилище MinIO, средство визуального проектирования Visual Paradigm, инструмент проектирования пользовательского интерфейса Figma, среда разработки IntelliJ IDEA и WebStorm.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – разработана информационная система карьерного развития сотрудников, обеспечивающая управление квалификационными уровнями, индивидуальными планами развития, собеседованиями на повышение, кадровыми решениями и отчетностью.

					ПГУ 09.03.04 – 08BP222.13 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Кареев А.А.			Информационная система карьерного развития и оценки сотрудников ИТ-компании Пояснительная записка	Лит.	Лист	Листов
Пров.		Такташкин Д.В					4	113
Реценз.						Группа 22ВП2		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку.....	8
1.1 Анализ предметной области	8
1.2 Обзор и анализ программных продуктов-аналогов	9
1.3 Постановка задачи на разработку приложения	10
1.4 Анализ требований к системе.....	11
1.4.1 Анализ функциональных требований	11
1.4.2 Анализ нефункциональных требований	20
1.5 Планирование разработки и оценка бюджета.....	22
2 Проектирование системы	25
2.1 Проектирование архитектуры системы.....	25
2.2 Проектирование структур данных	27
2.2.1 Проектирование концептуальной модели базы данных.....	27
2.2.2 Проектирование логической модели базы данных.....	29
2.3 Проектирование серверной части	31
2.4 Проектирование пользовательского интерфейса	34
3 Реализация программных средств.....	37
3.1 Реализация слоя базы данных.....	37
3.2 Реализация серверной части приложения	42
3.3 Реализация слоя представления	44
4 Тестирование системы карьерного развития сотрудников	47
4.1 Обоснование выбора методов тестирования	47
4.2 Тестирование методом черного ящика.....	48
4.2.1 Классы эквивалентности.....	48
4.2.2 Функциональная диаграмма и таблица решений.....	50
4.3 Расчет метрик качества программного кода	53
4.4 Анализ метрик программного кода.....	58
4.5 Модульное тестирование	60

Заключение	61
Список использованных источников	63
Приложение А (справочное) Глоссарий	66
Приложение Б (обязательное) Листинг программы	68
Приложение В (рекомендуемое) Листинг модульных тестов.....	109

Введение

Для ИТ-компаний важно отслеживать развитие сотрудников, но чаще всего это становится затруднительным так как в компании может быть большое количество работников, а их профессиональный рост нужно контролировать на протяжении продолжительного промежутка времени.

Оценка роста сотрудника, основанная только на решении руководителя, может быть неточной из-за субъективного восприятия. Для решения данной проблемы важно автоматизировать процессы анализа роста сотрудников. Это поможет создать унифицированные условия продвижения сотрудников по карьерной лестнице. Для этого система должна обладать следующим функционалом: ведение справочника квалификационных уровней, создание индивидуальных планов развития, принятие и хранение кадровых решений, контроль выполнения задач сотрудником и оценка завершившихся планов развития.

Объект исследования – средства автоматизации отслеживания и обеспечения профессионального роста сотрудников.

Предмет исследования – программные средства для отслеживания, поддержки и оценки карьерного развития сотрудников ИТ-компаний.

Целью выпускной квалификационной работы является проектирование и разработка информационной системы карьерного развития сотрудников ИТ-компаний. Для ее достижения нужно нужно проанализировать предметную область, проанализировать продукты-аналоги, чтобы определить особенности схожих систем, определить требования, спроектировать и реализовать программные средства системы, а также обеспечить контроль качества программного обеспечения разными методами тестирования.

Разработанная система должна помочь в создании структуры квалификационных уровней, ведении индивидуальных планов развития и карьерного роста сотрудников и помочь в оценке развития сотрудников.

1 Анализ предметной области и постановка задачи на разработку

1.1 Анализ предметной области

Предметная область выпускной квалификационной работы связана с процессами карьерного развития сотрудников в ИТ-компаниях, где профессиональный рост должен опираться на понятную квалификационную модель, систему квалификационных уровней и формализованные требования к компетенциям сотрудников. В условиях постоянного обновления технологий, инструментов и подходов к разработке, особую значимость приобретает не только техническая база организации, но и наличие прозрачного механизма оценки текущего уровня специалиста, определения направлений его дальнейшего развития и планирования карьерного продвижения. Одним из основных инструментов такого развития выступает индивидуальный план развития, в рамках которого фиксируются цели сотрудника, задачи, сроки их выполнения и ожидаемые результаты. В дальнейшем в тексте выпускной квалификационной работы индивидуальный план развития может обозначаться сокращением ИПР.

Компетенция – совокупность знаний, умений, навыков и способов деятельности, необходимых для эффективного выполнения задач в определенной профессиональной области [1].

Квалификационный уровень – ступень профессионального развития, характеризующая совокупностью требований к знаниям, умениям, компетенциям, ответственности и самостоятельности работника при выполнении трудовой деятельности [2].

Индивидуальный план развития – документ, в котором фиксируются мероприятия по профессиональному развитию конкретного сотрудника, направленные на развитие необходимых компетенций, достижение карьерных целей и контроль результатов развития [3].

Анализ предметной области показывает, что процессы карьерного развития сотрудников в компании являются сложными, многоэтапными и тесно

связанными между собой. Это обуславливает необходимость создания единой информационной системы, которая обеспечит централизованное хранение данных, поддержку бизнес-процессов развития сотрудников, разграничение прав доступа между участниками и формирование аналитической отчетности для руководства.

1.2 Обзор и анализ программных продуктов-аналогов

Перед разработкой информационной системы карьерного развития сотрудников целесообразно рассмотреть уже существующие программные решения в данной области. Такой анализ позволяет определить, какие функции чаще всего встречаются в подобных системах, какие подходы используются для организации карьерных дорожных карт, индивидуальных планов развития, оценки компетенций и поддержки управленческих решений. Кроме того, сравнение аналогов помогает выделить особенности, которые необходимо учесть при проектировании собственной системы.

Для сравнения были определены критерии, отражающие основные задачи разрабатываемой информационной системы:

- поддержка формализованного собеседования на повышение;
- возможность отслеживания прогресса развития;
- работа с компетенциями сотрудников;
- управление вложениями и комментариями в задачах ИПР;
- отчетность по выполнению ИПР и карьерным решениям;
- разграничение доступа по ролям;
- наличие квалификационной модели и карьерных уровней.

Оценка функционала аналогов проводилась по следующей шкале:

- 0 баллов – функционал полностью отсутствует;
- 1 балл – функционал реализован частично;
- 2 балла – функционал полностью реализован.

Для сравнения были выбраны следующие системы:

- Lattice Grow;

- Websoft HCM;
- Grade Factor.

Сравнение перечисленных выше систем представлено в таблице 1.

Таблица 1 – Сравнительный анализ

Критерий	Lattice Grow	Websoft HCM	Grade Factor
Поддержка формализованного собеседования на повышение	0	1	1
Возможность отслеживания прогресса развития	2	2	2
Работа с компетенциями сотрудников	2	2	2
Управление вложениями и комментариями в задачах ИПР	1	1	1
Отчетность по выполнению ИПР и карьерным решениям	1	2	2
Ориентация на специализированный процесс IT-компаний	1	0	1

Проведенное сравнение показывает, что существующие решения обладают развитым функционалом в области карьерного развития сотрудников. Наиболее полно рассматриваемые критерии представлены в Websoft HCM и Grade Factor, так как они охватывают не только индивидуальные планы развития, но и оценку, компетенции, карьерные маршруты, аналитику и разграничение доступа. Lattice Grow также хорошо подходит для управления развитием сотрудников, однако в большей степени ориентирован на карьерные треки и развитие, чем на полный цикл согласования повышения.

Разрабатываемая информационная система должна учитывать сильные стороны рассмотренных аналогов, но при этом быть ориентированной на конкретный процесс карьерного развития в IT-компаниях.

Полученные результаты анализа используются при формировании требований к системе и дальнейшем проектировании ее функциональности.

1.3 Постановка задачи на разработку приложения

На основе анализа предметной области была сформирована цель – спроектировать и разработать информационную систему карьерного развития

сотрудников в ИТ-компании. Для выполнения этой цели требуется решить следующие задачи:

- проанализировать предметную область;
- спроектировать систему;
- реализовать программные средства системы карьерного развития;
- протестировать систему разными методами.

Таким образом, сформулированные цель и задачи определяют основное направление выполнения выпускной квалификационной работы.

1.4 Анализ требований к системе

1.4.1 Анализ функциональных требований

Система предназначена для автоматизации процессов карьерного развития сотрудников: ведения индивидуальных планов развития, контроля выполнения задач, учета компетенций, проведения собеседований на повышение, хранения данных о кадровых решениях и формирования отчетности по эффективности выполнения индивидуальных планов развития.

С учетом особенностей предметной области разрабатываемая система должна решать следующие задачи:

- обеспечивать хранение и обработку данных о подразделениях и сотрудниках компании;
- поддерживать ведение квалификационной модели, включающей должности, квалификационные уровни и связанные с ними компетенции;
- создавать и сопровождать индивидуальные планы развития сотрудников с указанием целей, задач, сроков выполнения и ожидаемых результатов;
- обеспечивать фиксацию прогресса по задачам плана развития, а также подтверждение освоения компетенций;
- поддерживать процедуры согласования индивидуального плана развития между участниками процесса;

- обеспечивать планирование и проведение собеседований на повышение;
- фиксировать кадровые решения по результатам оценки сотрудника и вести аудит данных событий;
- формировать аналитические отчеты по выполнению планов развития и динамике карьерного продвижения сотрудников.

С учетом реализуемых бизнес-процессов в системе выделены четыре основные роли пользователей:

- сотрудник – работает с собственным индивидуальным планом развития, фиксирует прогресс выполнения задач, просматривает комментарии, вложения, компетенции, сведения о себе, отделе, квалификационных уровнях и политиках карьерного продвижения;
- руководитель команды – отвечает за создание и сопровождение индивидуальных планов развития сотрудников, подтверждение выполнения задач и достижения компетенций, управление задачами, комментариями и вложениями, а также просмотр отчетов об эффективности сотрудников;
- директор отдела – выполняет функции контроля и принятия управленческих решений в рамках отдела, ему доступны просмотр отчетности, управление собеседованиями на карьерное продвижение, одобрение или отклонение повышения, управление квалификационными уровнями, должностями и компетенциями;
- генеральной директор – этот пользователь управляет политиками повышения, учетными записями сотрудников компании, отделами, квалификационными уровнями, должностями и компетенциями.

Полученный перечень задач отражает ключевые бизнес-процессы карьерного развития сотрудников и является основой для дальнейшего проектирования информационной системы.

На основе определенных ролей пользователя и выполняемых задач была разработана диаграмма вариантов использования.

Диаграмма вариантов использования служит для выявления функциональных требований к системе и иллюстрирует стандартные взаимодействия между пользователями и системой [4]. Она позволяет определить, какие действия будут выполнять пользователи, какие варианты использования являются основными, а какие расширяют или дополняют другие сценарии.

Так как в разрабатываемой системе предусмотрено несколько категорий пользователей с различными правами доступа, диаграммы вариантов использования были подготовлены отдельно для каждой роли.

На рисунке 1 представлена диаграмма вариантов использования для роли «Сотрудник».

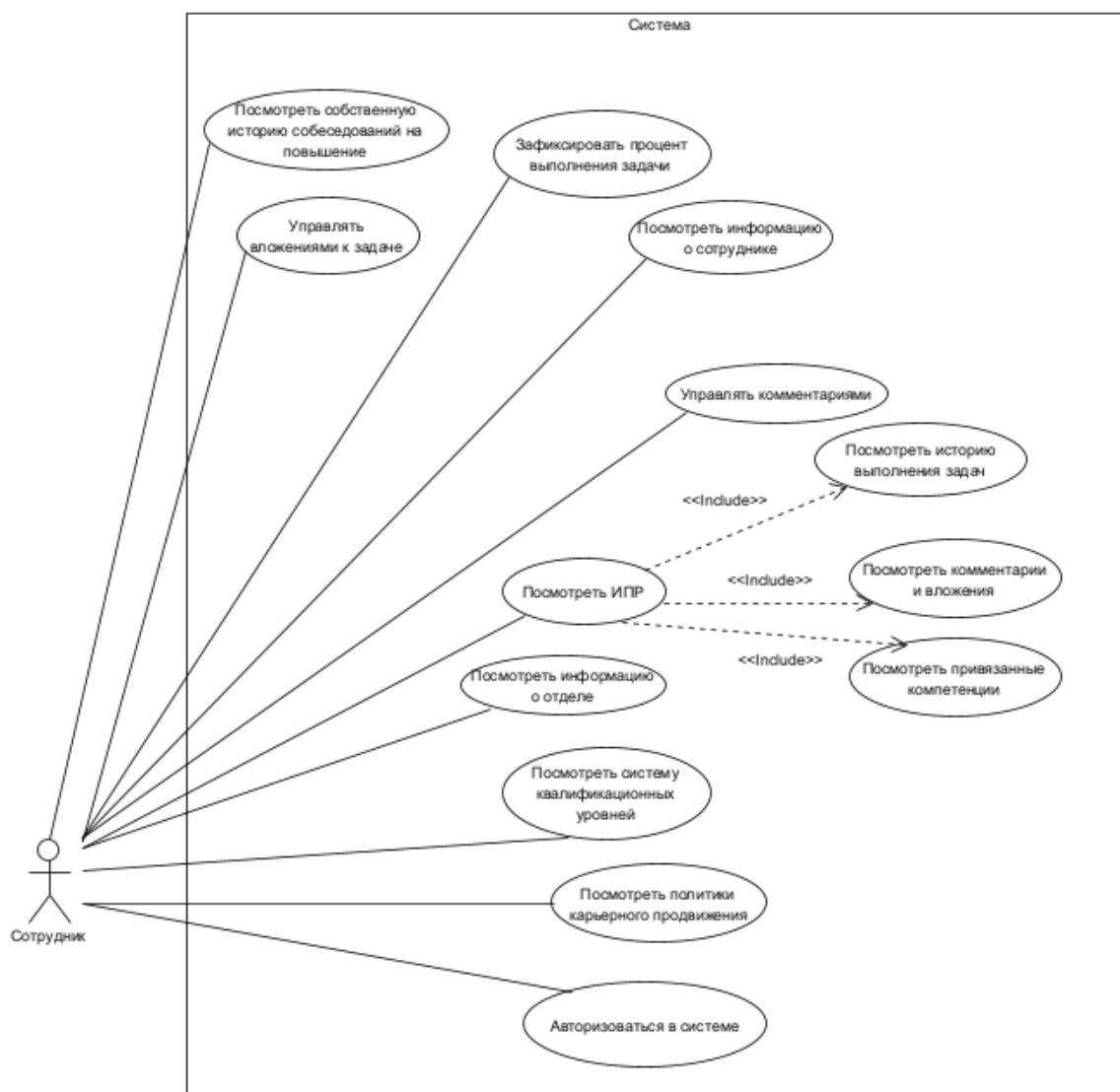


Рисунок 1 – Диаграмма вариантов для роли «Сотрудник»

Основными вариантами использования для этого пользователя являются взаимодействия с ИПР, такие как просмотр, оставление комментариев и фиксация прогресса.

Далее была разработана диаграмма вариантов использования для последующей роли в системе.

На рисунке 2 представлена диаграмма вариантов использования для роли «Генеральный директор».

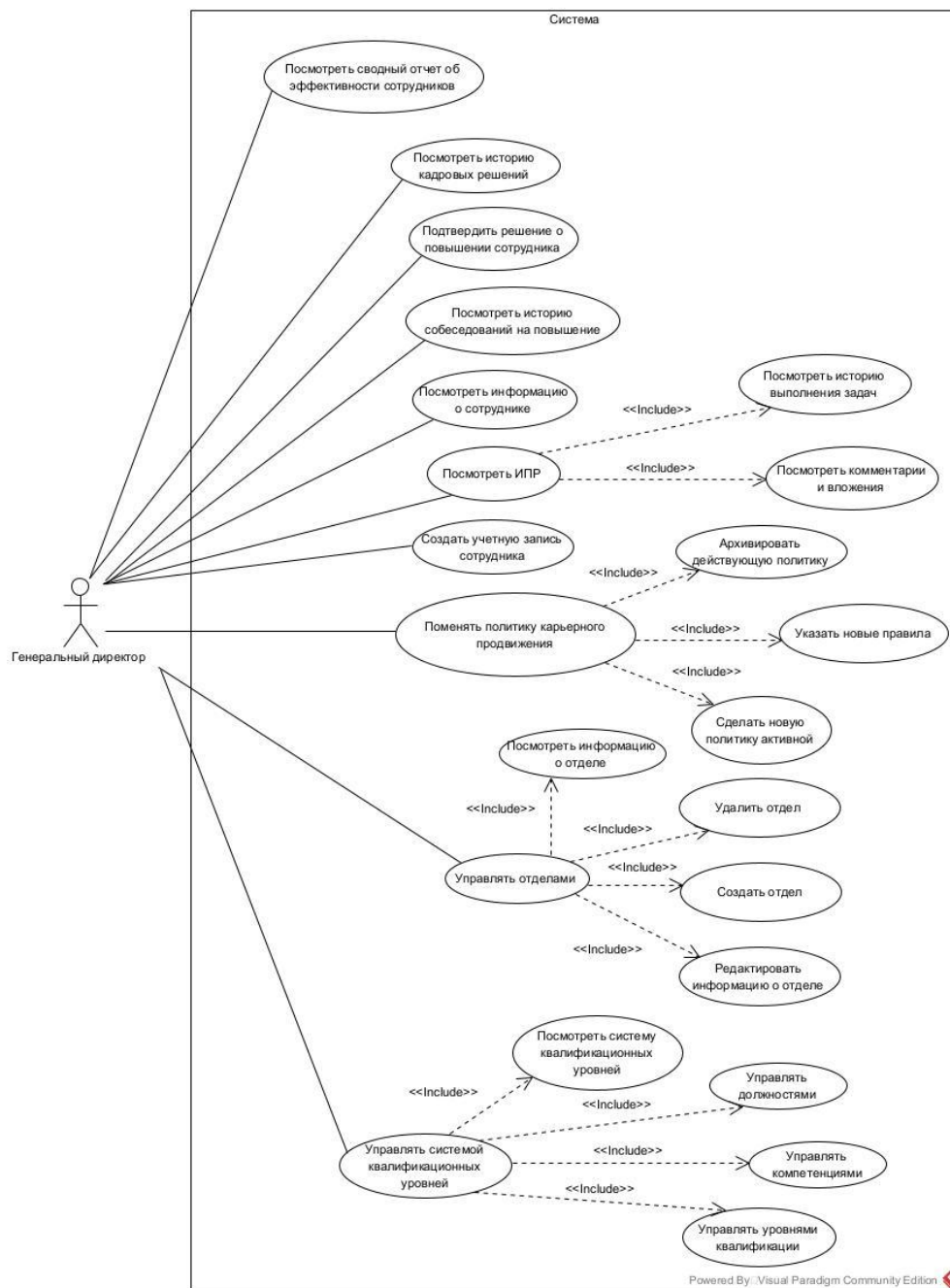


Рисунок 2 – Диаграмма вариантов использования для роли «Генеральный директор»

Наиболее широкий набор управленческих функций содержит диаграмма вариантов использования генерального директора. Основными являются подтверждение решений о повышении, управление отделами, политиками карьерного продвижения и системой квалификационных уровней.

На рисунке 3 представлена диаграмма вариантов использования для роли «Директор отдела».



Рисунок 3 – Диаграмма вариантов использования для роли «Директор отдела»

Характерными для директора отдела являются функции назначения собеседований, принятия решений о повышении, работы с квалификационными уровнями и просмотра аналитической информации.

На рисунке 4 представлена диаграмма вариантов использования для роли «Руководитель команды».



Рисунок 4 – Диаграмма вариантов использования для роли «Руководитель команды»

Руководитель команды в основном взаимодействует с сотрудниками и их индивидуальными планами развития.

Для более детального описания ключевых функций наиболее значимых сценариев применения были подготовлены их спецификации.

Сценарий варианта использования «Зафиксировать процент выполнения задачи» представлен на рисунке 5.

Наименование: Зафиксировать процент выполнения задачи
ID: 1
Краткое описание: сотрудник фиксирует процент выполнения задачи из индивидуального плана развития для учета результатов
Действующие лица: сотрудник
Предусловие: <ol style="list-style-type: none"> 1. Сотрудник авторизован в системе. 2. Сотрудник имеет активный индивидуальный план развития. 3. В ИПР есть задача, доступная для обновления прогресса.
Основной поток: <ol style="list-style-type: none"> 1. Сотрудник открывает индивидуальный план развития. 2. Сотрудник выбирает задачу из списка. 3. Система отображает карточку задачи. 4. Сотрудник вводит процент выполнения задачи. 5. Система проверяет корректность введенного значения. 6. Сотрудник нажимает кнопку сохранения. 7. Система сохраняет новый показатель прогресса. 8. Система обновляет историю выполнения задачи. Альтернативный поток: <ol style="list-style-type: none"> 5а. Если пользователь вводит значение меньше 0 или больше 100, система подсвечивает поле и сообщает о допустимом диапазоне значений. 5б. Если пользователь вводит нечисловое значение, система не принимает введенные данные. 6а. Если задача находится в статусе, при котором изменение прогресса недоступно, элементы ввода и кнопка сохранения не отображаются или становятся недоступными. 6б. Если пользователь оставляет поле процента выполнения пустым, система не позволяет сохранить изменение.
Постусловие: процент выполнения задачи сохранен

Рисунок 5 – Сценарий варианта использования «Зафиксировать процент выполнения задачи»

Сценарий варианта использования «Создать ИПР» представлен на рисунке 6.

Наименование: Создать ИПР
ID: 2
Краткое описание: руководитель команды создает индивидуальный план развития для сотрудника
Действующие лица: руководитель команды, сотрудник
Предусловие: <ol style="list-style-type: none"> 1. Руководитель команды авторизован в системе. 2. Руководитель команды открыл раздел индивидуальных планов развития. 3. В интерфейсе доступна форма создания ИПР.
Основной поток: <ol style="list-style-type: none"> 1. Руководитель команды открывает раздел индивидуальных планов развития. 2. Руководитель нажимает кнопку создания нового ИПР. 3. Система отображает форму создания ИПР. 4. Руководитель выбирает сотрудника. 5. Руководитель указывает основные параметры ИПР. 6. Руководитель добавляет задачи и связанные компетенции. 7. Система проверяет заполнение обязательных полей. 8. Руководитель подтверждает создание ИПР. 9. Система создает индивидуальный план развития. Альтернативный поток: <ol style="list-style-type: none"> 4а. Если пользователь не выбрал сотрудника, система не позволяет перейти к созданию ИПР или сохранить форму. 6а. Если пользователь не добавил задачи в ИПР, система сообщает о необходимости заполнить список задач.
Постусловие: индивидуальный план развития создан.

Рисунок 6 – Сценарий варианта использования «Создать ИПР»

Сценарий варианта использования «Назначить собеседование на карьерное продвижение» представлен на рисунке 7.

Прецедент: Создавать продукцию
ID: 1
Краткое описание: Пользователь/поставщик создает новый товар
Главные актеры: Пользователь/поставщик
Второстепенные актеры: Нет
Предусловия: 1. Авторизован в системе 2. Находится на странице управления своими товарами
Основной поток: 1. Просматривает список своих товаров 2. Выбирает действие – добавить новый товар 3. Заполняет необходимые поля 4. Сохраняет изменения и отправляет новый товар на модерацию Альтернативные потоки: 3а. Введены некорректные данные, получает сообщение об ошибке валидации 3б. Отправка на модерацию не удалась, получает уведомление 4а. Закрывает окно, не сохранив изменения
Постусловия: 1. Создан новый продукт, информация о товарах обновлена,

Рисунок 7 – Сценарий варианта использования «Создавать продукцию»

Спецификации вариантов использования позволяют описать логику взаимодействия пользователя с системой на уровне отдельных сценариев. В них фиксируются наименование и идентификатор прецедента, краткое описание его назначения, действующие лица, предусловия, основной поток выполнения, возможные альтернативные потоки и постусловия. Такое представление дает возможность структурированно описать поведение системы при выполнении ключевых операций и служит основой для последующего проектирования программных компонентов и сценариев тестирования.

В результате анализа функциональных требований были выделены основные варианты использования разрабатываемой системы. Для наиболее значимых сценариев подготовлены спецификации, позволяющие описать последовательность действий пользователя и реакцию системы при выполнении соответствующих операций.

1.4.2 Анализ нефункциональных требований

Нефункциональные требования определяют эксплуатационные характеристики системы и имеют не менее критическое значение, чем требования функциональные. Согласно подходу к разработке требований, нефункциональные требования определяют ограничения и характеристики качества, которым должна соответствовать система при выполнении своих функций [5]. Кроме того, при оценке качества программного обеспечения обычно учитываются следующие характеристики:

- безопасность;
- производительность;
- надежность;
- сопровождаемость.

При разработке системы используется разделение на клиентскую и серверную части. Такой подход позволяет отделить пользовательский интерфейс от бизнес-логики и упростить дальнейшее сопровождение приложения.

Для корректной работы клиента разработанная система должна корректно воспроизводиться на актуальных версиях следующих браузеров: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Яндекс Браузер.

Серверная часть реализуется на языке Java 21 с использованием фреймворка Spring Boot 3.2.8 [6]. В составе данного фреймворка используется встроенный контейнер сервлетов Apache Tomcat 10.1.26, обеспечивающий выполнение веб-приложения и обработку HTTP-запросов. В разрабатываемой системе серверная часть отвечает за выполнение основных бизнес-процессов. Через серверное приложение выполняется проверка входных данных, применение правил доступа, обработка запросов клиентской части и формирование ответов для пользовательского интерфейса.

Для постоянного хранения данных используется PostgreSQL 15 [7], а подключение к СУБД со стороны приложения осуществляется с помощью JDBC-драйвера PostgreSQL 42.6.2. Эта система управления базами данных

подходит для работы со структурированными данными и поддерживает механизмы обеспечения целостности, транзакционности и связей между таблицами. В рамках проектируемой системы это особенно важно, поскольку данные о сотрудниках, ИПР, задачах, компетенциях и кадровых решениях имеют большое количество взаимосвязей и должны сохраняться согласованно.

Изменение структуры базы данных выполняется с использованием миграций Flyway 9.22.3 [8]. Миграции позволяют фиксировать каждое изменение схемы базы данных в виде отдельного скрипта и применять его последовательно при развитии проекта. Такой подход помогает контролировать версии структуры данных, упрощает развертывание приложения в новой среде и снижает риск расхождения схемы базы данных между различными экземплярами системы.

Клиентская часть реализована в виде веб-приложения на React 19.2.6 с использованием React Router DOM 7.15.0, а сборка и запуск в среде разработки выполняются средствами Vite 8.0.12 [9]. Обмен данными между клиентом и сервером выполняется через REST API [10]. Для входа в систему и разграничения прав доступа используются JWT-токены и роли пользователей [11]. Файлы, прикрепляемые к задачам индивидуального плана развития, сохраняются в MinIO [12] версии 8.5.17. Для обмена событиями между сервисом карьерного развития и сервисом пользователей используется Apache Kafka 3.7 [13], а взаимодействие приложения с брокером сообщений реализовано через библиотеку Spring Kafka, управляемую экосистемой Spring Boot.

С точки зрения инфраструктурных нефункциональных требований хост-сервер, на котором разворачиваются серверные компоненты системы, должен обладать достаточными ресурсами для запуска двух Spring Boot сервисов, СУБД PostgreSQL, брокера Kafka и объектного хранилища MinIO. В минимальной конфигурации может использоваться сервер под управлением операционной системы семейства Linux или Windows, имеющий не менее 2 виртуальных или физических ядер процессора, не менее 8 ГБ оперативной памяти и постоянной объемом не менее 40 ГБ. При этом целесообразно предусмотреть не менее 10 ГБ

под данные PostgreSQL, не менее 3 ГБ под журналы транзакций и служебные журналы базы данных, не менее 5 ГБ под данные Kafka, не менее 10 ГБ под прикрепляемые файлы в MinIO и не менее 2 ГБ под журналы серверных приложений. Для предотвращения снижения производительности рекомендуется поддерживать запас свободного дискового пространства не менее 15 – 20 % от общего объема.

Перечисленные нефункциональные требования определяют качественные характеристики разрабатываемой информационной системы и должны учитываться при проектировании архитектуры, базы данных, серверной логики и пользовательского интерфейса.

1.5 Планирование разработки и оценка бюджета

Перед началом разработки информационной системы карьерного развития сотрудников был составлен календарный план работ. Его назначение заключается в том, чтобы определить состав основных этапов проекта, установить примерные сроки их выполнения, распределить трудовые ресурсы и получить предварительную оценку стоимости разработки. Такое планирование позволяет заранее увидеть наиболее трудоемкие части проекта и определить общую продолжительность выполнения выпускной квалификационной работы.

Для подготовки календарного плана использовалась программа Microsoft Project. В ней были заданы этапы проекта, их длительность, даты начала и завершения, связи между работами и назначенный ресурс. В качестве исполнителя всех этапов указан ресурс «Программист». При планировании учитывалось, что работы выполнялись не в формате полного рабочего дня, а в течение нескольких часов в день, поэтому продолжительность этапов отражает плановые трудозатраты с учетом частичной ежедневной занятости. На основании этих данных была сформирована диаграмма Ганта, отражающая порядок выполнения работ и временные интервалы каждого этапа [14]. Для построения диаграммы Ганта был заполнен список задач, представленный на рисунке 8.

	?	Название задачи	Длительность	Начало	Окончание	Предшественники	Затраты	Названия ресурсов
1		Проанализировать предметную область	1,88 дней?	Чт 09.04.26	Чт 16.04.26		14 062,50 Р	Программист
2		Проанализировать функциональные требования	1 день	Чт 16.04.26	Пн 20.04.26	1	7 500,00 Р	Программист
3		Проанализировать нефункциональные требования	1 день?	Пн 20.04.26	Пт 24.04.26	2	7 500,00 Р	Программист
4		Спроектировать архитектуру системы	1 день	Пт 24.04.26	Вт 28.04.26	3	7 500,00 Р	Программист
5		Разработка ИС	9 дней	Вт 28.04.26	Ср 03.06.26	4	67 500,00 Р	Программист
6		Оформление пояснительный записки для ВКР	4 дней	Ср 03.06.26	Пт 19.06.26	5	30 000,00 Р	Программист

Рисунок 8 – Список задач

Затем был получен календарный график выполнения работ, который показан на рисунке 9.

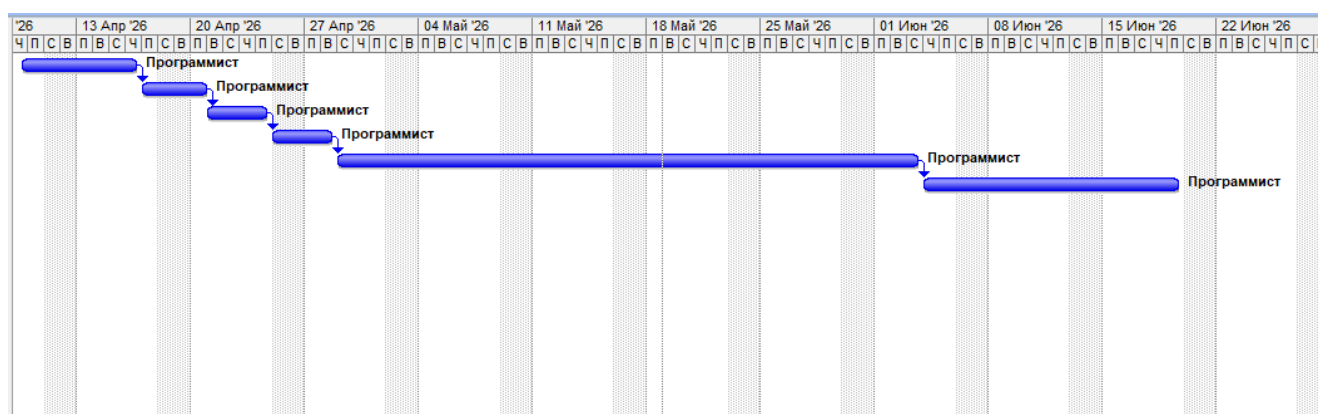


Рисунок 9 – Календарный график выполнения работ

В календарный план были включены работы, связанные как с исследовательской частью, так и с непосредственной разработкой программного продукта. К ним относятся анализ предметной области, определение функциональных и нефункциональных требований, проектирование архитектуры, разработка информационной системы и оформление пояснительной записки. Согласно плану, выполнение проекта начинается 9 апреля 2026 года, а завершение работ запланировано на 19 июня 2026 года.

Наиболее продолжительным этапом является разработка информационной системы. Это объясняется тем, что на данном этапе выполняется реализация основной функциональности проекта: серверной части, пользовательского интерфейса, структуры базы данных и авторизации. Вторым по продолжительности этап является оформление пояснительной записки, так как нужно подробно описать все этапы разработки и проектирования системы. Перед

этими этапами планируется анализ предметной области и проектирование системы суммарной продолжительностью 5 дней.

Для оценки бюджета была использована стоимость одного рабочего дня программиста, равная 7 500 рублей. Стоимость каждого этапа определялась путем умножения его продолжительности на дневную ставку исполнителя. Расчет затрат представлен в таблице 2.

Таблица 2 – Расчет затрат на разработку информационной системы

№	Этап работы	Длительность (рабочее время)	Затраты
1	Анализ предметной области	1,88 дней	14 062,50 руб.
2	Анализ функциональных требований	1 день	7 500,00 руб.
3	Анализ нефункциональных требований	1 день	7 500,00 руб.
4	Проектирование архитектуры системы	1 день	7 500,00 руб.
5	Разработка информационной системы	9 дней	67 500,00 руб.
6	Оформление пояснительной записки для ВКР	4 дня	30 000,00 руб.

По результатам расчета общая плановая продолжительность выполнения проекта составляет 17,88 рабочих дней, а общая стоимость разработки оценивается в 134 062,50 рубля.

Таким образом, составленный календарный план позволил определить структуру работ, оценить продолжительность разработки и рассчитать ориентировочный бюджет проекта. Полученные данные используются для обоснования трудоемкости создания информационной системы карьерного развития сотрудников и подтверждают необходимость предварительного планирования при выполнении выпускной квалификационной работы.

2 Проектирование системы

2.1 Проектирование архитектуры системы

В качестве архитектурного подхода была выбрана микросервисная архитектура. Микросервисная архитектура предполагает построение приложения как набора самостоятельных сервисов, каждый из которых отвечает за отдельную область системы и имеет собственные границы ответственности [15]. Для разрабатываемой информационной системы такой подход является целесообразным, поскольку процессы авторизации пользователей и процессы карьерного развития сотрудников относятся к разным функциональным областям. На рисунке 10 представлена диаграмма развертывания информационной системы карьерного развития сотрудников, отражающая архитектуру системы.

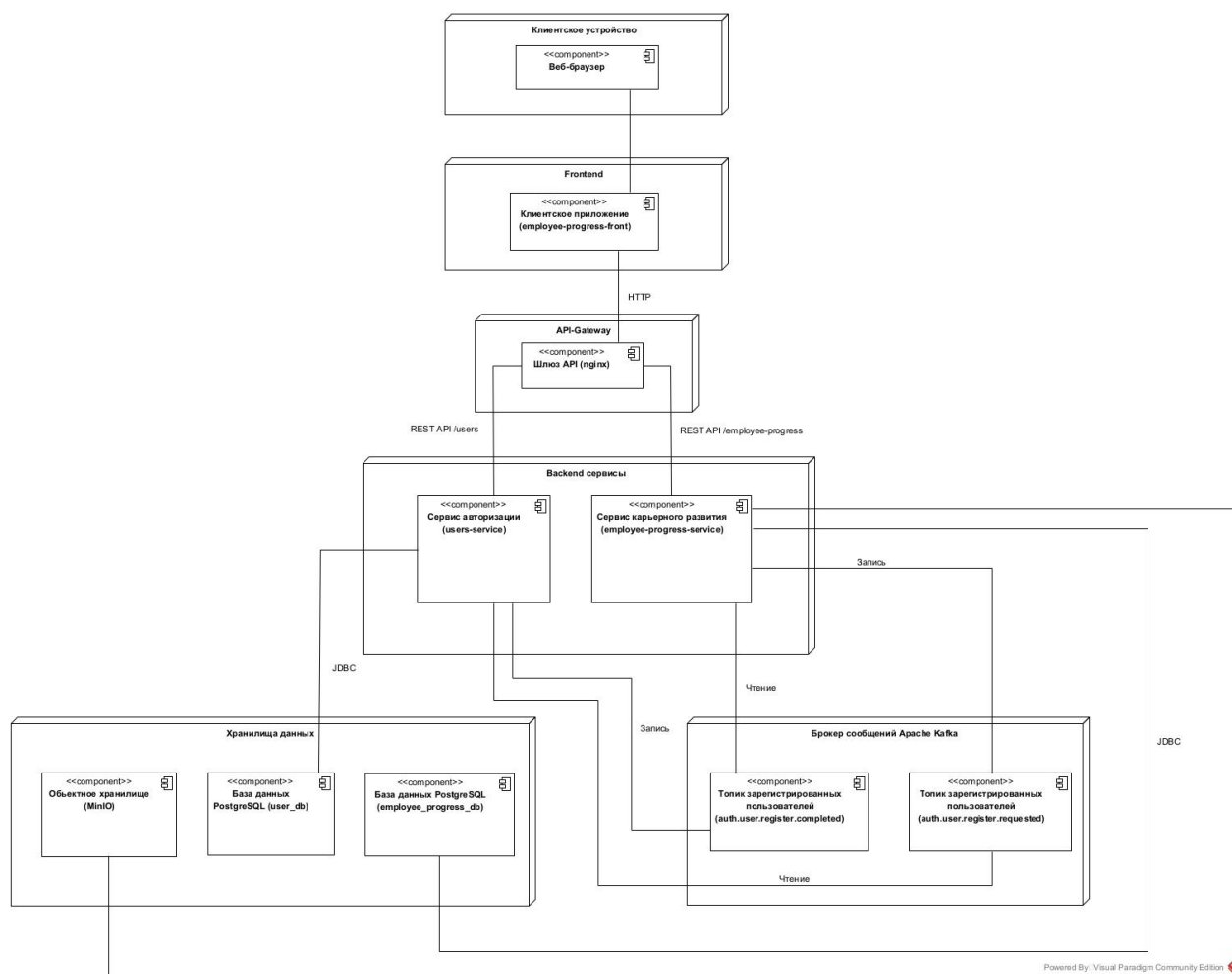


Рисунок 10 – Диаграмма развертывания

Пользователь взаимодействует с системой через веб-браузер. Клиентская часть представлена веб-приложением, разработанным на React. Через него пользователи получают доступ к разделам системы: сотрудникам, отделам, квалификационным уровням, индивидуальным планам развития, задачам, компетенциям, собеседованиям, кадровым решениям и отчетам.

Внешние запросы от клиентского приложения поступают на единый шлюз API Gateway, реализованный на базе Nginx [16]. Шлюз выполняет маршрутизацию запросов к соответствующим микросервисам. Запросы, связанные с учетными записями и авторизацией, направляются в users-service. Запросы, относящиеся к карьерному развитию сотрудников, передаются в employee-progress-service. Использование шлюза позволяет предоставить клиентскому приложению единую точку входа и скрыть внутреннюю структуру серверной части.

Серверная часть состоит из двух микросервисов. Микросервис users-service отвечает за учетные записи пользователей, аутентификацию, роли и выдачу JWT-токенов. Микросервис employee-progress-service реализует основную бизнес-логику системы. Каждый микросервис работает с собственной базой данных, что позволяет разделить их на обособленные архитектурные кванты. Для хранения файловых вложений, связанных с задачами индивидуальных планов развития, используется объектное хранилище MinIO. В нем размещаются прикрепленные пользователями файлы, а в базе данных сервиса карьерного развития сохраняются сведения о вложениях. Это позволяет отделить хранение файлов от основной реляционной модели данных.

Для асинхронного межсервисного взаимодействия используется Apache Kafka. Через брокер сообщений сервисы обмениваются событиями, связанными с регистрацией пользователей. На диаграмме представлены топики auth.user.register.requested и auth.user.register.completed. Первый используется для передачи события о необходимости регистрации пользователя, второй – для сообщения о завершении регистрации. Такой способ взаимодействия снижает

зависимость сервисов от прямых синхронных вызовов и делает обмен данными более гибким.

Таким образом, выбранная архитектура, за счет своего разделения на обособленные компоненты, обладает необходимыми свойствами, которые позволяют соблюдать разработанные нефункциональные требования.

2.2 Проектирование структур данных

2.2.1 Проектирование концептуальной модели базы данных

По результатам анализа предметной области и построения моделей бизнес-процессов была разработана концептуальная модель базы данных [17] информационной системы карьерного развития сотрудников.

Концептуальная модель базы данных отражает обобщенную структуру предметной области и позволяет определить состав ключевых сущностей, их основные атрибуты и взаимосвязи без учета особенностей физической реализации в конкретной системе управления базами данных. Данная модель служит основой для последующего перехода к логическому и физическому проектированию базы данных.

В разработанной концептуальной модели ключевую роль занимает сущность «Сотрудник». Также модель отображает сведения об организационной структуре компании, включая отделы и должности, а также элементы карьерной модели, представленные квалификационными уровнями, компетенциями и политиками карьерного продвижения. Такое построение модели позволяет отразить как структуру данных о сотрудниках, так и основные процессы их профессионального развития в рамках рассматриваемой предметной области.

Графическое представление концептуальной модели базы данных приведено на рисунке 11.

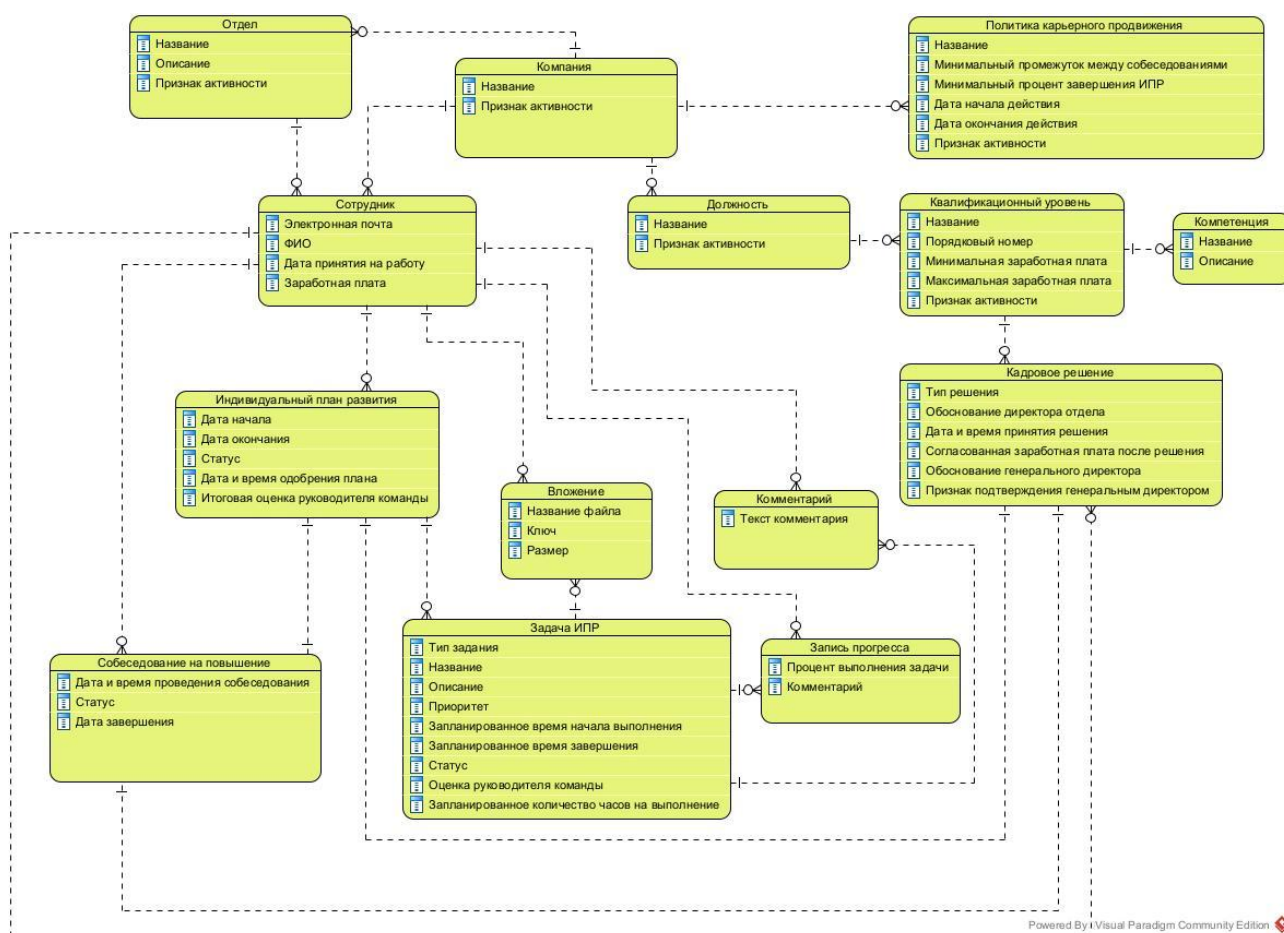


Рисунок 11 – Графическое представление концептуальной модели базы данных

На основе концептуальной модели базы данных было сформировано описание сущностей, представленное в таблице 3.

Таблица 3 – Сущности базы данных

Название сущности	Описание
Отдел	Содержит сведения о структурных подразделениях компании
Компания	Содержит сведения о компаниях, в рамках которых функционирует система
Политика карьерного продвижения	Содержит сведения о правилах и условиях карьерного продвижения сотрудников
Сотрудник	Содержит сведения о сотрудниках организации
Должность	Содержит сведения о должностях сотрудников
Квалификационный уровень	Содержит сведения о квалификационных уровнях сотрудников
Компетенция	Содержит сведения о профессиональных компетенциях сотрудников

Продолжение таблицы 3

Название сущности	Описание
Индивидуальный план развития	Содержит сведения об индивидуальных планах развития сотрудников
Вложение	Содержит сведения о файлах, прикрепленных к задачам индивидуального плана развития
Комментарий	Содержит сведения о комментариях, сопровождающих выполнение задач и оценку результатов
Собеседование на повышение	Содержит сведения о проведении собеседований по вопросу карьерного продвижения сотрудника
Задача ИПР	Содержит сведения о задачах, включенных в индивидуальный план развития
Запись прогресса	Содержит сведения о ходе выполнения задач индивидуального плана развития
Кадровое решение	Содержит сведения о результатах рассмотрения кандидатуры сотрудника и принятом кадровом решении

Концептуальная модель базы данных построена в среде Visual Paradigm 17.3 в нотации IDEF1X [18].

2.2.2 Проектирование логической модели базы данных

Логическая модель базы данных представляет собой развитие концептуальной модели и включает в себя сущности, их атрибуты, первичные и внешние ключи, а также взаимосвязи между ними, необходимые для формального описания структуры данных информационной системы.

На основании концептуальной модели была разработана логическая модель базы данных, отражающая состав данных, подлежащих хранению, и правила их взаимосвязи в рамках проектируемой информационной системы. Логическая модель базы данных представлена на рисунке 12.

В логической модели базы данных определены связи между сущностями, выделенными на этапе проектирования концептуальной модели. Так, сущность «Компания» связана с сущностями «Отдел», «Сотрудник», «Должность», «Квалификационный уровень» и «Политика карьерного продвижения», что отражает принадлежность организационной структуры, должностей, квалификационных уровней и правил карьерного продвижения к конкретной

компании. Сущность «Сотрудник» связана с сущностями «Индивидуальный план развития», «Задача ИПР», «Вложение», «Комментарий», «Запись прогресса», «Собеседование на повышение» и «Кадровое решение», поскольку сотрудник является центральным объектом системы.

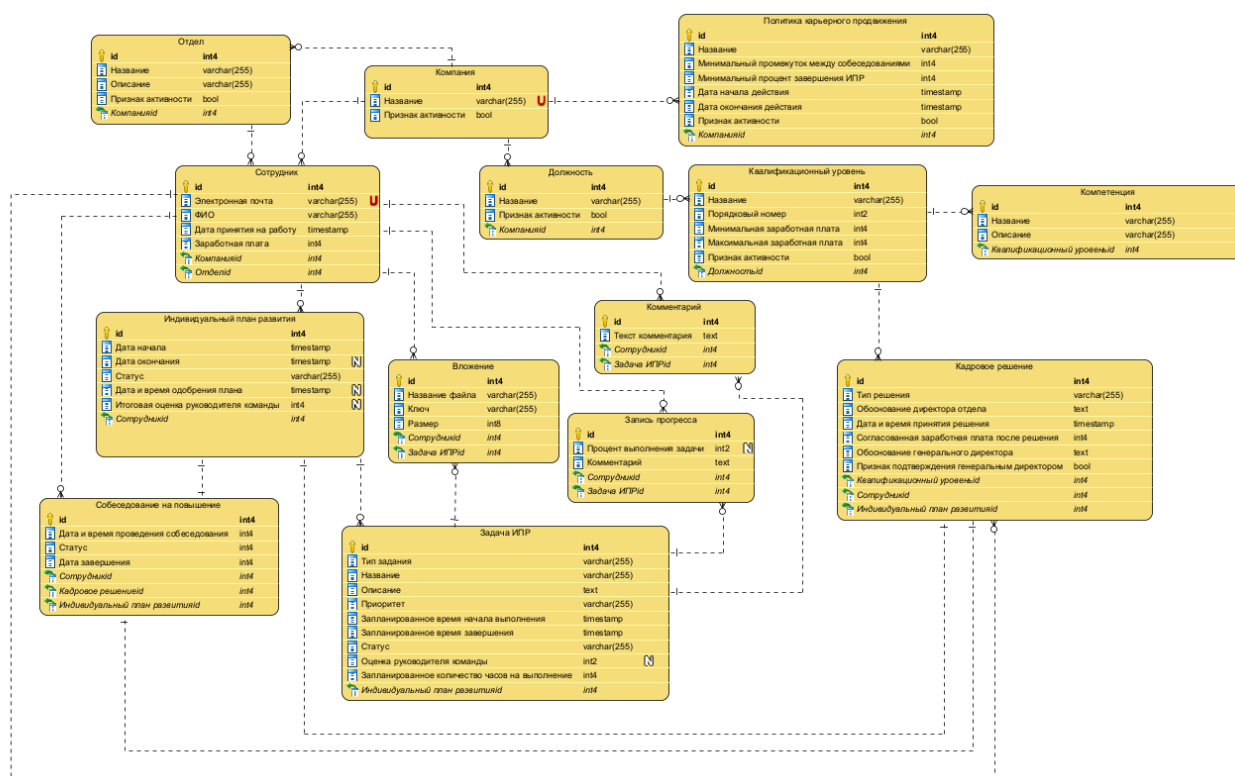


Рисунок 12 – Графическое представление концептуальной модели базы данных

Процесс формирования и сопровождения индивидуального плана развития представлен сущностями «Индивидуальный план развития», «Задача ИПР», «Вложение», «Комментарий» и «Запись прогресса». Указанные сущности позволяют хранить сведения о самом плане развития, входящих в него задачах, прикрепленных материалах, комментариях участников процесса и текущем уровне выполнения поставленных задач.

Контроль и оценка результатов развития сотрудника в логической модели представлены сущностями «Собеседование на повышение» и «Кадровое решение».

Следует отметить, что логическая модель содержит преимущественно связи типа «один-ко-многим». Например, одна компания может включать несколько отделов, сотрудников и должностей. Использование таких связей обеспечивает целостность данных и позволяет корректно отразить особенности рассматриваемой предметной области.

2.3 Проектирование серверной части

На основании результатов анализа предметной области, сформулированных функциональных требований и спроектированной структуры данных было выполнено проектирование программных средств информационной системы карьерного развития сотрудников. На данном этапе основное внимание уделялось определению состава программных компонентов, распределению ответственности между ними, а также описанию ключевых сценариев взаимодействия пользователя с системой. Архитектура разрабатываемого программного средства построена по многоуровневому принципу. В ее составе выделяются уровни представления, обработки запросов, бизнес-логики и хранения данных. Такой подход обеспечивает разделение ответственности между компонентами, упрощает сопровождение программного кода и создает основу для дальнейшего расширения функциональных возможностей системы. Взаимодействие между основными компонентами программного средства отражено на диаграмме классов, представленной на рисунке 13.

На диаграмме классов представлены ключевые элементы серверной части приложения: контроллеры, обработчики, сервисы и сущности предметной области. В частности, в ней отражены компоненты, обеспечивающие управление индивидуальными планами развития, квалификационными уровнями, политиками карьерного продвижения, итоговыми собеседованиями и кадровыми решениями.

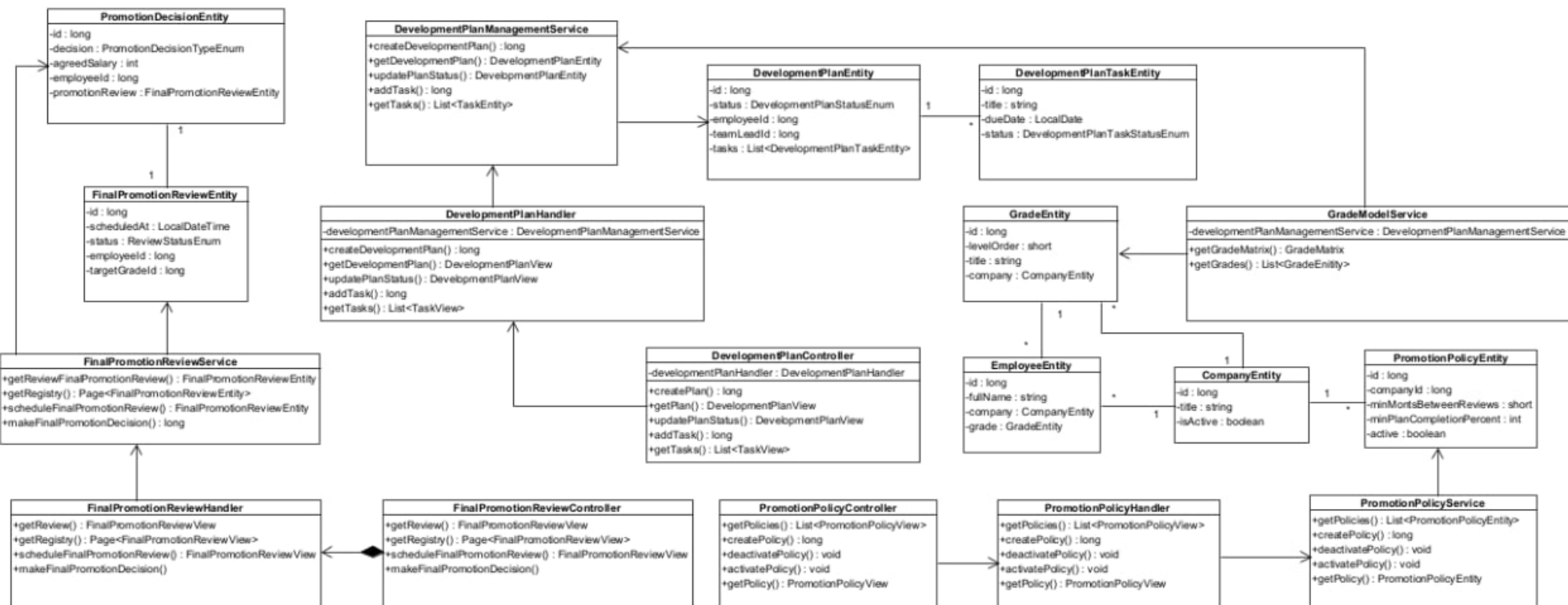


Рисунок 13 – Диаграмма классов основных компонентов программного средства

Для описания сценария создания квалификационного уровня была разработана диаграмма деятельности, представленная на рисунке 14. Данная диаграмма демонстрирует последовательность действий пользователя и реакцию системы на каждом этапе выполнения операции.

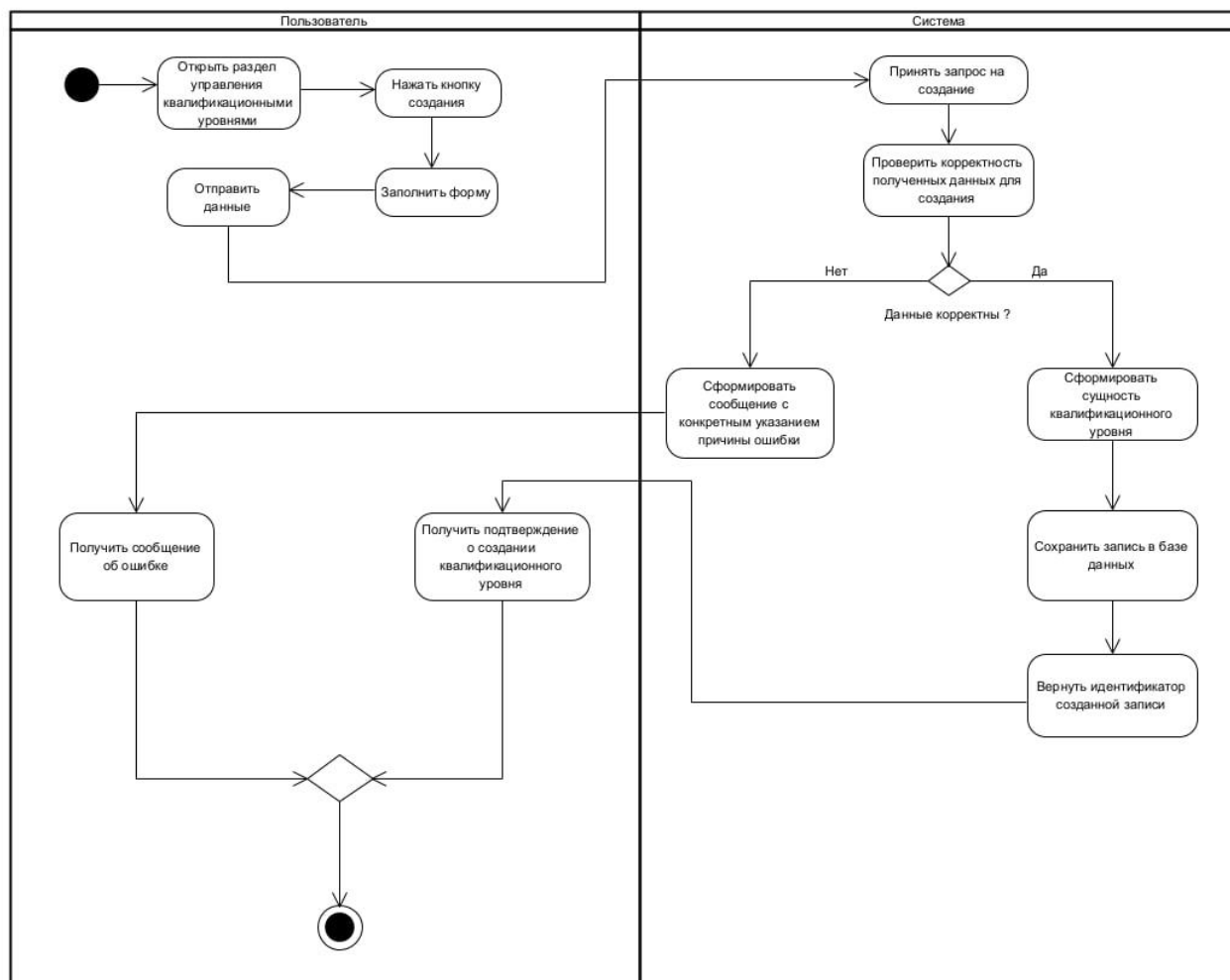


Рисунок 14 – Диаграмма деятельности для сценария «Создать квалификационный уровень»

Сценарий начинается с того, что пользователь инициирует появление формы создания, переходом в раздел управления квалификационными уровнями и нажатием соответствующей кнопки. Пользователь заполняет форму и отправляет данные. Система обрабатывает данные и проверяет их корректность. Если какое-то из полей не удовлетворяет условиям, то система формирует сообщение об ошибке и возвращает его пользователю. Если ошибки нет –

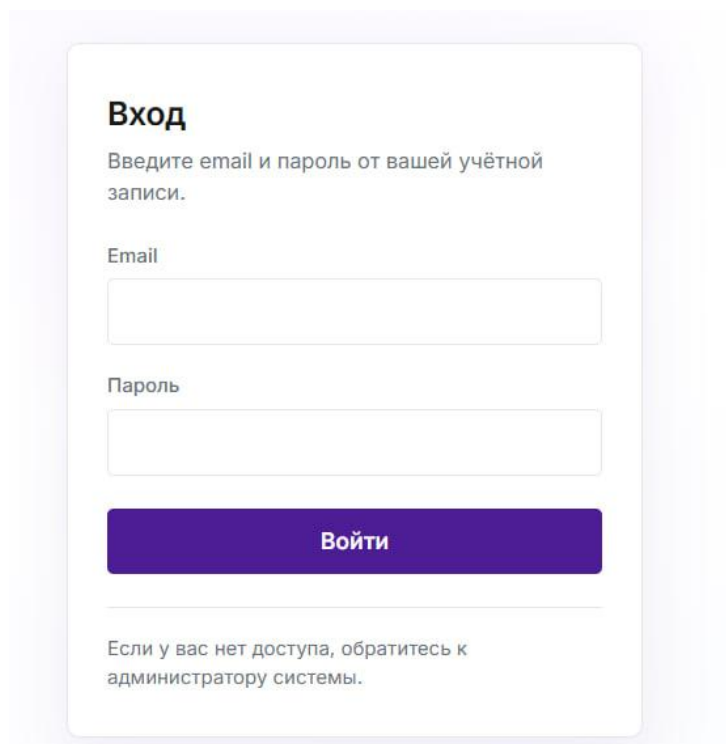
система формирует сущность из полученного запроса, сохраняет ее в базу данных и возвращает пользователю идентификатор.

Таким образом, в рамках данного раздела было выполнено проектирование программного средства, включающее определение структуры его основных компонентов и формализацию ключевых пользовательских сценариев.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс разрабатываемой информационной системы должен быть интуитивно понятным, удобным в освоении и обеспечивать выполнение всех разработанных вариантов использования. Для визуального проектирования эскизов интерфейса использовался инструмент Figma [19], позволяющий формировать прототипы экранов и уточнять состав элементов пользовательского взаимодействия.

При первом обращении к системе пользователь должен попадать на экран авторизации, представленный на рисунке 15. На данном экране размещены только основные элементы, необходимые для входа в систему: поля ввода электронной почты и пароля, кнопка подтверждения входа, а также краткое пояснение для пользователей, не имеющих доступа. Экран авторизации является важной частью пользовательского взаимодействия, поскольку именно с него начинается работа с системой. По этой причине при его проектировании были учтены требования к наглядности, простоте и доступности восприятия. Использование привычных для пользователя элементов управления и лаконичных текстовых пояснений позволяет сделать процесс входа понятным и предсказуемым. В результате разработанный экран авторизации обеспечивает удобный доступ к функциональным возможностям системы и формирует у пользователя целостное представление о ее структуре и принципах работы. Такое решение позволяет не перегружать стартовый экран лишними элементами и сосредоточить внимание пользователя на выполнении процедуры аутентификации.



Вход

Введите email и пароль от вашей учётной записи.

Email

Пароль

Войти

Если у вас нет доступа, обратитесь к администратору системы.

Рисунок 15 – Прототип экрана авторизации

После успешного входа пользователь переходит на основной экран приложения, представленный на рисунке 16. Данный экран выполняет роль рабочего стола и служит отправной точкой для дальнейшей работы с системой. На нем размещены основные разделы приложения, сгруппированные в виде навигационных карточек, а также верхнее меню, обеспечивающее быстрый переход между разделами.

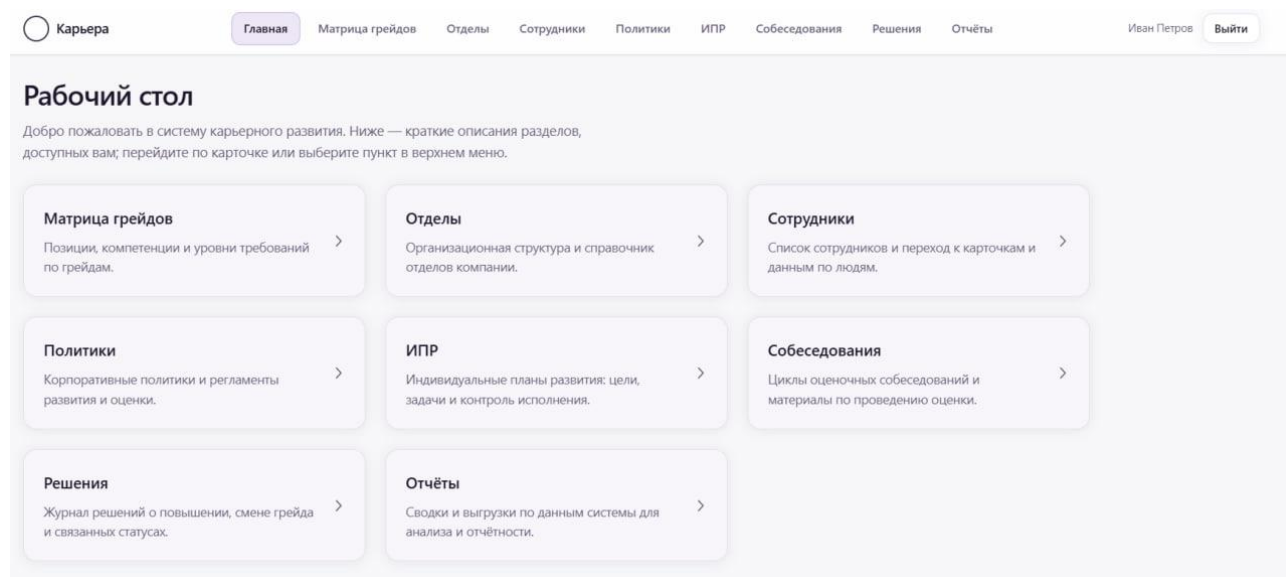


Рисунок 16 – Прототип главного экрана системы

Для выполнения операций создания и редактирования отдельных сущностей в системе предусмотрены модальные окна. Пример такого решения для создания новой должности представлен на рисунке 17. В модальном окне размещены поля для ввода кода, наименования и описания новой должности, а также кнопки сохранения и отмены действия. Использование модального окна позволяет выполнить ввод данных без перехода на отдельную страницу, тем самым сохранив контекст работы пользователя. Также использование модальных окон способствует упрощению разработки программного средства, так как является более простой альтернативой созданию новой страницы и связыванием ее с другими разделами.

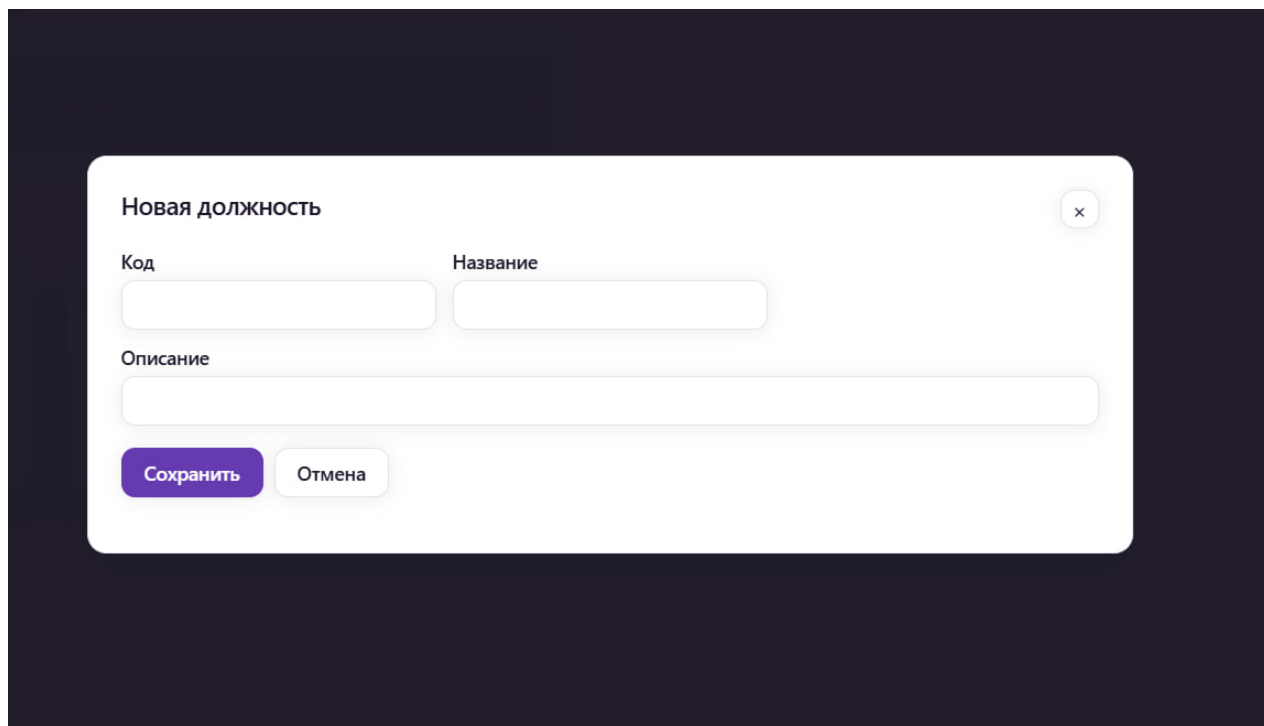
The image shows a dark-themed background with a white modal window in the center. The window has a title bar at the top left that says 'Новая должность' and a close button (an 'x' in a circle) at the top right. Inside the window, there are three input fields: 'Код' (Code) and 'Название' (Name) are side-by-side at the top, and 'Описание' (Description) is a larger field below them. At the bottom of the window, there are two buttons: a purple button labeled 'Сохранить' (Save) and a white button labeled 'Отмена' (Cancel).

Рисунок 17 – Прототип модального окна создания новой должности

Таким образом, проектирование пользовательского интерфейса было направлено на формирование понятной и удобной среды взаимодействия с системой. Это было достигнуто за счет разработки прототипов ключевых экранов, отражающих основные сценарии работы пользователя. Реализованные прототипы обеспечивают логичную последовательность действий, снижают сложность освоения системы и создают условия для эффективной работы пользователей с ее функциональными возможностями.

3 Реализация программных средств

3.1 Реализация слоя базы данных

Для реализации слоя базы данных была выбрана реляционная база данных PostgreSQL, механизм миграций Flyway, а также технологии Spring Data JPA и Hibernate, обеспечивающие структурированное хранение данных и доступ к ним из программного кода.

Структура базы данных формируется созданием версионизируемых SQL-миграций, которые располагаются в директории с кодовой базой проекта. Такой подход позволяет гарантировать воспроизводимость схемы на различных окружениях, отслеживать эволюцию структуры данных и последовательно вносить изменения в состав таблиц, ограничений и индексов.

Одной из ключевых таблиц системы является таблица `development_plans`, предназначенная для хранения сведений об индивидуальных планах развития сотрудников. Фрагмент SQL-кода, реализующего создание данной таблицы, приведен в листинге 1.

Листинг 1 – Миграция создания таблицы индивидуальных планов развития

```
create sequence if not exists development_plans_seq increment 1
minvalue 1 maxvalue 9223372036854775807 start 1 cache 1;

create table if not exists development_plans (
    id                bigint                not null,
    employee_id       bigint                not null,
    team_lead_id      bigint                null,
    period_start      date                  not null,
    period_end        date                  not null,
    status            development_plan_status not null,
    approved_at       timestamp with time zone null,
    created_at        timestamp with time zone not null default
now(),
    archived_at       timestamp                null,
    team_lead_plan_score_hundredths integer                null,
    target_grade_id   bigint                null,
    constraint development_plans_pk primary key (id),
    constraint development_plans_employees_fk foreign key
(employee_id) references employees (id),
    constraint development_plans_team_lead_fk foreign key
(team_lead_id) references employees (id),
```

```

        constraint development_plans_grades_fk foreign key
(target_grade_id) references grades (id)
);

comment on table development_plans is 'Индивидуальные планы
развития сотрудников';

comment on column development_plans.id is 'Идентификатор ИПР';
comment on column development_plans.employee_id is 'Идентификатор
сотрудника';
comment on column development_plans.team_lead_id is 'Идентификатор
тимлида';
comment on column development_plans.period_start is 'Дата начала
периода ИПР';
comment on column development_plans.period_end is 'Дата окончания
периода ИПР';
comment on column development_plans.status is 'Статус ИПР';
comment on column development_plans.approved_at is 'Дата и время
полного согласования ИПР';
comment on column development_plans.created_at is 'Дата и время
создания ИПР';
comment on column development_plans.archived_at is 'Дата и время
завершения ИПР';
comment on column development_plans.target_grade_id is 'Целевой
грейд ИПР';
comment on column
development_plans.team_lead_plan_score_hundredths is 'Итог по
оценкам тимлида по задачам ИПР (сотые долях балла';

```

Приведенный фрагмент показывает, что уже на уровне базы данных обеспечивается ссылочная целостность между связанными сущностями.

На уровне серверного приложения таблице `development_plans` соответствует сущность `DevelopmentPlanEntity`. Данная сущность реализует объектное представление записи из базы данных и описывает связи с другими сущностями системы. Фрагмент класса приведен в листинге 2.

Листинг 2 – Фрагмент сущности `DevelopmentPlanEntity`

```

@Entity
@Table(name = "development_plans")
@Comment("Индивидуальные планы развития")
@Getter
@Setter
public class DevelopmentPlanEntity {

    @SequenceGenerator(name = "development_plans_seq",
sequenceName = "development_plans_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator
= "development_plans_seq")
    @Id

```

```

@Column(name = "id", nullable = false)
@Comment("Логический идентификатор")
private Long id;

@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "employee_id", nullable = false)
@Comment("Сотрудник")
private EmployeeEntity employee;

@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "manager_id", nullable = false)
@Comment("Менеджер")
private EmployeeEntity manager;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "team_lead_id")
@Comment("Тимлид")
private EmployeeEntity teamLead;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "target_grade_id")
@Comment("Целевой грейд ИПР; компетенции берутся из
grade_competency_requirements выбранного грейда")
private GradeEntity targetGrade;

@Column(name = "period_start", nullable = false)
@Comment("Дата начала периода ИПР")
private LocalDate periodStart;

@Column(name = "period_end", nullable = false)
@Comment("Дата окончания периода ИПР")
private LocalDate periodEnd;

@Enumerated(EnumType.STRING)
@JdbcType(PostgreSQLEnumJdbcType.class)
@Column(name = "status", nullable = false, columnDefinition =
"development_plan_status")
@Comment("Статус ИПР")
private DevelopmentPlanStatusEnum status;

@Column(name = "approved_at")
@Comment("Дата и время полного согласования ИПР")
private LocalDateTime approvedAt;

@Column(name = "created_at", nullable = false)
@Comment("Дата и время создания")
private LocalDateTime createdAt;

@Column(name = "archived_at")
@Comment("Дата и время завершения (COMPLETED) или
закрытия/отмены (CLOSED)")
private LocalDateTime archivedAt;

```

```

@Column(name = "team_lead_plan_score_hundredths")
@Comment("Итог по оценкам тимлида по задачам ИПР в сотых долях
балла (10.00 = 1000), при завершении")
private Integer teamLeadPlanScoreHundredths;

@OneToMany(mappedBy = "plan", fetch = FetchType.LAZY)
@Comment("Строки компетенций ИПР")
private List<DevelopmentPlanCompetencyItemEntity>
competencyItems;
}

```

Использование сущности позволяет работать с данными базы не как с отдельными строками и столбцами, а как с объектами предметной области. Связи `@ManyToOne` обеспечивают возможность получить доступ от сущности индивидуального плана развития к сущностям сотрудника, руководителя команды и квалификационного уровня. За счет этого бизнес-логика может оперировать связанными объектами без необходимости вручную формировать SQL-запросы для каждого обращения к данным.

Отдельного пояснения требует поле `status`. В текущей реализации оно описывается перечислением `DevelopmentPlanStatusEnum`, а аннотация `@Enumerated(EnumType.STRING)` задает сохранение значения перечисления в строковом виде по имени его константы, а не по порядковому номеру. Это повышает надежность хранения данных, поскольку при изменении порядка элементов перечисления в исходном коде значения в базе данных не искажаются.

Для связей с сущностями используется режим загрузки `FetchType.LAZY`. Ленивый способ загрузки означает, что связанные объекты не извлекаются из базы данных одновременно с основной записью, а подгружаются только в тот момент, когда к ним происходит фактическое обращение. Такой подход снижает объем избыточно загружаемых данных и позволяет уменьшить нагрузку на базу данных при выборке списка объектов, так как не приходится соединять множество таблиц.

У использования ленивой загрузки есть недостаток в виде проблемы N+1. Эта проблема возникает, когда разработчик задействует в алгоритме поля сущности с ленивой загрузкой и для получения этих полей идет отдельный запрос в базу данных. Для решения этой проблемы используется аннотация

@EntityGraph. Она позволяет заранее указать, какие связанные сущности должны быть загружены вместе с основным объектом в рамках одного запроса.

Доступ к сущностям в системе реализован через репозитории Spring Data JPA. Существенной особенностью реализации слоя базы данных является использование механизма Specification [20] для построения динамических условий выборки. Данный механизм применяется при работе с реестрами записей, когда состав фильтров заранее не фиксирован и зависит от параметров запроса пользователя. В системе для этого используется класс DevelopmentPlanSpecificationBuilder.java, который последовательно добавляет условия к формируемой спецификации. Фрагмент кода приведен в листинге 3.

Листинг 3 – Фрагмент построения спецификации для фильтрации ИПР

```
public DevelopmentPlanSpecificationBuilder withCompanyIdEq(Long
companyId) {
    if (companyId != null) {
        spec =
spec.and(DevelopmentPlanSpecification.companyIdEq(companyId));
    }
    return this;
}
```

На практике формирование запроса с использованием спецификаций выполняется в сервисе управления индивидуальными планами развития. В методе getRegistry сначала создается объект строителя спецификации, затем к нему последовательно добавляются фильтры. После этого итоговая спецификация передается в метод findAll(specification, pageable), который выполняет выборку данных с постраничной выдачей результатов.

Применение спецификаций позволяет отказаться от создания большого количества односторонних методов репозитория для каждой комбинации параметров поиска. Вместо этого система формирует SQL-условия динамически, исходя из фактически переданных значений. Такой подход делает слой доступа к данным более гибким и способствует сохранению читаемости программного кода.

3.2 Реализация серверной части приложения

Реализация программных компонентов серверной части основана на многоуровневой архитектуре. В составе серверного приложения выделяются REST-контроллеры, обработчики запросов, сервисы бизнес-логики и репозитории доступа к данным, затронутые ранее. Такое разделение позволяет разграничить ответственность между компонентами, повысить читаемость кода и сопровождаемость системы.

Контроллеры в разработанном приложении отвечают за прием HTTP-запросов, извлечение параметров маршрута, запуск валидации входных данных и возврат результата клиенту. При этом контроллер не содержит прикладной логики, а передает выполнение следующему уровню системы, представленному обработчиками. Для примера можно рассмотреть получение индивидуального плана развития по идентификатору. Соответствующий метод контроллера приведен в листинге 4.

Листинг 4 – Фрагмент контроллера получения ИПР по идентификатору

```
@PreAuthorize(READ_DEVELOPMENT_PLAN)
@GetMapping("/{plan_id}")
public DevelopmentPlanView getPlan(
    @PathVariable("plan_id") Long planId
) {
    return developmentPlanHandler.getPlan(planId);
}
```

Из листинга видно, что контроллер выполняет только функции маршрутизации запроса и передачи управления обработчику DevelopmentPlanHandler. Дополнительно на уровне контроллера задается ограничение доступа с помощью аннотации @PreAuthorize, что позволяет допускать к операции только авторизованных пользователей с необходимыми полномочиями [21].

Следующим уровнем является handler. Его задача состоит в координации вызова сервисов, выполнении дополнительных проверок, а также преобразовании данных в конечный ответ. Пример метода обработчика приведен в листинге 5.

Листинг 5 – Фрагмент обработчика

```
public DevelopmentPlanView getPlan(Long planId) {
    return
    developmentPlanViewModelMapper.toDevelopmentPlanView(developmentPlanManagementService.getById(planId));
}
```

В данном случае обработчик сначала вызывает метод сервиса, получая данные, а затем преобразует их в объект представления, возвращаемый клиентскому приложению. Для операции чтения ИПР соответствующая логика реализована в `DevelopmentPlanManagementService`. Фрагмент метода приведен в листинге 6.

Листинг 6 – Фрагмент сервисного метода получения ИПР

```
public DevelopmentPlanEntity getById(Long id) {
    return developmentPlanRepository.findById(id)
        .orElseThrow(() -> new
    EntityNotFoundException(messageUtils.getMessage("error.not_found.development_plan", id)));
}
```

Как видно из приведенного кода, сервис сначала извлекает сущность по идентификатору. Если запись отсутствует, генерируется исключение с сообщением об ошибке. Сервис передает сущность обработчику, но клиентскому приложению сущность базы данных напрямую не передается, так как она может содержать данные, которые небезопасно показывать пользователю системы. Для возврата результата используется специальная record-модель `DevelopmentPlanView`, содержащая только те поля, которые необходимы внешнему потребителю. Ее формирование осуществляется с помощью mapper-компонента `DevelopmentPlanViewModelMapper`. Фрагмент преобразования приведен в листинге 7.

Листинг 7 – Фрагмент mapper-компонента преобразования сущности в представление

```
@Mapper(componentModel = "spring")
public interface DevelopmentPlanViewModelMapper {
    @Mapping(target = "employeeId", source = "plan.employee.id")
    @Mapping(target = "managerId", source = "plan.manager.id")
    @Mapping(target = "teamLeadId", source = "plan.teamLead.id")
    @Mapping(target = "targetGradeId", source =
    "plan.targetGrade.id")
}
```

```

    @Mapping(target = "teamLeadPlanScoreHundredths", source =
"plan.teamLeadPlanScoreHundredths")
    @Mapping(target = "tasks", source = "plan.tasks")
    @Mapping(target = "competencyItems", source =
"plan.competencyItems")
    @Mapping(target = "reviews", source = "plan.reviews")
    DevelopmentPlanView toDevelopmentPlanView(
        DevelopmentPlanEntity plan);
}

```

В методе `toDevelopmentPlanView()` задаются явные правила сопоставления полей: данные для конечного ответа извлекаются из соответствующих полей в сущности. Таким образом, преобразователь данных выполняет сборку итоговой структуры ответа, отделяя внутреннюю модель хранения данных от внешней модели представления, используемой при взаимодействии с клиентом.

3.3 Реализация слоя представления

Реализация пользовательского интерфейса осуществлялась с использованием языка JavaScript и технологии React. Для формирования единого визуального стиля переиспользуются общие компоненты, таблицы стилей и элементы оформления. Это позволило обеспечить единообразие пользовательского взаимодействия и удобство дальнейшего сопровождения программного кода.

После успешной авторизации пользователь получает доступ к основным разделам системы через верхнее навигационное меню.

Одним из ключевых разделов пользовательской части системы является страница матрицы квалификационных уровней, представленная на рисунке 18. Данный раздел предназначен для работы с перечнем должностей и связанными с ними квалификационными уровнями. На экране отображаются карточки должностей. Дополнительно предусмотрен переход к справочнику компетенций. Для пользователей, обладающих необходимыми правами, реализована функция управления матрицей квалификационных уровней. Таким образом, данный раздел обеспечивает наглядную работу с квалификационной структурой организации.

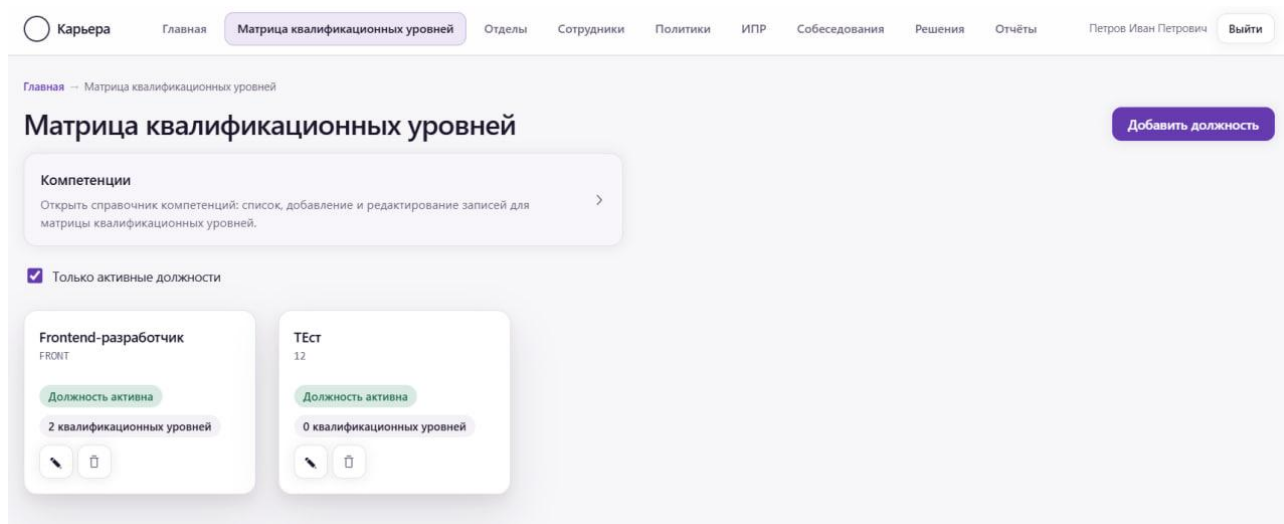


Рисунок 18 – Интерфейс раздела матрицы квалификационных уровней

Для работы с данными сотрудников реализован отдельный раздел, представленный на рисунке 19. Он предназначен для поиска и просмотра карточек сотрудников, работающих в компании. На странице размещено поле поиска, а также карточки сотрудников, содержащие сведения о ФИО, адресе электронной почты, принадлежности к отделу и текущем статусе активности. Выбор карточки позволяет перейти к подробному профилю сотрудника.

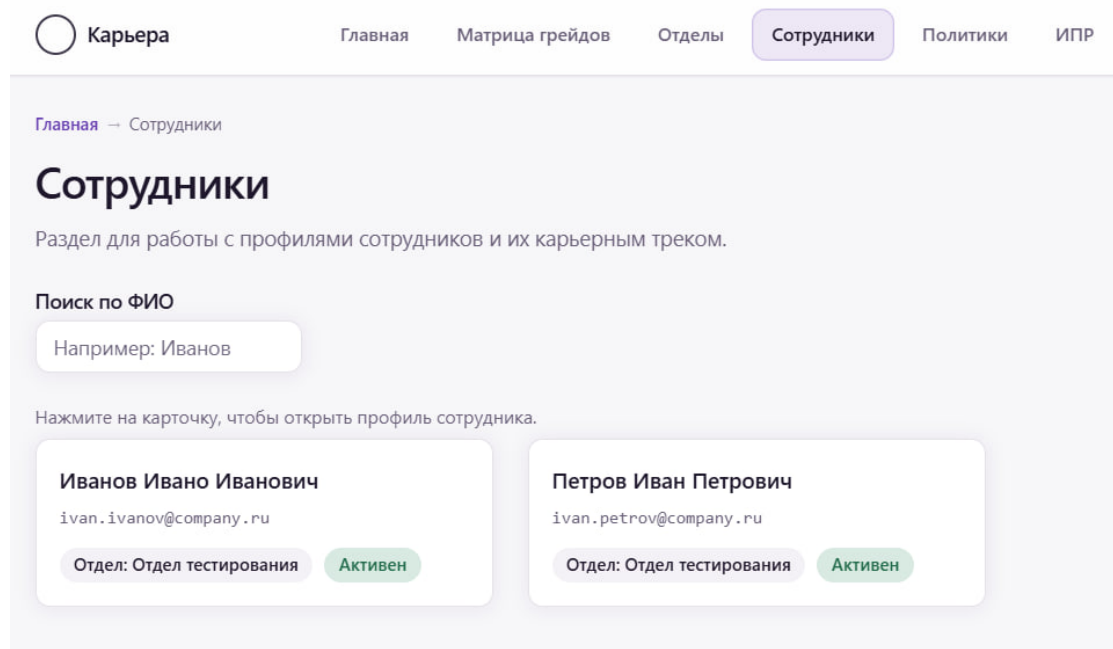


Рисунок 19 – Интерфейс раздела сотрудников

Для отображения организационной структуры компании предусмотрен раздел отделов, представленный на рисунке 20. На данном экране пользователь может просматривать список подразделений организации, сведения о количестве сотрудников в каждом отделе, а также признак активности подразделения. Дополнительно реализована возможность выделения отдела, к которому относится текущий пользователь.

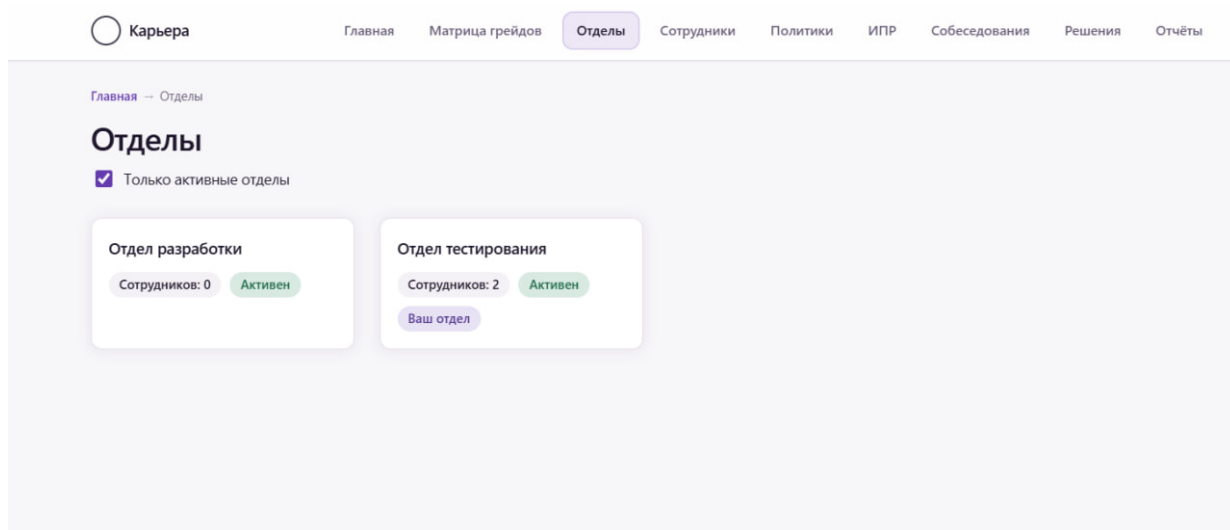


Рисунок 20 – Интерфейс раздела отделов

Таким образом, реализация пользовательского интерфейса была направлена на формирование удобной и логически организованной среды взаимодействия с системой. Использование общих компонентов, маршрутизации, модальных окон и карточного представления сущностей позволило обеспечить наглядность представления данных, удобство выполнения основных пользовательских операций и согласованность оформления.

В результате разработки удалось реализовать пользовательский интерфейс, обеспечивающий выполнение всех функциональных требований, заложенных на этапе проектирования системы. Кроме того, итоговая реализация интерфейса соответствует ранее разработанным прототипам, что позволило сохранить целостность первоначальной концепции, согласованность расположения элементов и единый визуальный стиль экранов.

4 Тестирование системы карьерного развития сотрудников

4.1 Обоснование выбора методов тестирования

Обеспечение качества разрабатываемой информационной системы является обязательной частью процесса ее создания, потому что сбои в системе подобного типа могут приносить тяжелые убытки, поскольку программное средство предназначено для автоматизации кадровых и управленческих процессов. В связи с этим для оценки корректности реализованного программного средства целесообразно применять совокупность взаимодополняющих режимов тестирования.

В рамках разрабатываемой системы используются тестирование методом черного ящика, модульное тестирование, а также анализ метрик и сложности программного кода. Применение тестирования методом черного ящика позволяет проверить соответствие внешнего поведения системы функциональным требованиям без учета внутренней реализации программных компонентов. Для формализации такой проверки используются классы эквивалентности, таблицы решений и функциональные диаграммы. Классы эквивалентности позволяют сократить число тестовых случаев за счет объединения однотипных входных данных, таблицы решений удобны при проверке сочетаний условий и статусов, а функциональные диаграммы обеспечивают наглядное представление проверяемых функций системы.

Модульное тестирование применяется для проверки отдельных компонентов серверной части приложения в изолированном виде. Его использование особенно важно для сервисов, в которых сосредоточена основная бизнес-логика, связанная с управлением индивидуальными планами развития, задачами, согласованием и расчетом оценок. Преимущество данного вида тестирования заключается в возможности быстро выявлять ошибки в отдельных методах и проверять корректность прикладных правил без необходимости запуска всей системы целиком.

Функциональное тестирование клиентской части направлено на проверку корректности реализации основных пользовательских сценариев работы с системой. Преимущество функционального тестирования заключается в том, что оно позволяет подтвердить работоспособность ключевых возможностей системы в условиях, максимально приближенных к реальному использованию, а также выявить ошибки, возникающие при последовательном выполнении пользовательских действий.

Дополнительно выполняется анализ метрик и сложности кода. Его применение позволяет получить количественную оценку характеристик исходного кода, связанных с объемом, структурной сложностью и трудоемкостью сопровождения. Преимущество данного подхода заключается в возможности выявления наиболее сложных участков программной реализации, которые требуют повышенного внимания при дальнейшем развитии и сопровождении системы.

4.2 Тестирование методом черного ящика

В качестве объекта тестирования методом черного ящика рассматривается функция добавления задачи в индивидуальный план развития. Для проверки выбранной функции целесообразно использовать классы эквивалентности, функциональную диаграмму и таблицу решений.

4.2.1 Классы эквивалентности

Для проведения тестирования выделено поле «Плановая трудоемкость задачи в часах». В рамках проверки принимается, что корректное значение данного поля должно быть целым положительным числом в диапазоне от 1 до 250 часов включительно. Значения, выходящие за установленные границы, рассматриваются как некорректные. Такой выбор позволяет проверить как корректную обработку допустимых значений, так и реакцию системы на граничные и ошибочные варианты ввода. Классы эквивалентности представлены в таблице 4.

Таблица 4 – Классы эквивалентности для поля «Плановая трудоемкость задачи в часах»

Название поля	Правильные КЭ	Неправильные КЭ
Плановая трудоемкость задачи в часах	1 – 250(1)	$-\infty - 0(2)$ $250 - \infty(3)$

Графическое представление классов эквивалентности изображено на рисунке 21.



Рисунок 21 – Класс эквивалентности (длина плановой трудоемкости)

В таблице 5 представлены представлены тест-кейсы для поля «Длина плановой трудоемкости».

Таблица 5 – Тест-кейсы для поля «Плановая трудоемкость задачи в часах»

Номер теста	Классы эквивалентност и	Входные данные	Выходные данные
1	1	40	Успешное сохранение значения
2	1	1	Успешное сохранение граничного минимального значения
3	1	250	Успешное сохранение граничного максимального значения
4	2	0	Сообщение об ошибке о недопустимом значении трудоемкости
5	2	-5	Сообщение об ошибке о недопустимом значении трудоемкости
6	3	251	Сообщение об ошибке о превышении допустимого значения

Таким образом, использование классов эквивалентности позволяет сократить число тестовых случаев и при этом охватить основные группы корректных и некорректных входных данных.

4.2.2 Функциональная диаграмма и таблица решений

Для функции добавления задачи в ИПР были выделены основные причины и следствия, определяющие результат выполнения операции. В таблице 6 представлены условия, влияющие на успешность добавления задачи, и соответствующие реакции системы.

Таблица 6 – Набор причин

Идентификатор причины	Причина
П1	Индивидуальный план развития существует
П2	Пользователь имеет разрешения на просмотр и редактирование ИПР
П3	Статус ИПР допускает добавление задачи
П4	Наименование задачи заполнено корректно
П5	Плановая трудоемкость задачи находится в диапазоне от 1 до 250 часов
П6	Дата начала задачи находится в пределах периода ИПР

В таблице 7 представлены реакции системы, соответствующие причинам их возникновения.

Таблица 7 – Результаты выполнения операций

Идентификатор следствия	Следствие
С1	Сообщение «ИПР не найден»
С2	Сообщение «Доступ к ИПР запрещен»
С3	Сообщение «Статус ИПР не допускает добавление задачи»
С4	Сообщение «Некорректное название задачи»
С5	Сообщение «Некорректная плановая трудоемкость задачи»
С6	Сообщение «Дата начала задачи выходит за пределы периода ИПР»
С7	Сообщение «Задача успешно добавлена»

В соответствии с выделенными причинами и следствиями была разработана функциональная диаграмма, которая представлена на рисунке 22. Функциональная диаграмма показывает связь между проверками входных данных и полученным результатом. Это позволяет выявить комбинации причин и следствий, отражающие реакцию системы на действия пользователя.

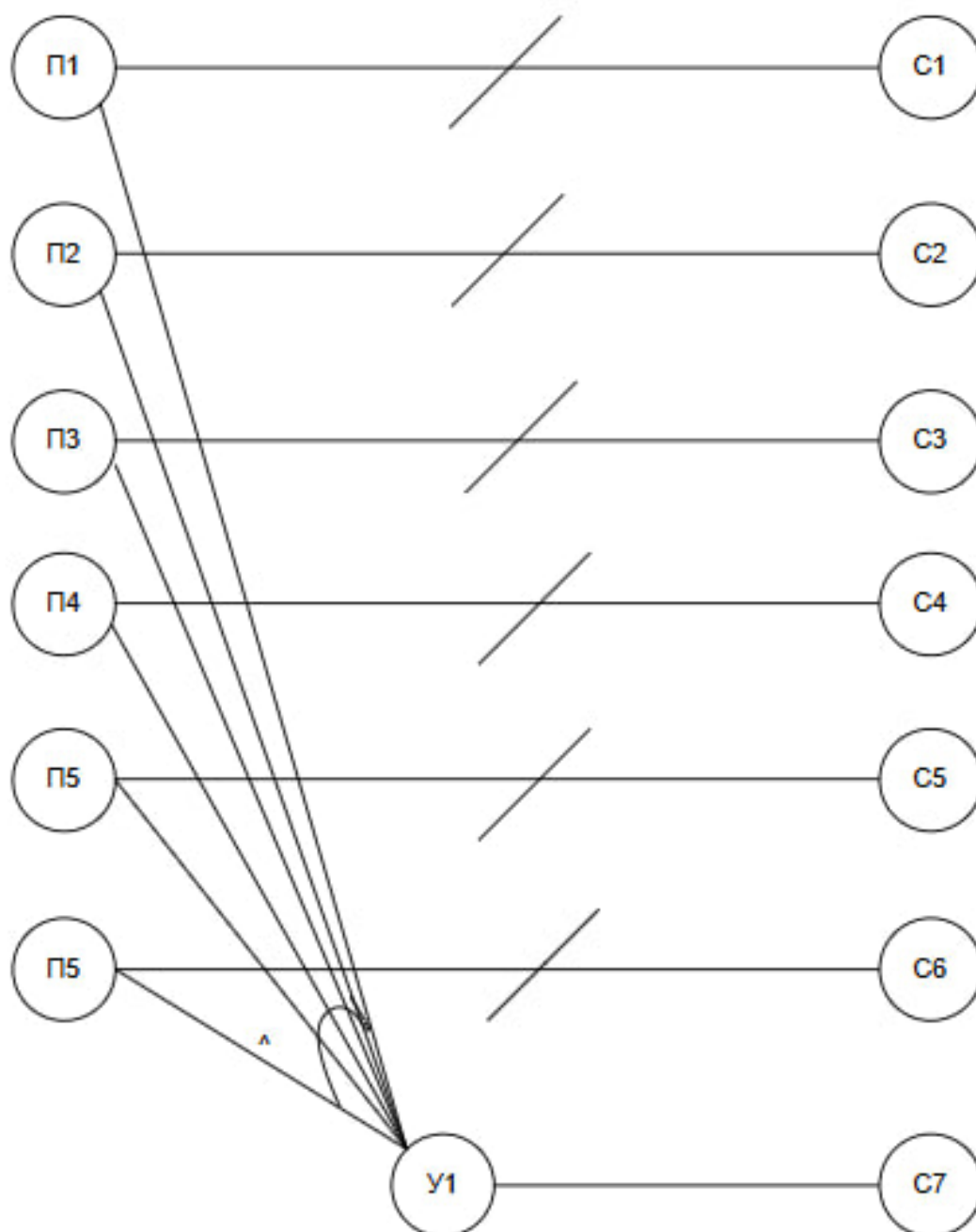


Рисунок 22 – Диаграмма функциональности

Составлены различные комбинации причин и соответствующих им следствий, представленные в таблице 8. Такой подход позволяет формализовать возможные варианты поведения системы и определить набор тестовых сценариев для последующей проверки.

Таблица 8 – Таблица решений для функции добавления задачи в ИПР

	1	2	3	4	5	6	7
П1	0	1	1	1	1	1	1

Продолжение таблицы 8

	1	2	3	4	5	6	7
П2	0	0	1	1	1	1	1
П3	0	0	0	1	1	1	1
П4	0	0	0	0	1	1	1
П5	0	0	0	0	0	1	1
П6	0	0	0	0	0	0	1
С1	1	0	0	0	0	0	0
С2	0	1	0	0	0	0	0
С3	0	0	1	0	0	0	0
С4	0	0	0	1	0	0	0
С5	0	0	0	0	1	0	0
С6	0	0	0	0	0	1	0
С7	0	0	0	0	0	0	1

С помощью таблицы решений были составлены тест-кейсы для проверки варианты поведения системы во время добавления задачи индивидуального плана развития. Тест-кейсы и ожидаемые результаты представлены в таблице 9.

Таблица 9 – Тест-кейсы для создания задачи ИПР

Номер тест-кейса	Входные данные	Ожидаемый результат
1	ИПР: отсутствует, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 40 часов	ИПР не найден
2	ИПР: существует, Доступ: отсутствует, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 40 часов	Доступ к ИПР запрещен
3	ИПР: существует, Доступ: разрешен, Статус ИПР: Завершен, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 40 часов	Статус ИПР не допускает добавление задачи
4	ИПР: существует, Доступ: разрешен, Статус ИПР: Черновик, Наименование задачи: пустое значение, Плановая трудоемкость: 40 часов	Некорректно заполнено наименование задачи
5	ИПР: существует, Доступ: разрешен, Статус ИПР: Черновик, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 0 часов	Некорректная плановая трудоемкость задачи
6	ИПР: существует, Доступ: разрешен, Статус ИПР: Черновик, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 251 час	Некорректная плановая трудоемкость задачи
7	ИПР: существует, Доступ: разрешен, Статус ИПР: Черновик, Наименование задачи: Изучение Spring Boot, Плановая трудоемкость: 40 часов	Задача успешно добавлена

Таким образом были сформированы входные данные и соответствующие им результаты для создания задачи ИПР.

4.3 Расчет метрик качества программного кода

Метрики позволяют оценить сложность системы, определить стиль написания и контролировать качество программного кода.

Для метода сохранения прогресса по задаче индивидуального плана развития был произведен расчет метрик SLOC и Холстеда. Код этого метода приведен в листинге 8.

Листинг 8 – Метод сохранения прогресса задачи

```
public DevelopmentPlanTaskProgressUpdateView
recordTaskProgress(Long planId,
Long taskId,
DevelopmentPlanTaskProgressUpdateRequest request) {
    Integer progressPercent =
request.progressPercent();
    validateProgressPercent(progressPercent);

    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    validatePlanReadyForExecution(planId);
    DevelopmentPlanTaskEntity task =
validateTaskBelongsToPlan(taskId, planId);
    validateEmployeeActive(plan.getEmployee());

    DevelopmentPlanTaskStatusEnum currentTaskStatus =
resolveTaskStatusByProgress(progressPercent);
    task.setStatus(currentTaskStatus);

    developmentPlanTaskService.save(task);

    TaskProgressEntryEntity entry = new TaskProgressEntryEntity();
    entry.setTask(task);
    entry.setAuthorEmployee(plan.getEmployee());
    entry.setEntryType(TaskProgressEntryTypeEnum.PROGRESS);
    entry.setProgressPercent(progressPercent);
    entry.setComment(trimOptional(request.comment()));
    entry.setArtifactUrl(null);
    entry.setCreatedAt(LocalDateTime.now());

    TaskProgressEntryEntity savedEntry =
taskProgressEntryService.save(entry);
    return developmentPlanTaskExecutionViewMapper
        .toDevelopmentPlanTaskProgressUpdateView(savedEntry,
currentTaskStatus);
}
```

Расчет количественной метрики SLOC для метода сохранения прогресса представлен в таблице 10.

Таблица 10 – Расчет метрики SLOC

Метрика	Результат
Физические строки кода	28
Исходные строки кода	23
Строки комментариев	0
Пустые строки	5

Результат расчета метрики показывает, что рассматриваемый метод имеет умеренный объем и остается удобным для анализа и сопровождения.

Обычно расчета количественной метрики недостаточно для принятия решения о сложности, поэтому дополнительно используется метрика Холстеда [22]. Для ее расчета необходимо выделить уникальные операнды и операторы функции, а также подсчитать их вхождения. Операторы функции представлены в таблице 11.

Таблица 11 – Операторы функции recordTaskProgress

Оператор	Число вхождений
=	6
return	1
new	1
.	16
validateProgressPercent	1
getPlanOrThrow	1
validatePlanReadyForExecution	1
validateTaskBelongsToPlan	1
validateEmployeeActive	1
resolveTaskStatusByProgress	1
setStatus	1
save	2

Продолжение таблицы 11

Оператор	Число вхождений
setTask	1
setAuthorEmployee	1
setEntryType	1
setProgressPercent	1
setComment	1
trimOptional	1
comment	1
setArtifactUrl	1
setCreatedAt	1
now	1
toDevelopmentPlanTaskProgressUpdateView	1

Количество операторов (n1) составило 23, а их вхождения (N1) – 44.
Операнды функции recordTaskProgress приведены в таблице 12.

Таблица 12 – Операнды функции recordTaskProgress

Операнд	Число вхождений
planId	4
taskId	3
request	3
progressPercent	4
plan	3
task	4
currentTaskStatus	3
entry	8
savedEntry	2

Продолжение таблицы 12

Операнд	Число вхождений
developmentPlanTaskService	1
taskProgressEntryService	1
developmentPlanTaskExecutionViewMapper	1
TaskProgressEntryTypeEnum	1
PROGRESS	1
LocalDateTime	1
null	1

Количество операнд (n_2) составило 16, а их вхождения (N_2) – 41.

Произведем расчет метрики Холстеда для сохранения пргресса (смотри листинг 8).

Длина кода программы рассчитывается по формуле:

$$N=N_1+N_2=44+41=85,$$

где N – длина программы;

N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

Словарь программы рассчитывается по формуле:

$$n=n_1+n_2=23+16=39,$$

где n – словарь программы;

n_1 – число уникальных операторов в программе;

n_2 – число уникальных операндов в программе.

Теоретическая длина программы рассчитывается по формуле:

$$N'=n_1 \times \log_2(n_1) + n_2 \times \log_2(n_2) = 23 \times \log_2(23) + 16 \times \log_2(16) = 168,04,$$

где N' – теоретическая длина программы;

n_1 – число уникальных операторов в программе;

n_2 – число уникальных операндов в программе.

Объем программы по метрике Холстеда рассчитывается по формуле:

$$V=N \times \log_2(n) = 85 \times \log_2(39) = 449,26,$$

где V – объем программы;
 N – длина программы;
 n – словарь программы.

Теоретический объем программы рассчитывается по формуле:

$$V' = N' \times \log_2(n) = 168,04 \times \log_2(39) = 888,17,$$

где V' – теоритический объем программы;
 N' – длина программы;
 n – словарь программы.

Абсолютный уровень качества программирования рассчитывается по формуле:

$$L = V' \div V = 888,17 \div 449,26 = 1,98,$$

где L – абсолютный уровень качества программирования;
 V' – теоретический объем программы;
 V – объем программы.

Относительный уровень качества программирования рассчитывается по формуле:

$$L' = (2 \times n_2) \div (n_1 \times N_2) = (2 \times 16) \div (23 \times 41) = 0,0339,$$

где L – относительный уровень качества программирования;
 n_2 – число уникальных операндов программы;
 n_1 – число уникальных операторов программы;
 N_2 – общее число операндов в программе;

Дополнительно рассчитан показатель сопровождаемости Maintainability Index (MI) по формуле:

$$\begin{aligned} MI &= 171 - 5.2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LoC) = \\ &= 171 - 5.2 \times \ln(449.26) - 0.23 - 16.2 \times \ln(23) = 88.2, \end{aligned}$$

где HV – вычислительная сложность;
 CC – структурная сложность кода;
 LoC – количество строк исходного кода.

По расчетам абсолютный уровень качества программирования имеет высокое значение, этот результат является следствием высокой лаконичности

кода. Относительный уровень качества низкий, так как в методе одновременно содержится низкое количество уникальных операндов и относительно большое количество операторов.

Рассчитанный индекс сопровождаемости соответствует высокой оценке.

Таким образом, рассчитанные метрики указывают на высокое качество написанного программного кода.

4.4 Анализ метрик программного кода

Для измерения качества программного кода были выбраны MOOD-метрики. Они позволяют проверить следующие качества кода:

- инкапсуляция;
- полиморфизм;
- наследование;
- связность.

Полученные MOOD-метрики представлены на рисунке 23.

Project metrics						
project ▾	АНФ	АИФ	CF	МНФ	МИФ	РФ
project	86,26%	1,09%	2,67%	33,91%	0,49%	52,63%

Рисунок 23 – MOOD-метрики программного кода

Значение АНФ равно 86,26 % – это высокий уровень сокрытия атрибутов классов. Значение МНФ равно 33,91 % – это означает, что большая часть методов имеют модификатор доступа private. Значение CF, обозначающее связанность классов, равно 2,67 % – показывают высокую модульность. Значение РФ

равно 52,63 % – показывает умеренное использование полиморфизма в программном коде.

Также были рассчитаны метрики сложности методов. К числу их показателей относятся когнитивная сложность, цикломатическая сложность, существенная сложность и сложность проектирования. Полученные значения данных метрик представлены на рисунке 24.

Method	CogC	ev(G)	iv(G)	v(G)
Irm.employeeprogressservice.service.DevelopmentPlanTaskExecutionManagementService.getCommentEn	2	2	3	3
Irm.employeeprogressservice.service.DevelopmentPlanTaskExecutionManagementService.deleteTaskComment(Long, Long, Long)	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskExecutionManagementService.addTaskComm	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.saveAll(Collection<Develo	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.save(DevelopmentPlan Tas	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.getByld(Long)	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.findByTaskIdAndObjectKe	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.findByld(Long)	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.findAllByTaskIdOrderByCr	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.findAll()	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.existsByld(Long)	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.deleteByld(Long)	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentService.count()	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.validateTaskE	1	2	2	2
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.validatePlanR	1	2	2	2
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.validateObjec	1	2	2	2
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.validateFileNa	1	2	2	2
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.trimOptional(	2	2	1	3
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.getTaskAttac	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.getTaskAttac	0	1	1	1
Irm.employeeprogressservice.service.DevelopmentPlanTaskAttachmentManagementService.getPlanOrThru	0	1	1	1

Рисунок 24 – Метрики сложности методов программного средства

Анализ показал, что для большинства методов характерны низкие значения показателей сложности. Значения CogC в основном находятся в диапазоне от 0 до 2, что свидетельствует о невысокой когнитивной сложности и удобстве восприятия программного кода. Значения v(G) преимущественно составляют от 1 до 3, что указывает на небольшое число независимых путей выполнения и отсутствие избыточно разветвленной логики. Показатели ev(G) и iv(G) также остаются на низком уровне, что характеризует методы как достаточно простые по структуре и удобные для сопровождения. Полученные

результаты позволяют сделать вывод о том, что реализованные программные компоненты обладают приемлемым уровнем сложности и не содержат чрезмерно усложненных алгоритмических конструкций.

Результаты анализа полученных метрик показывают, что программный код разработанной системы имеет качественную структуру.

4.5 Модульное тестирование

Было проведено тестирование серверной части приложения с помощью модульного тестирования. Для тестирования были использованы технологии JUnit и Mockito. Mockito используется для имитации зависимостей между классами. Mock-объект заменяет реальный класс и его поведение для проверки конкретного сценария. Это позволяет не воссоздавать полное окружение, заменив необязательные зависимости.

Пример программного кода класса модульных тестов для работы с отделами приведен в приложении В. Полный код всех модульных тестов расположен на сервисе GitHub [23].

Всего было разработано и успешно выполнено 232 модульных теста. Результаты представлены на рисунке 25.

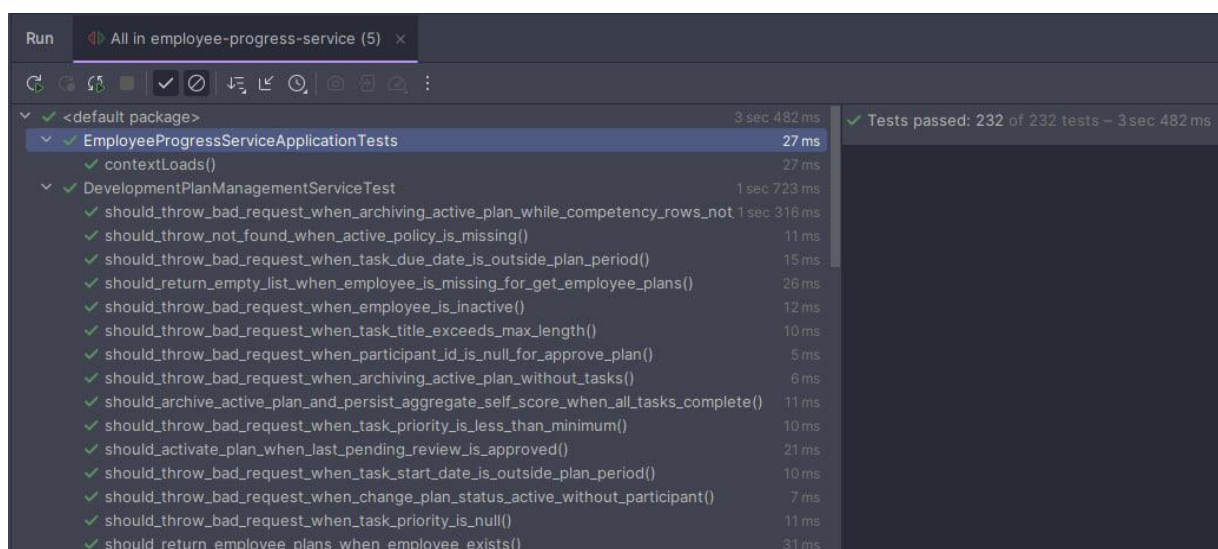


Рисунок 25 – Результаты модульного тестирования

Было успешно проведено модульное тестирование, что гарантирует корректное поведение и устойчивость системы к регрессу.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована и разработана информационная система карьерного развития и оценки сотрудников. Целью работы являлось создание программного средства, обеспечивающего управление квалификационными уровнями, индивидуальными планами развития, политиками карьерного продвижения, процедурами оценки сотрудников и кадровыми решениями в ИТ-компании. Поставленная цель была достигнута.

В рамках работы был проведен анализ предметной области. А также обзор существующих программных продуктов-аналогов, были выявлены их функциональные возможности и ограничения с точки зрения решаемой задачи. На основании проведенного анализа были сформулированы функциональные и нефункциональные требования к разрабатываемой информационной системе.

Выполнено проектирование программного средства. Были разработаны модели бизнес-процессов, концептуальная и логическая модели базы данных, определены основные сущности предметной области и их взаимосвязи. Также были спроектированы программные компоненты системы, структура пользовательского интерфейса и ключевые сценарии взаимодействия пользователей с приложением.

Осуществлена реализация программного продукта. Серверная часть системы разработана на языке Java с использованием фреймворка Spring Boot, а клиентская часть реализована на языке JavaScript с применением технологии React. В результате были реализованы основные функциональные возможности системы.

Проведено тестирование разработанной системы. Были применены тестирование методом черного ящика, модульное и функциональное тестирование, а также анализ метрик и сложности программного кода. Полученные результаты показали корректность реализации основных функций

системы, приемлемый уровень качества программного кода и возможность дальнейшего сопровождения и развития программного средства.

Таким образом, в результате выполнения выпускной квалификационной работы была разработана информационная система карьерного развития сотрудников. Практическая значимость работы заключается в возможности использования разработанной системы в ИТ-компаниях для повышения прозрачности карьерных процессов, упрощения контроля выполнения индивидуальных планов развития и улучшения качества управленческих решений.

Список использованных источников

1. Зимняя, И. А. Ключевые компетентности как результативно-целевая основа компетентного подхода в образовании / И. А. Зимняя. – М. : Исследовательский центр проблем качества подготовки специалистов, 2004. – 38 с.
2. Натейкина Ю. О. Особенности формирования индивидуального плана развития в организации // Международный научно-исследовательский журнал. 2015. № 2 [Электронный ресурс]. URL: <https://research-journal.org/archive/2-33-2015-february/osobennosti-formirovaniya-individualnogo-plana-razvitiya-v-organizacii> (дата обращения: 01.05.2026).
3. Временные методические рекомендации по разработке отраслевой рамки квалификаций на основе Национальной рамки квалификаций Российской Федерации // Информационно-правовой портал [Электронный ресурс]. URL: <https://www.garant.ru/products/ipo/prime/doc/70035136/> (дата обращения: 03.05.2026).
4. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Якобсон. – М. : ДМК Пресс, 2007. – 496 с.
5. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс, Д. Битти. – СПб. : БХВ-Петербург, 2014. – 736 с.
6. Уоллс, К. Spring Boot в действии / К. Уоллс. – М. : ДМК Пресс, 2017. – 352 с.
7. Обе, Р. PostgreSQL: основы и практика применения / Р. Обе, Л. Хсу. – 3-е изд. – М. : ДМК Пресс, 2018. – 312 с.
8. Flyway Documentation. Database migrations [Электронный ресурс]. URL: <https://documentation.red-gate.com/flyway/flyway-concepts/migrations> (дата обращения: 14.05.2026).
9. React. The library for web and native user interfaces [Электронный ресурс]. URL: <https://react.dev/> (дата обращения: 15.05.2026).
10. Филдинг Р. Т. Архитектурные стили и проектирование программных архитектур, основанных на сетевом взаимодействии // Калифорнийский

университет в Ирвайне [Электронный ресурс]. URL: <https://roy.gbiv.com/pubs/dissertation/> (дата обращения: 17.05.2026).

11. Джонс М., Брэдли Дж., Сакимура Н. JSON Web Token (JWT) // IETF. 2015 [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/rfc7519/> (дата обращения: 19.05.2026).

12. MinIO Documentation. Object Storage Documentation [Электронный ресурс]. URL: <https://docs.min.io/> (дата обращения: 19.05.2026).

13. Apache Kafka Documentation [Электронный ресурс]. URL: <https://kafka.apache.org/documentation/> (дата обращения: 19.05.2026).

14. Льюис, С. Microsoft Project 2019. Шаг за шагом / С. Льюис, К. Четфилд, Т. Джонсон ; пер. с англ. Д. А. Беликова. – М. : ДМК Пресс, 2021. – 570 с.

15. Ньюмен, С. Создание микросервисов / С. Ньюмен. – 2-е изд. – СПб. : Питер, 2023. – 624 с.

16. Де Йонге, Д. NGINX. Книга рецептов / Д. Де Йонге ; пер. с англ. – М. : ДМК Пресс, 2020. – 178 с.

17. Дадян, Э. Г. Современные базы данных. Основы. Часть 1 / Э. Г. Дадян. – М. : ИНФРА-М, 2017. – 88 с.

18. National Institute of Standards and Technology. Federal Information Processing Standards Publication. Integration Definition for Information Modeling (IDEF1X) [Электронный ресурс]. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub184.pdf> (дата обращения: 22.05.2026).

19. Figma. Free Prototyping Tool [Электронный ресурс]. URL: <https://www.figma.com/prototyping/> (дата обращения: 23.05.2026).

20. Spring Documentation. Specifications [Электронный ресурс]. URL: <https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html> (дата обращения: 25.05.2026).

21. Spring Documentation. Method Security [Электронный ресурс]. URL: <https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html> (дата обращения: 25.05.2026).
22. Холстед, М. Х. Начала науки о программах / М. Х. Холстед ; пер. с англ. – М. : Финансы и статистика, 1981. – 128 с.
23. Кареев, А.А. Информационная система карьерного развития и оценки сотрудников ИТ-компании // GitHub [Электронный ресурс]. URL: <https://github.com/andrysheq/employee-progress> (дата обращения: 15.05.2026).

Приложение А
(справочное)
Глоссарий

API Gateway – единая точка входа для внешних запросов к набору внутренних сервисов. В разработанной системе используется для маршрутизации запросов клиентского приложения к backend-компонентам.

JWT (JSON Web Token) – компактный формат токена для передачи утверждений между сторонами в защищенном виде. В системе применяется для аутентификации пользователей и разграничения прав доступа.

REST API – архитектурный подход к организации взаимодействия между компонентами системы по протоколу HTTP. В работе используется для обмена данными между клиентской и серверной частями приложения.

Spring Data JPA – модуль платформы Spring для организации доступа к данным с использованием JPA. В работе применяется для реализации репозиторий и взаимодействия с базой данных.

Specification – механизм формирования динамических условий выборки данных в Spring Data JPA. В системе используется для построения гибких фильтров при работе с реестрами записей.

ИПР (индивидуальный план развития) – документ или сущность системы, содержащая цели, задачи, сроки и результаты развития сотрудника. В разработанной системе ИПР является одним из центральных объектов предметной области.

Кадровое решение – результат рассмотрения кандидатуры сотрудника по итогам оценки и собеседования на повышение. В системе фиксирует итог управленческой процедуры карьерного продвижения.

Квалификационный уровень – характеристика профессионального уровня сотрудника, определяющая требования к компетенциям, ответственности и ожидаемым результатам работы. В системе используется при построении карьерной модели и управлении развитием сотрудников.

Компетенция – совокупность знаний, навыков и умений, необходимых сотруднику для выполнения профессиональной деятельности на определенном квалификационном уровне.

Приложение Б
(обязательное)
Листинг программы

Листинг Б.1 – Файл DevelopmentPlanManagementService.java

```
package lrm.employeeprogressservice.service;

import lombok.RequiredArgsConstructor;
import
lrm.employeeprogressservice.mapper.developmentplan.DevelopmentPlan
  ViewMapper;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanEntity;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanReviewEntity;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanTaskAttachmen
  tEntity;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanTaskEntity;
import lrm.employeeprogressservice.model.dao.EmployeeEntity;
import lrm.employeeprogressservice.model.dao.GradeEntity;
import
lrm.employeeprogressservice.model.dao.TaskProgressEntryEntity;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanCreateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanRegistryRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanTaskCreateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanTaskStatusUpdateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanReviewView;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanStatusUpdateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanTaskUpdateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanCompetencyItemView;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanTaskView;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
  lanUpdateRequest;
```

```

import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanView;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanReviewStatu
sEnum;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanReviewerRol
eEnum;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanStatusEnum;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanTaskPriorit
yEnum;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanTaskStatusE
num;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanTaskTypeEnu
m;
import
lrm.employeeprogressservice.service.specification.DevelopmentPlans
pecificationBuilder;
import lrm.employeeprogressservice.utils.MessageUtils;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.server.ResponseStatusException;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@Service
@Transactional(readOnly = true)
@RequiredArgsConstructor
public class DevelopmentPlanManagementService {

    private static final DevelopmentPlanStatusEnum
PLAN_STATUS_DRAFT = DevelopmentPlanStatusEnum.DRAFT;
    private static final DevelopmentPlanStatusEnum
PLAN_STATUS_ACTIVE = DevelopmentPlanStatusEnum.ACTIVE;
    private static final DevelopmentPlanStatusEnum
PLAN_STATUS_COMPLETED = DevelopmentPlanStatusEnum.COMPLETED;
    private static final DevelopmentPlanStatusEnum
PLAN_STATUS_CLOSED = DevelopmentPlanStatusEnum.CLOSED;

    private static final DevelopmentPlanTaskStatusEnum
TASK_STATUS_PLANNED = DevelopmentPlanTaskStatusEnum.PLANNED;

```

```

        private static final DevelopmentPlanReviewStatusEnum
REVIEW_STATUS_PENDING = DevelopmentPlanReviewStatusEnum.PENDING;
        private static final DevelopmentPlanReviewStatusEnum
REVIEW_STATUS_APPROVED = DevelopmentPlanReviewStatusEnum.APPROVED;

        private static final DevelopmentPlanReviewerRoleEnum
REVIEW_ROLE_EMPLOYEE = DevelopmentPlanReviewerRoleEnum.EMPLOYEE;
        private static final DevelopmentPlanReviewerRoleEnum
REVIEW_ROLE_MANAGER = DevelopmentPlanReviewerRoleEnum.MANAGER;
        private static final DevelopmentPlanReviewerRoleEnum
REVIEW_ROLE_TEAM_LEAD = DevelopmentPlanReviewerRoleEnum.TEAM_LEAD;

        private final DevelopmentPlanService developmentPlanService;
        private final DevelopmentPlanTaskService
developmentPlanTaskService;
        private final DevelopmentPlanReviewService
developmentPlanReviewService;
        private final TaskProgressEntryService
taskProgressEntryService;
        private final DevelopmentPlanTaskAttachmentService
developmentPlanTaskAttachmentService;
        private final EmployeeService employeeService;
        private final PromotionPolicyService promotionPolicyService;
        private final GradeService gradeService;
        private final DevelopmentPlanViewMapper
developmentPlanViewMapper;
        private final DevelopmentPlanTeamLeadScoreAggregationService
developmentPlanTeamLeadScoreAggregationService;
        private final DevelopmentPlanCompetencyManagementService
developmentPlanCompetencyManagementService;

        private final MessageUtils messageUtils;

        public DevelopmentPlanView getPlan(Long planId) {
            DevelopmentPlanEntity plan = getPlanOrThrow(planId);
            return toPlanView(plan);
        }

        public List<DevelopmentPlanView> getEmployeePlans(Long
employeeId) {
            return
developmentPlanService.findAllByEmployeeIdOrderByPeriodStartDescId
Desc(employeeId).stream()
                .map(this::toPlanView)
                .toList();
        }

        public Page<DevelopmentPlanView>
getRegistry(DevelopmentPlanRegistryRequest request, Pageable
pageable) {
            if (request.periodStartFrom() != null
                && request.periodEndTo() != null

```

```

        &&
        request.periodEndTo().isBefore(request.periodStartFrom())) {
            throw new
                ResponseStatusException(HttpStatus.BAD_REQUEST,
                    messageUtils.getMessage("error.reporting.date_range_invalid"));
        }

        DevelopmentPlanSpecificationBuilder specificationBuilder =
            new DevelopmentPlanSpecificationBuilder()
                .withCompanyIdEq(request.companyId())
                .withEmployeeIdEq(request.employeeId())
                .withManagerIdEq(request.managerId())
                .withTeamLeadIdEq(request.teamLeadId())
                .withTargetGradeIdEq(request.targetGradeId())
                .withStatusEq(request.status())
                .withPeriodStartFrom(request.periodStartFrom())
                .withPeriodEndTo(request.periodEndTo())

            .withEmployeeTitleLike(request.employeeTitleLike());

        return
            developmentPlanService.findAll(specificationBuilder.build(),
                pageable)
                .map(this::toPlanView);
    }

    public List<DevelopmentPlanTaskView> getPlanTasks(Long planId)
    {
        getPlanOrThrow(planId);
        return
            developmentPlanTaskService.findAllByPlanIdOrderByDueDateAscIdAsc(p
                lanId).stream()

            .map(developmentPlanViewMapper::toDevelopmentPlanTaskView)
                .toList();
    }

    @Transactional
    public Long createPlan(Long employeeId,
        DevelopmentPlanCreateRequest request) {
        validateCreatePlanRequest(request);

        EmployeeEntity employee = getEmployeeOrThrow(employeeId);
        validateEmployeeActive(employee);

        EmployeeEntity teamLead =
            resolveTeamLeadForPlanCreate(employee, request.teamLeadId());
        EmployeeEntity manager = resolveManager(employee,
            request.managerId());
        ensureActivePolicyExists(employee.getCompany().getId(),
            request.periodStart());
    }

```

```

        GradeEntity targetGrade =
resolveTargetGradeForPlanCreate(request.targetGradeId(),
employee);

        DevelopmentPlanEntity plan = new DevelopmentPlanEntity();
        plan.setEmployee(employee);
        plan.setManager(manager);
        plan.setTeamLead(teamLead);
        plan.setTargetGrade(targetGrade);
        plan.setPeriodStart(request.periodStart());
        plan.setPeriodEnd(request.periodEnd());
        plan.setStatus(PLAN_STATUS_DRAFT);
        plan.setApprovedAt(null);
        plan.setCreatedAt(LocalDateTime.now());
        DevelopmentPlanEntity savedPlan =
developmentPlanService.save(plan);

        List<DevelopmentPlanTaskEntity> tasksToSave =
request.tasks().stream()
                .map(taskRequest -> buildTaskEntity(savedPlan,
taskRequest))
                .toList();
        developmentPlanTaskService.saveAll(tasksToSave);

        List<DevelopmentPlanReviewEntity> reviewsToSave = new
ArrayList<>();
        reviewsToSave.add(buildReviewEntity(savedPlan, employee,
REVIEW_ROLE_EMPLOYEE));
        reviewsToSave.add(buildReviewEntity(savedPlan, manager,
REVIEW_ROLE_MANAGER));
        reviewsToSave.add(buildReviewEntity(savedPlan, teamLead,
REVIEW_ROLE_TEAM_LEAD));
        developmentPlanReviewService.saveAll(reviewsToSave);

developmentPlanCompetencyManagementService.seedCompetencyItemsFrom
PlanTargetGrade(savedPlan);

        return savedPlan.getId();
    }

    @Transactional
    public DevelopmentPlanView updatePlan(Long planId,
DevelopmentPlanUpdateRequest request) {
        validatePlanPeriod(request.periodStart(),
request.periodEnd());

        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        validatePlanIsDraft(plan);

        EmployeeEntity employee = plan.getEmployee();
        EmployeeEntity manager = resolveManager(employee,
request.managerId());

```

```

        EmployeeEntity teamLead = resolveTeamLead(employee,
request.teamLeadId());

        validateAllTaskDatesFitPeriod(planId,
request.periodStart(), request.periodEnd());

        plan.setManager(manager);
        plan.setTeamLead(teamLead);
        plan.setPeriodStart(request.periodStart());
        plan.setPeriodEnd(request.periodEnd());

        return toPlanView(developmentPlanService.save(plan));
    }

    @Transactional
    public DevelopmentPlanView changePlanStatus(Long planId,
DevelopmentPlanStatusUpdateRequest request) {
        DevelopmentPlanStatusEnum target = request.status();
        if (PLAN_STATUS_ACTIVE.equals(target)) {
            if (request.participantEmployeeId() == null) {
                throw new ResponseStatusException(
                    HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.status.participant
_required_for_active")
                );
            }
            return approvePlan(planId,
request.participantEmployeeId());
        }
        if (PLAN_STATUS_COMPLETED.equals(target)) {
            if (request.participantEmployeeId() != null) {
                throw new ResponseStatusException(
                    HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.status.participant
_forbidden_for_terminal")
                );
            }
            completePlan(planId);
            return getPlan(planId);
        }
        if (PLAN_STATUS_CLOSED.equals(target)) {
            if (request.participantEmployeeId() != null) {
                throw new ResponseStatusException(
                    HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.status.participant
_forbidden_for_terminal")
                );
            }
            closePlan(planId);
            return getPlan(planId);
        }
    }

```

```

    }
    if (PLAN_STATUS_DRAFT.equals(target)) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.status.transition_
to_draft_not_supported")
        );
    }
    throw new ResponseStatusException(
        HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.status.unsupported
_target_status")
    );
}

@Transactional
public void completePlan(Long planId) {
    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    if (PLAN_STATUS_COMPLETED.equals(plan.getStatus())) {
        return;
    }
    if (PLAN_STATUS_CLOSED.equals(plan.getStatus())) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.complete_status_in
valid")
        );
    }

    if (PLAN_STATUS_ACTIVE.equals(plan.getStatus())) {
        List<DevelopmentPlanTaskEntity> tasks =
developmentPlanTaskService.findAllByPlanIdOrderByDueDateAscIdAsc(p
lanId);
        if (tasks.isEmpty()) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.complete_active_pl
an_has_no_tasks")
            );
        }
        for (DevelopmentPlanTaskEntity task : tasks) {
            if
(!DevelopmentPlanTaskStatusEnum.DONE.equals(task.getStatus())) {
                throw new ResponseStatusException(
                    HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.complete_not_all_t
asks_done")
                );
            }
        }
    }
}

```

```

        }
        if (task.getTeamLeadTaskScore() == null) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.complete_task_team
_lead_score_missing", task.getId())
            );
        }
    }
    if
    (developmentPlanCompetencyManagementService.hasUnapprovedCompetenc
yRows(planId)) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.complete_competenc
ies_not_tl_approved")
        );
    }

plan.setTeamLeadPlanScoreHundredths(developmentPlanTeamLeadScoreAg
gregationService.computeTeamLeadPlanScoreHundredths(tasks));
    }

    plan.setStatus(PLAN_STATUS_COMPLETED);
    if (plan.getArchivedAt() == null) {
        plan.setArchivedAt(LocalDate.now());
    }
    developmentPlanService.save(plan);
}

@Transactional
public void closePlan(Long planId) {
    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    if (PLAN_STATUS_CLOSED.equals(plan.getStatus())) {
        return;
    }
    if (PLAN_STATUS_COMPLETED.equals(plan.getStatus())) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.close_status_inval
id")
        );
    }
    if (!PLAN_STATUS_DRAFT.equals(plan.getStatus()) &&
!PLAN_STATUS_ACTIVE.equals(plan.getStatus())) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.close_status_inval
id")

```

```

        );
    }
    plan.setStatus(PLAN_STATUS_CLOSED);
    if (plan.getArchivedAt() == null) {
        plan.setArchivedAt(LocalDateTime.now());
    }
    developmentPlanService.save(plan);
}

private boolean isTerminalPlanStatus(DevelopmentPlanStatusEnum
status) {
    return PLAN_STATUS_COMPLETED.equals(status) ||
PLAN_STATUS_CLOSED.equals(status);
}

@Transactional
public DevelopmentPlanTaskView createTask(Long planId,
DevelopmentPlanTaskCreateRequest request,
Long
teamLeadEmployeeId) {
    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    validatePlanAllowsTaskCreate(plan, teamLeadEmployeeId);

    DevelopmentPlanTaskEntity task = buildTaskEntity(plan,
request);
    return
developmentPlanViewMapper.toDevelopmentPlanTaskView(developmentPla
nTaskService.save(task));
}

@Transactional
public DevelopmentPlanTaskView updateTask(Long planId, Long
taskId, DevelopmentPlanTaskUpdateRequest request) {
    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    validatePlanIsDraft(plan);
    DevelopmentPlanTaskEntity task =
getTaskAndValidateBelongsToPlan(taskId, planId);

    applyTaskUpdate(task, plan, request);
    return
developmentPlanViewMapper.toDevelopmentPlanTaskView(developmentPla
nTaskService.save(task));
}

@Transactional
public void deleteTask(Long planId, Long taskId) {
    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    validatePlanIsDraft(plan);
    DevelopmentPlanTaskEntity task =
getTaskAndValidateBelongsToPlan(taskId, planId);

```

```

        for (TaskProgressEntryEntity progressEntry :
taskProgressEntryService.findAllByTaskId(task.getId())) {

taskProgressEntryService.deleteById(progressEntry.getId());
        }
        for (DevelopmentPlanTaskAttachmentEntity attachment :
developmentPlanTaskAttachmentService.findAllByTaskIdOrderByCreated
AtDescIdDesc(task.getId())) {

developmentPlanTaskAttachmentService.deleteById(attachment.getId()
);
        }
        developmentPlanTaskService.deleteById(task.getId());
    }

    @Transactional
    public DevelopmentPlanView approvePlan(Long planId, Long
participantEmployeeId) {
        validateApprovePlanRequest(participantEmployeeId);

        DevelopmentPlanEntity plan = getPlanOrThrow(planId);

        finalizePlanActivationIfReady(plan);
        if (isPlanFullyApproved(plan)) {
            return toPlanView(plan);
        }
        validatePlanAllowsApproval(plan);

        List<DevelopmentPlanReviewEntity> participantReviews =
developmentPlanReviewService
            .findAllByPlanIdAndReviewerId(planId,
participantEmployeeId);
        validateParticipantReviews(participantReviews);

        LocalDateTime now = LocalDateTime.now();
        List<DevelopmentPlanReviewEntity> reviewsToSave =
participantReviews.stream()
            .filter(review ->
!REVIEW_STATUS_APPROVED.equals(review.getStatus()))
            .peek(review -> {
                review.setStatus(REVIEW_STATUS_APPROVED);
                review.setApprovedAt(now);
            })
            .toList();
        if (!reviewsToSave.isEmpty()) {
            developmentPlanReviewService.saveAll(reviewsToSave);
        }

        finalizePlanActivationIfReady(plan);

        return toPlanView(plan);
    }

```

```

@Transactional
public DevelopmentPlanTaskView updateTaskStatus(Long planId,
Long taskId, DevelopmentPlanTaskStatusUpdateRequest request) {
    if (request.participantEmployeeId() == null) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_task_status.partic
ipant_required")
        );
    }

    DevelopmentPlanEntity plan = getPlanOrThrow(planId);
    validatePlanIsActiveAndApproved(plan);

    DevelopmentPlanTaskEntity task =
getTaskAndValidateBelongsToPlan(taskId, planId);

    if
(DevelopmentPlanTaskStatusEnum.DONE.equals(request.status())) {
        EmployeeEntity teamLead = plan.getTeamLead();
        if (teamLead == null) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_competency.team_le
ad_not_assigned")
            );
        }
        if (!Objects.equals(teamLead.getId(),
request.participantEmployeeId())) {
            throw new ResponseStatusException(
                HttpStatus.FORBIDDEN,

messageUtils.getMessage("error.development_plan_task_status.team_l
ead_required_for_done")
            );
        }
        if (request.teamLeadTaskScore() == null) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_task_status.team_l
ead_score_required_for_done")
            );
        }

        validateTeamLeadTaskScore1To10(request.teamLeadTaskScore());

        task.setTeamLeadTaskScore(request.teamLeadTaskScore().shortValue()
);
    } else {

```

```

        if (!Objects.equals(plan.getEmployee().getId(),
request.participantEmployeeId())) {
            throw new ResponseStatusException(
                HttpStatus.FORBIDDEN,

messageUtils.getMessage("error.development_plan_task_status.partic
ipant_not_plan_employee")
            );
        }
        if (request.teamLeadTaskScore() != null) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_task_status.team_l
ead_score_forbidden_not_done")
            );
        }
        task.setTeamLeadTaskScore(null);
    }

    task.setStatus(request.status());
    DevelopmentPlanTaskEntity savedTask =
developmentPlanTaskService.save(task);
    return
developmentPlanViewMapper.toDevelopmentPlanTaskView(savedTask);
}

    private DevelopmentPlanView toPlanView(DevelopmentPlanEntity
plan) {
        List<DevelopmentPlanTaskView> taskViews =
developmentPlanTaskService.findAllByPlanIdOrderByDueDateAscIdAsc(p
lan.getId()).stream()

.map(developmentPlanViewMapper::toDevelopmentPlanTaskView)
        .toList();
        List<DevelopmentPlanCompetencyItemView>
competencyItemViews =

developmentPlanCompetencyManagementService.listCompetencyItemViews
ByPlanId(plan.getId());
        List<DevelopmentPlanReviewView> reviewViews =
developmentPlanReviewService

.findAllByPlanIdOrderByCreatedAtAscIdAsc(plan.getId()).stream()

.map(developmentPlanViewMapper::toDevelopmentPlanReviewView)
        .toList();
        return
developmentPlanViewMapper.toDevelopmentPlanView(plan, taskViews,
competencyItemViews, reviewViews);
    }

    private DevelopmentPlanEntity getPlanOrThrow(Long planId) {

```

```

        return developmentPlanService.findById(planId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.development_plan",
planId)));
    }

    private DevelopmentPlanTaskEntity
getTaskAndValidateBelongsToPlan(Long taskId, Long planId) {
        DevelopmentPlanTaskEntity task =
developmentPlanTaskService.findById(taskId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.development_plan_task",
taskId)));
        validateTaskBelongsToPlan(task, planId);
        return task;
    }

    private EmployeeEntity resolveManager(EmployeeEntity employee,
Long managerId) {
        EmployeeEntity manager = getEmployeeOrThrow(managerId);
        validateEmployeeActive(manager);
        ensureSameCompany(employee, manager);
        return manager;
    }

    private EmployeeEntity resolveTeamLead(EmployeeEntity
employee, Long teamLeadId) {
        if (teamLeadId == null) {
            return null;
        }
        EmployeeEntity teamLead = getEmployeeOrThrow(teamLeadId);
        validateEmployeeActive(teamLead);
        ensureSameCompany(employee, teamLead);
        return teamLead;
    }

    private void applyTaskUpdate(DevelopmentPlanTaskEntity task,
                                DevelopmentPlanEntity plan,
                                DevelopmentPlanTaskUpdateRequest
request) {
        validateTaskDurationDays(request.durationDays());

        String title = request.title().trim();
        validateTaskTitle(title);

        String description = request.description().trim();
        String successCriteria = request.successCriteria().trim();

        LocalDate plannedStartDate = request.plannedStartDate() ==
null ? plan.getPeriodStart() : request.plannedStartDate();

```

```

        validateTaskPlannedStartDate(plannedStartDate,
plan.getPeriodStart(), plan.getPeriodEnd());

        LocalDate dueDate =
plannedStartDate.plusDays(request.durationDays() - 1L);
        validateTaskDueDate(dueDate, plan.getPeriodEnd());

        validateTaskEffortHours(request.effortHoursPlanned());

        task.setTaskType(request.taskType());
        task.setTitle(title);
        task.setDescription(description);
        task.setSuccessCriteria(successCriteria);
        task.setPriority(request.priority());
        task.setPlannedStartDate(plannedStartDate);
        task.setDueDate(dueDate);
        task.setEffortHoursPlanned(request.effortHoursPlanned());
    }

    private void
validateCreatePlanRequest(DevelopmentPlanCreateRequest request) {
        validatePlanPeriod(request.periodStart(),
request.periodEnd());
        validateTaskList(request.tasks());
    }

    private EmployeeEntity
resolveTeamLeadForPlanCreate(EmployeeEntity employee, Long
teamLeadId) {
        if (teamLeadId == null) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.team_lead_required
")
            );
        }
        return resolveTeamLead(employee, teamLeadId);
    }

    private GradeEntity resolveTargetGradeForPlanCreate(Long
targetGradeId, EmployeeEntity employee) {
        GradeEntity grade =
gradeService.findById(targetGradeId).orElseThrow(() -> new
ResponseStatusException(
            HttpStatus.NOT_FOUND,
            messageUtils.getMessage("error.not_found.grade",
targetGradeId)
        ));
        if (!Objects.equals(grade.getCompany().getId(),
employee.getCompany().getId())) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,

```

```

messageUtils.getMessage("error.development_plan.target_grade_compa
ny_mismatch")
        );
    }
    if (Boolean.FALSE.equals(grade.getIsActive())) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan.target_grade_inact
ive")
        );
    }
    return grade;
}

    private void validateApprovePlanRequest(Long
participantEmployeeId) {
        if (participantEmployeeId == null) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.participant_employ
ee_id_required"));
        }
    }

    private boolean isPlanFullyApproved(DevelopmentPlanEntity
plan) {
        return PLAN_STATUS_ACTIVE.equals(plan.getStatus());
    }

    private void validatePlanIsDraft(DevelopmentPlanEntity plan) {
        if (!PLAN_STATUS_DRAFT.equals(plan.getStatus())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.status_not_draft")
);
        }
    }

    private void
validatePlanAllowsTaskCreate(DevelopmentPlanEntity plan, Long
teamLeadEmployeeId) {
        if (PLAN_STATUS_DRAFT.equals(plan.getStatus())) {
            return;
        }
        if (PLAN_STATUS_ACTIVE.equals(plan.getStatus())) {
            EmployeeEntity teamLead = plan.getTeamLead();
            if (teamLead == null ||
!Objects.equals(teamLead.getId(), teamLeadEmployeeId)) {
                throw new ResponseStatusException(
                    HttpStatus.FORBIDDEN,

```

```

messageUtils.getMessage("error.development_plan_task_create.forbid
den_not_team_lead")
        );
    }
    validateEmployeeActive(teamLead);
    return;
}
throw new ResponseStatusException(
    HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_task_create.plan_s
tatus_forbidden")
    );
}

    private void validatePlanAllowsApproval(DeploymentPlanEntity
plan) {
        validatePlanIsDraft(plan);
    }

    private void
validateParticipantReviews(List<DevelopmentPlanReviewEntity>
participantReviews) {
        if (participantReviews.isEmpty()) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.participant_not_in
_reviewers"));
        }
    }

    private void
validatePlanIsActiveAndApproved(DeploymentPlanEntity plan) {
        if (!PLAN_STATUS_ACTIVE.equals(plan.getStatus())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.status_not_active_
or_not_approved"));
        }
    }

    private void
finalizePlanActivationIfReady(DeploymentPlanEntity plan) {
        Long planId = plan.getId();
        if
(developmentPlanReviewService.countByPlanIdAndStatus(planId,
REVIEW_STATUS_PENDING) > 0) {
            return;
        }

        List<DevelopmentPlanReviewEntity> reviews =

```

```

developmentPlanReviewService.findAllByPlanIdOrderByCreatedAtAscIdAsc(planId);
    if (reviews.isEmpty()) {
        return;
    }

    boolean allApproved = reviews.stream().allMatch(review ->
REVIEW_STATUS_APPROVED.equals(review.getStatus()));
    if (!allApproved) {
        return;
    }

    if (PLAN_STATUS_ACTIVE.equals(plan.getStatus())) {
        return;
    }

    if
(developmentPlanService.existsOtherActivePlanForEmployee(plan.getEmployee().getId(), planId)) {
        return;
    }

    LocalDateTime approvedAt = reviews.stream()
        .map(DevelopmentPlanReviewEntity::getApprovedAt)
        .filter(Objects::nonNull)
        .max(LocalDateTime::compareTo)
        .orElse(LocalDateTime.now());

    plan.setStatus(PLAN_STATUS_ACTIVE);
    if (plan.getApprovedAt() == null) {
        plan.setApprovedAt(approvedAt);
    }
    developmentPlanService.save(plan);
}

private void
validateTaskBelongsToPlan(DevelopmentPlanTaskEntity task, Long
planId) {
    if (!Objects.equals(task.getPlan().getId(), planId)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_not_belongs_to_plan"));
    }
}

private void validateTaskTitle(String title) {
    if (title.length() > 255) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_title_too_long"));
    }
}

```

```

    }
}

private void validateTaskDurationDays(Integer durationDays) {
    if (durationDays == null || durationDays <= 0) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_duration_days
_invalid"));
    }
}

private void validateTaskPlannedStartDate(LocalDate
plannedStartDate, LocalDate periodStart, LocalDate periodEnd) {
    if (plannedStartDate.isBefore(periodStart) ||
plannedStartDate.isAfter(periodEnd)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_planned_start
_out_of_plan_period"));
    }
}

private void validateTaskDueDate(LocalDate dueDate, LocalDate
periodEnd) {
    if (dueDate.isAfter(periodEnd)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_due_date_out_
of_plan_period"));
    }
}

private void validateTaskEffortHours(Integer
effortHoursPlanned) {
    if (effortHoursPlanned != null && effortHoursPlanned <= 0)
    {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.task_effort_hours_
_invalid"));
    }
}

private void validateAllTaskDatesFitPeriod(Long planId,
LocalDate newPeriodStart, LocalDate newPeriodEnd) {
    List<DevelopmentPlanTaskEntity> tasks =
developmentPlanTaskService.findAllByPlanId(planId);
    for (DevelopmentPlanTaskEntity task : tasks) {

validateTaskPlannedStartDate(task.getPlannedStartDate(),
newPeriodStart, newPeriodEnd);
        validateTaskDueDate(task.getDueDate(), newPeriodEnd);
    }
}

```

```

    }
}

private DevelopmentPlanTaskEntity
buildTaskEntity(DevelopmentPlanEntity plan,
DevelopmentPlanTaskCreateRequest request) {
    DevelopmentPlanTaskTypeEnum taskType = request.taskType();

    String title = request.title().trim();
    validateTaskTitle(title);

    String description = request.description().trim();
    String successCriteria = request.successCriteria().trim();

    DevelopmentPlanTaskPriorityEnum priority =
request.priority();
    validateTaskDurationDays(request.durationDays());

    LocalDate plannedStartDate = request.plannedStartDate() ==
null ? plan.getPeriodStart() : request.plannedStartDate();
    validateTaskPlannedStartDate(plannedStartDate,
plan.getPeriodStart(), plan.getPeriodEnd());

    LocalDate dueDate =
plannedStartDate.plusDays(request.durationDays() - 1L);
    validateTaskDueDate(dueDate, plan.getPeriodEnd());

    validateTaskEffortHours(request.effortHoursPlanned());

    DevelopmentPlanTaskEntity task = new
DevelopmentPlanTaskEntity();
    task.setPlan(plan);
    task.setTaskType(taskType);
    task.setTitle(title);
    task.setDescription(description);
    task.setSuccessCriteria(successCriteria);
    task.setPriority(priority);
    task.setPlannedStartDate(plannedStartDate);
    task.setDueDate(dueDate);
    task.setStatus(TASK_STATUS_PLANNED);
    task.setEffortHoursPlanned(request.effortHoursPlanned());
    return task;
}

private DevelopmentPlanReviewEntity
buildReviewEntity(DevelopmentPlanEntity plan,
EmployeeEntity reviewer,
DevelopmentPlanReviewerRoleEnum reviewerRole) {
    DevelopmentPlanReviewEntity review = new
DevelopmentPlanReviewEntity();

```

```

        review.setPlan(plan);
        review.setReviewer(reviewer);
        review.setReviewerRole(reviewerRole);
        review.setStatus(REVIEW_STATUS_PENDING);
        review.setApprovedAt(null);
        review.setCreatedAt(LocalDateTime.now());
        return review;
    }

    private void validateTeamLeadTaskScore1To10(int score) {
        if (score < 1 || score > 10) {
            throw new ResponseStatusException(
                HttpStatus.BAD_REQUEST,
                messageUtils.getMessage("error.development_plan_task.team_lead_score_out_of_range")
            );
        }
    }

    private void ensureActivePolicyExists(Long companyId,
        LocalDate asOf) {
        promotionPolicyService.findEffectiveByCompanyId(companyId,
            asOf)
            .orElseThrow(() -> new
                ResponseStatusException(HttpStatus.NOT_FOUND,
                    messageUtils.getMessage("error.not_found.active_promotion_policy_by_company", companyId)));
    }

    private void validatePlanPeriod(LocalDate periodStart,
        LocalDate periodEnd) {
        if (periodEnd.isBefore(periodStart)) {
            throw new
                ResponseStatusException(HttpStatus.BAD_REQUEST,
                    messageUtils.getMessage("error.development_plan.period_invalid"));
        }
    }

    private void
        validateTaskList(List<DevelopmentPlanTaskCreateRequest> tasks) {
        if (tasks == null || tasks.isEmpty()) {
            throw new
                ResponseStatusException(HttpStatus.BAD_REQUEST,
                    messageUtils.getMessage("error.development_plan.task_list_empty"));
        }
    }

    private EmployeeEntity getEmployeeOrThrow(Long employeeId) {
        return employeeService.findById(employeeId)
    }

```

```

        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.employee", employeeId)));
    }

    private void validateEmployeeActive(EmployeeEntity employee) {
        if (!Boolean.TRUE.equals(employee.getIsActive())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.employee_inactive"
));
        }
    }

    private void ensureSameCompany(EmployeeEntity employee,
EmployeeEntity relatedEmployee) {
        if (!Objects.equals(employee.getCompany().getId(),
relatedEmployee.getCompany().getId())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan.employee_company_m
ismatch"));
        }
    }
}

```

Листинг Б.2 – Файл DevelopmentPlanTaskExecutionManagementService.java

```

package lrm.employeeprogressservice.service;

import lombok.RequiredArgsConstructor;
import
lrm.employeeprogressservice.mapper.developmentplan.DevelopmentPlan
TaskExecutionMapViewMapper;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanEntity;
import
lrm.employeeprogressservice.model.dao.DevelopmentPlanTaskEntity;
import lrm.employeeprogressservice.model.dao.EmployeeEntity;
import
lrm.employeeprogressservice.model.dao.TaskProgressEntryEntity;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskCommentCreateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskCommentUpdateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskCommentView;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskProgressRecordView;

```

```

import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskProgressUpdateRequest;
import
lrm.employeeprogressservice.model.dto.developmentplan.DevelopmentP
lanTaskProgressUpdateView;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanStatusEnum;
import
lrm.employeeprogressservice.model.enums.DevelopmentPlanTaskStatusE
num;
import
lrm.employeeprogressservice.model.enums.TaskProgressEntryTypeEnum;
import lrm.employeeprogressservice.utils.MessageUtils;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.server.ResponseStatusException;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Objects;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class DevelopmentPlanTaskExecutionManagementService {

    private static final int ZERO = 0;
    private static final int HUNDRED = 100;

    private final DevelopmentPlanService developmentPlanService;
    private final DevelopmentPlanTaskService
developmentPlanTaskService;
    private final TaskProgressEntryService
taskProgressEntryService;
    private final DevelopmentPlanTaskExecutionViewMapper
developmentPlanTaskExecutionViewMapper;

    private final MessageUtils messageUtils;

    @Transactional
    public DevelopmentPlanTaskCommentView addTaskComment(Long
planId,
                                                         Long
taskId,
DevelopmentPlanTaskCommentCreateRequest request,
                                                         Long
authorEmployeeId) {
        String comment = request.comment().trim();

        DevelopmentPlanEntity plan = getPlanOrThrow(planId);

```

```

        validatePlanReadyForExecution(planId);
        DevelopmentPlanTaskEntity task =
validateTaskBelongsToPlan(taskId, planId);
        EmployeeEntity author = resolveCommentAuthor(plan,
authorEmployeeId);

        TaskProgressEntryEntity entry = new
TaskProgressEntryEntity();
        entry.setTask(task);
        entry.setAuthorEmployee(author);
        entry.setEntryType(TaskProgressEntryTypeEnum.COMMENT);
        entry.setProgressPercent(null);
        entry.setComment(comment);
        entry.setArtifactUrl(null);
        entry.setCreatedAt(LocalDateTime.now());

        TaskProgressEntryEntity savedEntry =
taskProgressEntryService.save(entry);
        return
developmentPlanTaskExecutionViewMapper.toDevelopmentPlanTaskCommen
tView(savedEntry);
    }

    @Transactional
    public DevelopmentPlanTaskProgressUpdateView
recordTaskProgress(Long planId,

Long taskId,

DevelopmentPlanTaskProgressUpdateRequest request) {
        Integer progressPercent = request.progressPercent();
        validateProgressPercent(progressPercent);

        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        validatePlanReadyForExecution(planId);
        DevelopmentPlanTaskEntity task =
validateTaskBelongsToPlan(taskId, planId);
        validateEmployeeActive(plan.getEmployee());

        DevelopmentPlanTaskStatusEnum currentTaskStatus =
resolveTaskStatusByProgress(progressPercent);
        task.setStatus(currentTaskStatus);

        developmentPlanTaskService.save(task);

        TaskProgressEntryEntity entry = new
TaskProgressEntryEntity();
        entry.setTask(task);
        entry.setAuthorEmployee(plan.getEmployee());
        entry.setEntryType(TaskProgressEntryTypeEnum.PROGRESS);
        entry.setProgressPercent(progressPercent);
        entry.setComment(trimOptional(request.comment()));
        entry.setArtifactUrl(null);

```

```

        entry.setCreatedAt(LocalDateTime.now());

        TaskProgressEntryEntity savedEntry =
taskProgressEntryService.save(entry);
        return
developmentPlanTaskExecutionViewMapper.toDevelopmentPlanTaskProgre
ssUpdateView(savedEntry, currentTaskStatus);
    }

    public List<DevelopmentPlanTaskCommentView>
getTaskComments(Long planId, Long taskId) {
        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        validatePlanReadyForExecution(planId);
        validateTaskBelongsToPlan(taskId, planId);

        return
taskProgressEntryService.findAllByTaskIdAndEntryTypeOrderByCreated
AtAscIdAsc(taskId, TaskProgressEntryTypeEnum.COMMENT)
            .stream()

.map(developmentPlanTaskExecutionViewMapper::toDevelopmentPlanTask
CommentView)

            .toList();
    }

    public List<DevelopmentPlanTaskProgressRecordView>
getTaskProgressHistory(Long planId, Long taskId) {
        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        validatePlanReadyForExecution(planId);
        validateTaskBelongsToPlan(taskId, planId);

        return
taskProgressEntryService.findAllByTaskIdAndEntryTypeOrderByCreated
AtAscIdAsc(taskId, TaskProgressEntryTypeEnum.PROGRESS)
            .stream()

.map(developmentPlanTaskExecutionViewMapper::toDevelopmentPlanTask
ProgressRecordView)

            .toList();
    }

    @Transactional
    public DevelopmentPlanTaskCommentView updateTaskComment(Long
planId,
                                                                    Long
taskId,
                                                                    Long
commentId,

DevelopmentPlanTaskCommentUpdateRequest request) {
        String comment = request.comment().trim();

        DevelopmentPlanEntity plan = getPlanOrThrow(planId);

```

```

        validatePlanReadyForExecution(planId);
        validateTaskBelongsToPlan(taskId, planId);
        validateEmployeeActive(plan.getEmployee());

        TaskProgressEntryEntity entry =
getCommentEntryOrThrow(commentId, taskId);
        entry.setComment(comment);

        TaskProgressEntryEntity savedEntry =
taskProgressEntryService.save(entry);
        return
developmentPlanTaskExecutionViewModelMapper.toDevelopmentPlanTaskComm
tView(savedEntry);
    }

    @Transactional
    public void deleteTaskComment(Long planId, Long taskId, Long
commentId) {
        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        validatePlanReadyForExecution(planId);
        validateTaskBelongsToPlan(taskId, planId);
        validateEmployeeActive(plan.getEmployee());

        getCommentEntryOrThrow(commentId, taskId);
        taskProgressEntryService.deleteById(commentId);
    }

    private DevelopmentPlanEntity getPlanOrThrow(Long planId) {
        return developmentPlanService.findById(planId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.development_plan",
planId)));
    }

    private void validatePlanReadyForExecution(Long planId) {
        DevelopmentPlanEntity plan = getPlanOrThrow(planId);
        if
(!DevelopmentPlanStatusEnum.ACTIVE.equals(plan.getStatus())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan_task_execution.pla
n_not_active_or_not_approved"));
        }
    }

    private DevelopmentPlanTaskEntity
validateTaskBelongsToPlan(Long taskId, Long planId) {
        DevelopmentPlanTaskEntity task =
developmentPlanTaskService.findById(taskId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,

```

```

messageUtils.getMessage("error.not_found.development_plan_task",
taskId));
    if (!Objects.equals(task.getPlan().getId(), planId)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan_task_execution.task_not_belongs_to_plan"));
    }
    return task;
}

private TaskProgressEntryEntity getCommentEntryOrThrow(Long
commentId, Long taskId) {
    TaskProgressEntryEntity entry =
taskProgressEntryService.findById(commentId)
        .orElseThrow(() -> new ResponseStatusException(
            HttpStatus.NOT_FOUND,

messageUtils.getMessage("error.not_found.task_progress_entry",
commentId)

        ));
    if
(!TaskProgressEntryTypeEnum.COMMENT.equals(entry.getEntryType())
|| !Objects.equals(entry.getTask().getId(), taskId)) {
        throw new ResponseStatusException(
            HttpStatus.NOT_FOUND,

messageUtils.getMessage("error.not_found.task_progress_entry",
commentId)

        );
    }
    return entry;
}

private EmployeeEntity
resolveCommentAuthor(DevelopmentPlanEntity plan, Long
authorEmployeeId) {
    if (authorEmployeeId == null) {
        throw new ResponseStatusException(
            HttpStatus.BAD_REQUEST,

messageUtils.getMessage("error.development_plan_task_execution.com
ment_author_required")

        );
    }
    if (Objects.equals(plan.getEmployee().getId(),
authorEmployeeId)) {
        validateEmployeeActive(plan.getEmployee());
        return plan.getEmployee();
    }
    EmployeeEntity teamLead = plan.getTeamLead();
    if (teamLead != null && Objects.equals(teamLead.getId(),
authorEmployeeId)) {

```

```

        validateEmployeeActive(teamLead);
        return teamLead;
    }
    throw new ResponseStatusException(
        HttpStatus.FORBIDDEN,

messageUtils.getMessage("error.development_plan_task_execution.com
ment_author_forbidden")
    );
}

    private void validateEmployeeActive(EmployeeEntity employee) {
        if (!Boolean.TRUE.equals(employee.getIsActive())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan_task_execution.emp
loyee_inactive"));
        }
    }

    private void validateProgressPercent(Integer progressPercent)
{
        if (progressPercent < ZERO || progressPercent > HUNDRED) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.development_plan_task_execution.pro
gress_percent_out_of_range"));
        }
    }

    private DevelopmentPlanTaskStatusEnum
resolveTaskStatusByProgress(Integer progressPercent) {
        if (progressPercent == ZERO) {
            return DevelopmentPlanTaskStatusEnum.PLANNED;
        }
        if (progressPercent == HUNDRED) {
            return DevelopmentPlanTaskStatusEnum.DONE;
        }
        return DevelopmentPlanTaskStatusEnum.IN_PROGRESS;
    }

    private String trimOptional(String value) {
        if (value == null) {
            return null;
        }
        String trimmed = value.trim();
        return trimmed.isEmpty() ? null : trimmed;
    }
}

```

Листинг Б.3 – Файл GradeModelManagementService.java

```
package lrm.employeeprogressservice.service;
```

```

import lombok.RequiredArgsConstructor;
import
lrm.employeeprogressservice.mapper.grademodel.GradeModelViewMapper
;
import lrm.employeeprogressservice.model.dao.CompetencyEntity;
import lrm.employeeprogressservice.model.dao.CompanyEntity;
import
lrm.employeeprogressservice.model.dao.GradeCompetencyRequirementEn
tity;
import lrm.employeeprogressservice.model.dao.GradeEntity;
import lrm.employeeprogressservice.model.dao.PositionEntity;
import
lrm.employeeprogressservice.model.dao.RefCompetencyLevelEntity;
import
lrm.employeeprogressservice.model.dto.grademodel.CompetencyUpsertR
equest;
import
lrm.employeeprogressservice.model.dto.grademodel.CompetencyView;
import
lrm.employeeprogressservice.model.dto.grademodel.GradeCriterionUps
ertRequest;
import
lrm.employeeprogressservice.model.dto.grademodel.GradeCriterionVie
w;
import
lrm.employeeprogressservice.model.dto.grademodel.GradeModelMatrixV
iew;
import
lrm.employeeprogressservice.model.dto.grademodel.GradeRegistryRequ
est;
import
lrm.employeeprogressservice.model.dto.grademodel.GradeUpsertReques
t;
import lrm.employeeprogressservice.model.dto.grademodel.GradeView;
import
lrm.employeeprogressservice.model.dto.grademodel.PositionMatrixVie
w;
import
lrm.employeeprogressservice.model.dto.grademodel.PositionRegistryR
equest;
import
lrm.employeeprogressservice.model.dto.grademodel.PositionUpsertReq
uest;
import
lrm.employeeprogressservice.model.dto.grademodel.PositionView;
import
lrm.employeeprogressservice.model.dto.grademodel.RefCompetencyLeve
lView;
import
lrm.employeeprogressservice.service.specification.GradeSpecificati
onBuilder;

```

```

import
    lrm.employeeprogressservice.service.specification.PositionSpecific
    ationBuilder;
import lrm.employeeprogressservice.utils.MessageUtils;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.server.ResponseStatusException;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;

@Service
@Transactional(readOnly = true)
@RequiredArgsConstructor
public class GradeModelManagementService {

    private final CompanyService companyService;
    private final PositionService positionService;
    private final GradeService gradeService;
    private final CompetencyService competencyService;
    private final RefCompetencyLevelService
refCompetencyLevelService;
    private final GradeCompetencyRequirementService
requirementService;
    private final GradeModelViewMapper gradeModelViewMapper;
    private final MessageUtils messageUtils;

    public List<PositionView> getPositions(Long companyId, boolean
onlyActive) {
        ensureCompanyExists(companyId);
        List<PositionEntity> positions = onlyActive
            ? positionService.findActiveByCompanyId(companyId)
            : positionService.findAllByCompanyId(companyId);
        return positions.stream()
            .map(gradeModelViewMapper::toPositionView)
            .toList();
    }

    public Page<PositionView>
getPositionRegistry(PositionRegistryRequest request, Pageable
pageable) {
        PositionSpecificationBuilder specificationBuilder = new
PositionSpecificationBuilder()
            .withCompanyIdEq(request.companyId())

```

```

        .withIsActiveEq(request.isActive())
        .withCodeLike(request.codeLike())
        .withNameLike(request.nameLike());

    return
positionService.findAll(specificationBuilder.build(), pageable)
        .map(gradeModelViewMapper::toPositionView);
}

@Transactional
public Long createPosition(Long companyId,
PositionUpsertRequest request) {
    String normalizedCode = normalizeCode(request.code());
    if (positionService.existsByCompanyIdAndCode(companyId,
normalizedCode)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.position_code_duplicate
"));
    }

    CompanyEntity company = companyService.findById(companyId)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.company", companyId)));

    PositionEntity position = new PositionEntity();
    position.setCompany(company);
    position.setCode(normalizedCode);
    position.setName(request.name().trim());

    position.setDescription(trimToNull(request.description()));
    position.setIsActive(request.isActive() == null ?
Boolean.TRUE : request.isActive());
    return positionService.save(position).getId();
}

@Transactional
public PositionView updatePosition(Long positionId,
PositionUpsertRequest request) {
    PositionEntity position = getPositionOrThrow(positionId);
    Long companyId = position.getCompany().getId();

    if (request.code() != null) {
        String normalizedCode = normalizeCode(request.code());
        if (!Objects.equals(position.getCode(),
normalizedCode) &&
positionService.existsByCompanyIdAndCode(companyId,
normalizedCode)) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.position_code_duplicate
"));
        }
    }

```

```

        }
        position.setCode(normalizedCode);
    }
    if (request.name() != null) {
        position.setName(request.name().trim());
    }
    if (request.description() != null) {
position.setDescription(trimToNull(request.description()));
    }
    if (request.isActive() != null) {
        position.setIsActive(request.isActive());
    }

    return
gradeModelViewMapper.toPositionView(positionService.save(position)
);
    }

    @Transactional
    public void deactivatePosition(Long positionId) {
        PositionEntity position = getPositionOrThrow(positionId);
        position.setIsActive(Boolean.FALSE);
        positionService.save(position);
    }

    public List<GradeView> getGrades(Long companyId, Long
positionId, boolean onlyActive) {
        ensureCompanyExists(companyId);
        List<GradeEntity> grades;
        if (positionId == null) {
            grades = onlyActive
                ?
gradeService.findActiveByCompanyId(companyId)
                : gradeService.findAllByCompanyId(companyId);
        } else {
            grades = onlyActive
                ?
gradeService.findActiveByCompanyIdAndPositionId(companyId,
positionId)
                :
gradeService.findAllByCompanyIdAndPositionId(companyId,
positionId);
        }

        return grades.stream()
            .map(grade ->
gradeModelViewMapper.toGradeView(grade,
findCriteria(grade.getId())))
            .toList();
    }

```

```

    public Page<GradeView> getGradeRegistry(GradeRegistryRequest
request, Pageable pageable) {
        GradeSpecificationBuilder specificationBuilder = new
GradeSpecificationBuilder()
            .withCompanyIdEq(request.companyId())
            .withPositionIdEq(request.positionId())
            .withIsActiveEq(request.isActive())
            .withLevelFrom(request.levelFrom())
            .withLevelTo(request.levelTo())
            .withCodeLike(request.codeLike())
            .withNameLike(request.nameLike());

        return gradeService.findAll(specificationBuilder.build(),
pageable)
            .map(grade ->
gradeModelViewMapper.toGradeView(grade,
findCriteria(grade.getId())));
    }

    @Transactional
    public Long createGrade(Long companyId, GradeUpsertRequest
request) {
        validateSalaryRange(request.salaryMinAmount(),
request.salaryMaxAmount());

        String normalizedCode = normalizeCode(request.code());
        if (gradeService.existsByCompanyIdAndCode(companyId,
normalizedCode)) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.grade_code_duplicate"))
;
        }

        PositionEntity position =
resolvePositionForCompany(companyId, request.positionId());
        checkLevelOrderConflict(companyId, position.getId(),
request.levelOrder(), null);

        CompanyEntity company = companyService.findById(companyId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.company", companyId)));

        GradeEntity grade = new GradeEntity();
        grade.setCompany(company);
        grade.setPosition(position);
        grade.setCode(normalizedCode);
        grade.setName(request.name().trim());
        grade.setLevelOrder(request.levelOrder());
        grade.setDescription(trimToNull(request.description()));
        grade.setSalaryMinAmount(request.salaryMinAmount());
        grade.setSalaryMaxAmount(request.salaryMaxAmount());
    }

```

```

        grade.setIsActive(request.isActive() == null ?
Boolean.TRUE : request.isActive());

        return gradeService.save(grade).getId();
    }

    @Transactional
    public GradeView updateGrade(Long gradeId, GradeUpsertRequest
request) {
        GradeEntity grade = gradeService.findById(gradeId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.grade", gradeId)));

        Long companyId = grade.getCompany().getId();

        if (request.code() != null) {
            String normalizedCode = normalizeCode(request.code());
            if (!Objects.equals(grade.getCode(), normalizedCode)
&& gradeService.existsByCompanyIdAndCode(companyId,
normalizedCode)) {
                throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.grade_code_duplicate"))
;
            }
            grade.setCode(normalizedCode);
        }
        if (request.name() != null) {
            grade.setName(request.name().trim());
        }

        Long currentPositionId = grade.getPosition() == null ?
null : grade.getPosition().getId();
        Long newPositionId = request.positionId() == null ?
currentPositionId : request.positionId();
        PositionEntity newPosition =
resolvePositionForCompany(companyId, newPositionId);
        Short newLevelOrder = request.levelOrder() == null ?
grade.getLevelOrder() : request.levelOrder();
        checkLevelOrderConflict(companyId, newPositionId,
newLevelOrder, grade.getId());

        grade.setPosition(newPosition);
        grade.setLevelOrder(newLevelOrder);

        if (request.description() != null) {
grade.setDescription(trimToNull(request.description()));
        }
        if (request.salaryMinAmount() != null ||
request.salaryMaxAmount() != null) {

```

```

        Integer min = request.salaryMinAmount() == null ?
grade.getSalaryMinAmount() : request.salaryMinAmount();
        Integer max = request.salaryMaxAmount() == null ?
grade.getSalaryMaxAmount() : request.salaryMaxAmount();
        validateSalaryRange(min, max);
        grade.setSalaryMinAmount(min);
        grade.setSalaryMaxAmount(max);
    }
    if (request.isActive() != null) {
        grade.setIsActive(request.isActive());
    }

    GradeEntity saved = gradeService.save(grade);
    return gradeModelViewMapper.toGradeView(saved,
findCriteria(saved.getId()));
}

@Transactional
public void deactivateGrade(Long gradeId) {
    GradeEntity grade = gradeService.findById(gradeId)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.grade", gradeId)));
    grade.setIsActive(Boolean.FALSE);
    gradeService.save(grade);
}

public List<CompetencyView> getCompetencies() {
    return competencyService.findAll().stream()
        .map(gradeModelViewMapper::toCompetencyView)
        .toList();
}

public List<RefCompetencyLevelView>
getCompetencyLevels(boolean onlyActive) {
    return refCompetencyLevelService.findAll().stream()
        .filter(level -> !onlyActive ||
Boolean.TRUE.equals(level.getIsActive()))
        .sorted(Comparator.comparing(RefCompetencyLevelEntity::getOrderNum
))
        .map(gradeModelViewMapper::toRefCompetencyLevelView)
        .toList();
}

@Transactional
public Long createCompetency(CompetencyUpsertRequest request)
{
    String normalizedCode = normalizeCode(request.code());
    if (competencyService.existsByCode(normalizedCode)) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,

```

```

messageUtils.getMessage("error.grade_model.competency_code_dupli-
cate"));
    }

    CompetencyEntity competency = new CompetencyEntity();
    competency.setCode(normalizedCode);
    competency.setName(request.name().trim());

    competency.setDescription(trimToNull(request.description()));
    return competencyService.save(competency).getId();
}

@Transactional
public CompetencyView updateCompetency(Long competencyId,
CompetencyUpsertRequest request) {
    CompetencyEntity competency =
competencyService.findById(competencyId)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.competency",
competencyId)));

    if (request.code() != null) {
        String normalizedCode = normalizeCode(request.code());
        if (!Objects.equals(competency.getCode(),
normalizedCode) && competencyService.existsByCode(normalizedCode))
        {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.competency_code_dupli-
cate"));
        }
        competency.setCode(normalizedCode);
    }
    if (request.name() != null) {
        competency.setName(request.name().trim());
    }
    if (request.description() != null) {

competency.setDescription(trimToNull(request.description()));
    }
    return
gradeModelViewMapper.toCompetencyView(competencyService.save(compe-
tency));
}

@Transactional
public void deleteCompetency(Long competencyId) {
    if (!competencyService.existsById(competencyId)) {
        throw new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.competency",
competencyId));
    }
}

```

```

    }
    try {
        competencyService.deleteById(competencyId);
    } catch (DataIntegrityViolationException ex) {
        throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.competency_used_in_crit
eria"));
    }
}

    public List<GradeCriterionView> getGradeCriteria(Long gradeId)
{
    gradeService.findById(gradeId)
        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.grade", gradeId)));
    return findCriteria(gradeId);
}

    @Transactional
    public Long createGradeCriterion(Long gradeId,
GradeCriterionUpsertRequest request) {
        GradeEntity grade = gradeService.findById(gradeId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.grade", gradeId)));

        if
(requestService.existsByGradeIdAndCompetencyId(gradeId,
request.competencyId())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.criterion_competency_du
plicate"));
        }

        CompetencyEntity competency =
competencyService.findById(request.competencyId())
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.competency",
request.competencyId())));
        RefCompetencyLevelEntity requiredLevel =
refCompetencyLevelService.findById(request.requiredLevelId())
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.ref_competency_level",
request.requiredLevelId())));

        GradeCompetencyRequirementEntity criterion = new
GradeCompetencyRequirementEntity();
        criterion.setGrade(grade);

```

```

        criterion.setCompetency(competency);
        criterion.setRequiredLevel(requiredLevel);

        return requirementService.save(criterion).getId();
    }

    @Transactional
    public GradeCriterionView updateGradeCriterion(Long
criterionId, GradeCriterionUpsertRequest request) {
        GradeCompetencyRequirementEntity criterion =
requirementService.findById(criterionId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.grade_competency_requirem
ent", criterionId)));
        GradeEntity grade = criterion.getGrade();

        Long newCompetencyId = request.competencyId() == null ?
criterion.getCompetency().getId() : request.competencyId();
        Long newRequiredLevelId = request.requiredLevelId() ==
null ? criterion.getRequiredLevel().getId() :
request.requiredLevelId();

        if (!Objects.equals(newCompetencyId,
criterion.getCompetency().getId())
            &&
requirementService.existsByGradeIdAndCompetencyId(grade.getId(),
newCompetencyId)) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.criterion_competency_du
plicate"));
        }

        CompetencyEntity competency =
competencyService.findById(newCompetencyId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.competency",
newCompetencyId)));
        RefCompetencyLevelEntity requiredLevel =
refCompetencyLevelService.findById(newRequiredLevelId)
            .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.ref_competency_level",
newRequiredLevelId)));

        criterion.setCompetency(competency);
        criterion.setRequiredLevel(requiredLevel);

        return
gradeModelViewMapper.toGradeCriterionView(requirementService.save(
criterion));

```

```

    }

    @Transactional
    public void deleteGradeCriterion(Long criterionId) {
        requirementService.findById(criterionId)
            .orElseThrow(() -> new
                ResponseStatusException(HttpStatus.NOT_FOUND,
                    messageUtils.getMessage("error.not_found.grade_competency_requirement", criterionId)));
        requirementService.deleteById(criterionId);
    }

    public GradeModelMatrixView getMatrix(Long companyId, boolean
        onlyActive) {
        ensureCompanyExists(companyId);

        List<PositionEntity> positions = onlyActive
            ? positionService.findActiveByCompanyId(companyId)
            : positionService.findAllByCompanyId(companyId);

        List<GradeEntity> grades = onlyActive
            ? gradeService.findActiveByCompanyId(companyId)
            : gradeService.findAllByCompanyId(companyId);

        Map<Long, List<GradeCriterionView>> criteriaByGradeId =
            grades.stream()
                .collect(Collectors.toMap(
                    GradeEntity::getId,
                    grade -> findCriteria(grade.getId())
                ));

        Map<Long, List<GradeView>> gradesByPositionId = new
            LinkedHashMap<>();
        for (GradeEntity grade :
            grades.stream().sorted(Comparator.comparing(GradeEntity::getLevelOrder)).toList()) {
            GradeView gradeView =
                gradeModelViewMapper.toGradeView(grade,
                    criteriaByGradeId.getOrDefault(grade.getId(), List.of()));
            gradesByPositionId.computeIfAbsent(grade.getPosition().getId(),
                ignored -> new ArrayList<>()).add(gradeView);
        }

        List<PositionMatrixView> positionMatrix =
            positions.stream()
                .map(position -> new PositionMatrixView(
                    gradeModelViewMapper.toPositionView(position),
                    gradesByPositionId.getOrDefault(position.getId(), List.of())
                ))
                .toList();
    }

```

```

        return new GradeModelMatrixView(companyId,
positionMatrix);
    }

    private List<GradeCriterionView> findCriteria(Long gradeId) {
        return
requirementService.findAllByGradeId(gradeId).stream()
                .map(gradeModelViewMapper::toGradeCriterionView)
                .toList();
    }

    private PositionEntity getPositionOrThrow(Long positionId) {
        return positionService.findById(positionId)
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.position", positionId)));
    }

    private PositionEntity resolvePositionForCompany(Long
companyId, Long positionId) {
        if (positionId == null) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.position_id_required"))
;
        }
        PositionEntity position =
positionService.findById(positionId)
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.position", positionId)));
        if (!companyId.equals(position.getCompany().getId())) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.position_company_mismat
ch"));
        }
        return position;
    }

    private void checkLevelOrderConflict(Long companyId, Long
positionId, Short levelOrder, Long currentGradeId) {
        if (positionId == null) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.position_id_required"))
;
        }
        boolean existsConflict = currentGradeId == null
?
gradeService.existsByCompanyIdAndPositionIdAndLevelOrder(companyId
, positionId, levelOrder)

```

```

        :
gradeService.existsByCompanyIdAndPositionIdAndLevelOrderAndIdNot(c
ompanyId, positionId, levelOrder, currentGradeId);
        if (existsConflict) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.level_order_conflict"))
;
        }
    }

    private void validateSalaryRange(Integer min, Integer max) {
        if (min != null && min < 0) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.salary_min_negative"));
        }
        if (max != null && max < 0) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.salary_max_negative"));
        }
        if (min != null && max != null && max < min) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST,
messageUtils.getMessage("error.grade_model.salary_range_invalid"))
;
        }
    }

    private String normalizeCode(String code) {
        return code.trim().toUpperCase();
    }

    private String trimToNull(String value) {
        if (value == null) {
            return null;
        }
        String trimmed = value.trim();
        return trimmed.isEmpty() ? null : trimmed;
    }

    private void ensureCompanyExists(Long companyId) {
        if (!companyService.existsById(companyId)) {
            throw new
ResponseStatusException(HttpStatus.NOT_FOUND,
messageUtils.getMessage("error.not_found.company", companyId));
        }
    }
}

```

Приложение В
(рекомендуемое)
Листинг модульных тестов

```

package lrm.employeeprogressservice.service;

import
lrm.employeeprogressservice.mapper.department.DepartmentViewMapper
;
import lrm.employeeprogressservice.model.dao.CompanyEntity;
import lrm.employeeprogressservice.model.dao.DepartmentEntity;
import
lrm.employeeprogressservice.model.dto.department.DepartmentUpsertR
equest;
import
lrm.employeeprogressservice.model.dto.department.DepartmentView;
import lrm.employeeprogressservice.repository.EmployeeRepository;
import
lrm.employeeprogressservice.testsupport.MessageUtilsTestSupport;
import lrm.employeeprogressservice.utils.MessageUtils;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.ArgumentMatchers.anyLong;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

@ExtendWith(MockitoExtension.class)
class DepartmentManagementServiceTest {

    private static final Long COMPANY_ID = 10L;
    private static final Long DEPARTMENT_ID_1 = 100L;
    private static final Long DEPARTMENT_ID_2 = 101L;

    @Mock
    private DepartmentService departmentService;
    @Mock
    private CompanyService companyService;
    @Mock
    private EmployeeRepository employeeRepository;
    @Mock
    private DepartmentViewMapper departmentViewMapper;
    @Mock
    private MessageUtils messageUtils;

    @InjectMocks
    private DepartmentManagementService service;

    @BeforeEach

```

```

void setUpMessageUtils() {
    MessageUtilsTestSupport.stubReturnI18nKeys(messageUtils);
}

@Test
void
should_return_departments_with_employee_count_when_only_active_is_
true() {
    DepartmentEntity first = department(DEPARTMENT_ID_1,
company(COMPANY_ID), "DEV", true);
    DepartmentEntity second = department(DEPARTMENT_ID_2,
company(COMPANY_ID), "QA", true);
    DepartmentView firstView = departmentView(DEPARTMENT_ID_1,
COMPANY_ID, "DEV", 2L, true);
    DepartmentView secondView =
departmentView(DEPARTMENT_ID_2, COMPANY_ID, "QA", 0L, true);

    when(companyService.existsById(COMPANY_ID)).thenReturn(true);

    when(departmentService.findActiveByCompanyId(COMPANY_ID)).thenRetu
rn(List.of(first, second));

    when(employeeRepository.countByDepartment_Id(DEPARTMENT_ID_1)).the
nReturn(2L);

    when(employeeRepository.countByDepartment_Id(DEPARTMENT_ID_2)).the
nReturn(0L);
    when(departmentViewMapper.toView(first,
2L)).thenReturn(firstView);
    when(departmentViewMapper.toView(second,
0L)).thenReturn(secondView);

    List<DepartmentView> result =
service.getDepartments(COMPANY_ID, true);

    assertThat(result).containsExactly(firstView, secondView);
    verify(departmentService,
never()).findAllByCompanyId(anyLong());
}

@Test
void
should_return_department_with_employee_count_when_get_by_id() {
    DepartmentEntity department = department(DEPARTMENT_ID_1,
company(COMPANY_ID), "DEV", true);
    DepartmentView mapped = departmentView(DEPARTMENT_ID_1,
COMPANY_ID, "DEV", 3L, true);

    when(departmentService.getById(DEPARTMENT_ID_1)).thenReturn(depart
ment);

```

```

when(employeeRepository.countByDepartment_Id(DEPARTMENT_ID_1)).thenReturn(3L);
    when(departmentViewMapper.toView(department,
3L)).thenReturn(mapped);

    DepartmentView result =
service.getDepartment(DEPARTMENT_ID_1);

    assertThat(result).isSameAs(mapped);
}

@Test
void
should_return_updated_department_with_employee_count_when_update()
{
    DepartmentEntity current = department(DEPARTMENT_ID_1,
company(COMPANY_ID), "DEV", true);
    DepartmentUpsertRequest request = new
DepartmentUpsertRequest(null, " Разработка ", null, null);
    DepartmentView mapped = departmentView(DEPARTMENT_ID_1,
COMPANY_ID, "DEV", 5L, true);

when(departmentService.getById(DEPARTMENT_ID_1)).thenReturn(current);
    when(departmentService.save(current)).thenReturn(current);

when(employeeRepository.countByDepartment_Id(DEPARTMENT_ID_1)).thenReturn(5L);
    when(departmentViewMapper.toView(current,
5L)).thenReturn(mapped);

    DepartmentView result =
service.updateDepartment(DEPARTMENT_ID_1, request);

    assertThat(result).isSameAs(mapped);
    assertThat(current.getName()).isEqualTo("Разработка");
}

private static CompanyEntity company(Long id) {
    CompanyEntity company = new CompanyEntity();
    company.setId(id);
    company.setName("Company " + id);
    company.setIsActive(true);
    return company;
}

private static DepartmentEntity department(Long id,
CompanyEntity company, String code, boolean isActive) {
    DepartmentEntity department = new DepartmentEntity();
    department.setId(id);
    department.setCompany(company);
}

```

```

        department.setCode(code);
        department.setName("Department " + code);
        department.setDescription("Description " + code);
        department.setIsActive(isActive);
        return department;
    }

    private static DepartmentView departmentView(Long id,
                                                    Long companyId,
                                                    String code,
                                                    Long
employeeCount,
                                                    boolean isActive)
    {
        return new DepartmentView(
            id,
            companyId,
            code,
            "Department " + code,
            "Description " + code,
            null,
            employeeCount,
            isActive
        );
    }
}

```