МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Бизнес-проект «Информационная система планирования отпусков. Серверная часть информационной системы контроля отпусков»

Студент _____ Акчурин Али Радикович
(подпись, дата) (ФИО полностью)

Руководитель _____ Самуйлов С.В.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 2026

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

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

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ В ФОРМЕ СТАРТАП-ПРОЕКТА

1. Студент Акчурин Али Радикович гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Бизнес-проект «Информационная система планирования отпусков. Серверная часть информационной системы контроля отпусков»

Тема утверждена приказом ПГУ №_____ от "___" _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Самуйлов С.В.

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

Назначение: система контроля отпусков предназначена для планирования и согласования отпусков сотрудников организации

Среда выполнения: операционная Windows

Языковые средства разработки: C#, UML, SQL, Linq

Инструментальные средства разработки: JetBrains Rider, Docker Compose, PgAdmin 4, Visual Paradigm, Swagger UI

Технологии разработки: Клиент-серверная архитектура, ООП, REST API, Dapper, JWT-аунтефикация, модульный монолит

Основные функции приложения:

а) регистрация и авторизация работников;

б) управление пользователями и организационной структорой предприятия;

в) создание и обработка заявок на отпуск;

г) согласование и управление отпусками;

д) настройка отпускной политики;

е) расчет отпускных дней и проверка ограничений;

ж) выявление конфлиектов при планировании;

и) просмотр календаря отпусков;

к) массовый импорт сотрудников из Excel;

л) формирование отчетов по отпускам в компании.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование программных средств

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 149 листов, 42 рисунка, 71 таблицу, 35 использованных источника, 7 приложений.

СЕРВЕРНАЯ ЧАСТЬ, СИСТЕМА КОНТРОЛЯ ОТПУСКОВ, ВЕБ-ПРИЛОЖЕНИЕ, ASP.NET CORE, POSTGRESQL, C#, DAPPER, JWT, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, МОДУЛЬНЫЙ МОНОЛИТ, UML, C4, DOCKER, ТЕСТИРОВАНИЕ ЧЕРНЫМ ЯЩИКОМ, ЮНИТ-ТЕСТЫ

Объектом разработки является серверная часть веб-приложения для автоматизации контроля отпусков сотрудников организации.

Цель работы – разработка программного обеспечения, позволяющего хранить и обрабатывать данные о сотрудниках, организационной структуре, отпускных политиках, заявках на отпуск, уведомлениях и истории изменений. Также целью является реализация логики проверки ограничений, пересечений и согласования отпусков.

Серверная часть реализована на языке C# с использованием платформы ASP.NET Core. В качестве СУБД была использована PostgreSQL, а для доступа к данным использована ORM Dapper.

Обеспечение разработано по принципу клиент-серверной архитектуры. Серверная часть построена по принципам архитектуры модульного монолита. Серверная часть разрабатывалась под управлением операционной системы Windows 11. Приложение предназначено для использования через современные браузеры. Например, Yandex, Google Chrome, Microsoft Edge и другие. Серверная часть может функционировать под управлением операционных систем Windows 10/11 и Linux.

Содержание

Введение 8

1 Стартап-часть 10

1.1 Описание существующей проблемы на рынке 10

1.2 Описание решения проблемы и продуктового решения 12

1.3 Объем, структура и развитие рынка 17

1.4 Анализ конкурентов 19

1.5 Бизнес-модель проекта 21

1.6 Команда стартап-проекта 22

1.7 Описание финансовой модели и необходимости привлечения инвестиций 23

1.8 Описание развития бизнес-проекта 25

1.9 Описание рисков 27

2 Анализ предметной области и постановка задачи на разработку средств автоматизации контроля отпусков 29

2.1 Анализ предметной области 29

2.2 Анализ программных решений аналогичного назначения 30

2.2.1 Популярные программные решения управления отпусками 31

2.2.2 Сравнительный анализ программных обеспечений 33

2.3 Анализ целевой аудитории программного продукта 35

2.4 Анализ требований 36

2.4.1 Анализ функциональных требований 37

2.4.2 Анализ нефункциональных требований 41

2.5 Планирование разработки и оценка бюджета 42

3 Проектирование серверной части приложения системы контроля отпусков.. 46
3.1 Выбор технологий разработки ... 46
3.2 Проектирование общей архитектуры программного обеспечения ... 47
3.3 Проектирование базы данных ... 49
3.3.1 Концептуальное проектирование базы данных ... 49
3.3.2 Логическое проектирование базы данных ... 52
3.4 Проектирование основных алгоритмов системы ... 54
3.5 Проектирование взаимодействия компонентов системы ... 58
3.6 Проектирование архитектуры серверной части приложения ... 59
4 Программная реализация ... 63
4.1 Реализация базы данных ... 63
4.2 Реализация серверной части ... 66
4.2.1 Описание архитектуры серверной части в нотации C4 ... 66
4.2.2 Структура модулей серверной части ... 73
4.2.3 Спецификация модулей серверной части ... 77
4.3 Структурная организация системы ... 78
5 Тестирование системы контроля отпусков ... 80
5.1 Обоснование выбора тестирования ... 80
5.2 Функциональное тестирование черным ящиком ... 81
5.3 Тестирование с помощью юнит-тестов ... 84
5.4 Оценка кода с помощью метрик ... 86
5.4.1 Обоснование выбора метрик кода ... 88
5.4.2 Метрика SLOC ... 89
5.4.3 Метрика Холстеда ... 90
Заключение ... 95

Список использованных источников 97

Приложение А (рекомендуемое) Спецификация диаграммы вариантов использования 101

Приложение Б (рекомендуемое) Спецификация логической модели базы данных 111

Приложение В (рекомендуемое) Спецификация физической модели базы данных 117

Приложение Г (рекомендуемое) Спецификация модулей системы контроля отпусков 123

Приложение Д (обязательное) Код программ 127

Приложение Е (рекомендуемое) Результаты тестирования черным ящиком 138

Приложение Ж (обязательное) Код юнит-тестов 144

Введение

Разработка информационной системы контроля отпусков обусловлена необходимостью автоматизации управления отпусками в организациях с большим числом сотрудников. В таких предприятиях ручное планирование может стать главной проблемой, из-за рисков ошибочного расчета отпускных дней и сложностями в процессе согласования заявок на отдых. В условиях роста цифровизации кадровых процессов возникает потребность в универсальном программном решении, обеспечивающем удобный контроль отпусков сотрудников.

Данный бизнес-проект, направлен на разработку продукта, который будет представлять ценность для организаций различного профиля, где необходим контроль над отпусками большого числа сотрудников. При этом разрабатываемая система ориентирована на применение в рамках действующего трудового законодательства Российской Федерации [1]. Это обеспечивает корректность учета отпускных дней и соблюдение нормативных требований.

Целью разработки является создание серверной части веб-ориентированной клиент-серверной системы, предназначенной для автоматизации планирования и согласования отпусков сотрудников в организациях различного профиля, где ручное планирование не столь эффективно как цифровое. Такой подход позволит повысить эффективность работы отделов управления кадрами и заметно снизить число ошибок при формировании графиков отпусков.

Для достижения поставленной цели решаются следующие задачи:

– проработка бизнес части проекта, включая анализ рынка, определение потенциальных пользователей и оценку перспектив внедрения системы;

– анализ предметной области и требований к системе управления отпусками;

– моделирование бизнес-процессов планирования и согласования отпусков;

– проектирование архитектуры системы и структуры базы данных;

– реализация серверной части с использованием актуальных технологий разработки;

– разработка логики отправки, обработки и согласования заявок на отпуск;

– обеспечение автоматической проверки пересечений дат, контроля лимитов отпускных дней и выявления конфликтных ситуаций;

– обеспечение гибкого управления политикой отпусков в компании;

– тестирование разработанных программных компонентов.

Разрабатываемый продукт представляет собой веб-ориентированное программное обеспечение в виде клиент-серверной системы [2], обеспечивающее подачу и согласование заявок на отпуск, централизованное хранение данных о сотрудниках и организациях, а также формирование наглядного календаря отпусков.

Система ориентирована на использование в организациях различного масштаба и может служить основой для дальнейшего развития программного продукта в рамках бизнес-проекта.

1 Стартап-часть

1.1 Описание существующей проблемы на рынке

На рынке управления персоналом одной из глобальных проблем является недостаток автоматизации процессов учета и планирования отпусков. Во многих организациях до сих пор данная операция происходит вручную или с использованием электронных таблиц. Такой подход приводит к ошибкам в расчетах отпускных дней и пересечению графиков отпусков.

В частности, данная проблема особо заметна в организациях с большим числом сотрудников, где необходимо учитывать загрузку подразделений и доступность персонала. Такая же проблема наблюдается и в предприятиях, где необходимо поддерживать работу круглосуточно. В таких условиях ручное управление отпусками становится неэффективным и увеличивает нагрузку на HR-специалистов и руководителей.

Основываясь на аналитических исследованиях, уровень автоматизации HR-процессов остается низким и по сей день. Согласно исследованию по автоматизации персонала, около 47% компаний продолжают использовать электронные таблицы (Excel) для управления кадровыми процессами, включая учет отпусков. Это увеличивает вероятность ошибок и снижает эффективность работы HR-подразделений. Подробнее с данным исследованием можно ознакомиться в статье «HR Automation Statistics» [3].

По данным другой работы по автоматизации процессов, 57% компаний используют автоматизацию HR для улучшения опыта сотрудников, а 51% организаций применяют автоматизацию для снижения затрат. Также отмечается, что автоматизация позволяет повысить производительность труда на 0,8–1,4% ежегодно. На рисунке 1 представлена диаграмма, отражающая преимущества внедрения автоматизации управления персоналом. Подробнее с данным исследованием можно ознакомиться в статье «120 Automation Statistics: Business, Machine Learning and Challenges» [4].

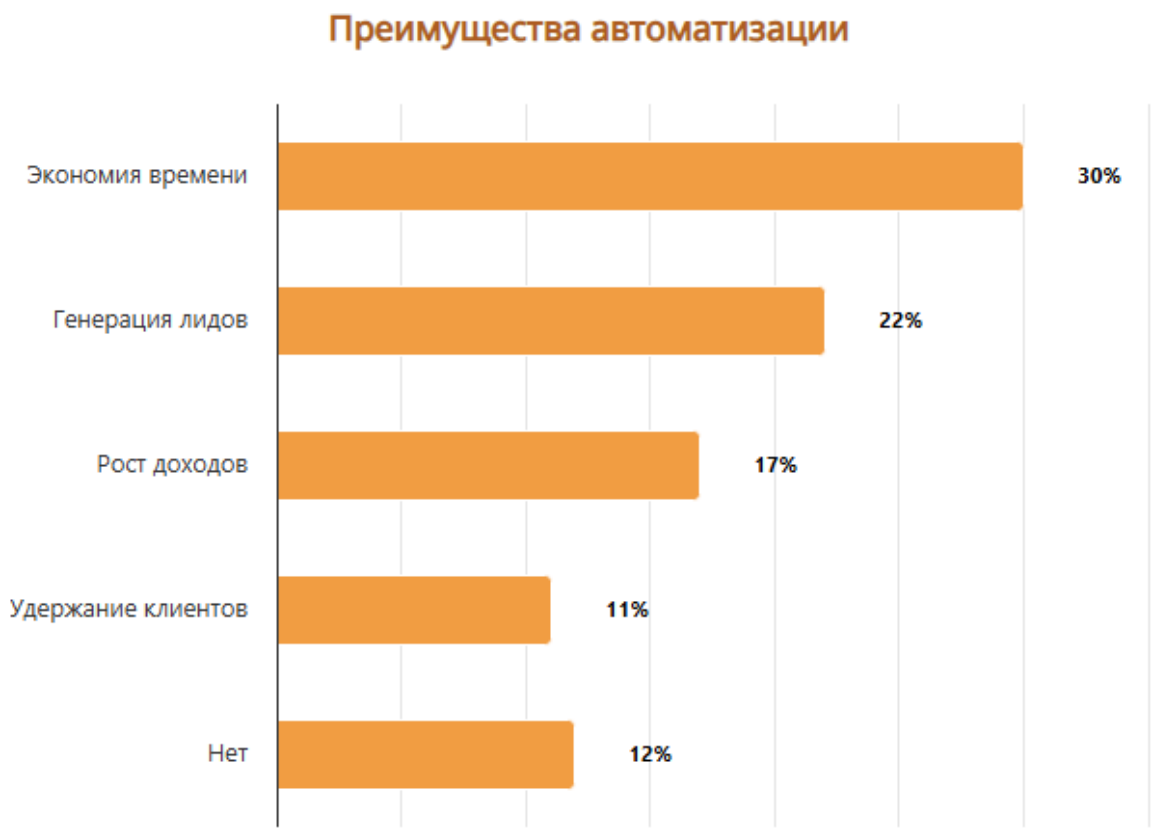

Рисунок 1 – Преимущества автоматизации отпусков

Дополнительно стоит отметить, что в организациях с круглосуточным режимом работы (например, военные или производственные предприятия) проблема планирования отпусков становится критически важной. Отсутствие контроля замещения сотрудников может привести к снижению производительности, возникновению простоев и как следствие необратимым последствиям.

Таким образом, неэффективное управление отпусками сотрудников - это актуальная проблема, существование которой подтверждается многими исследованиями. Это обосновывает необходимость разработки специализированного программного решения, направленного на автоматизацию процессов планирования и согласования отпусков. Данное решение позволит снизить количество ошибок и повысить эффективность управления персоналом.

1.2 Описание решения проблемы и продуктового решения

Для решения выявленной проблемы предлагается разработка информационной системы контроля отпусков, предназначенной для автоматизации процессов планирования, учета и согласования отпусков сотрудников. Предлагаемое решение направлено на устранение недостатков ручного управления, снижение количества ошибок и повышение эффективности работы HR-подразделений и руководителей.

Суть решения заключается в создании сайта, реализованного на основе клиент-серверной архитектуры, в котором серверная часть обеспечивает хранение данных, обработку запросов и выполнение бизнес-логики. Система позволяет централизованно управлять информацией о сотрудниках, подразделениях, отпускных политиках и заявках на отпуск.

Предлагаемое решение устраняет выявленные проблемы за счет автоматизации ключевых процессов. Заявки на отпуск формируются в системе и проходят этап согласования с учетом организационной структуры. При этом автоматически выполняется проверка пересечений дат, контроль доступных отпускных дней и соблюдение установленных ограничений. Это позволяет исключить ошибки, связанные с человеческим фактором и повысить эффективность обработки данных.

Особенностью системы является разграничение доступа на основе заданных ролей. В системе выделено несколько основных ролей: сотрудник, руководитель, HR-специалист и администратор. Каждая роль обладает собственным набором функциональных возможностей.

Сотрудник может создавать заявки на отпуск, редактировать их, отправлять на согласование, а также отслеживать статус заявок и просматривать остаток доступных отпускных дней. Интерфейс личного кабинета сотрудника представлен на рисунке 2.

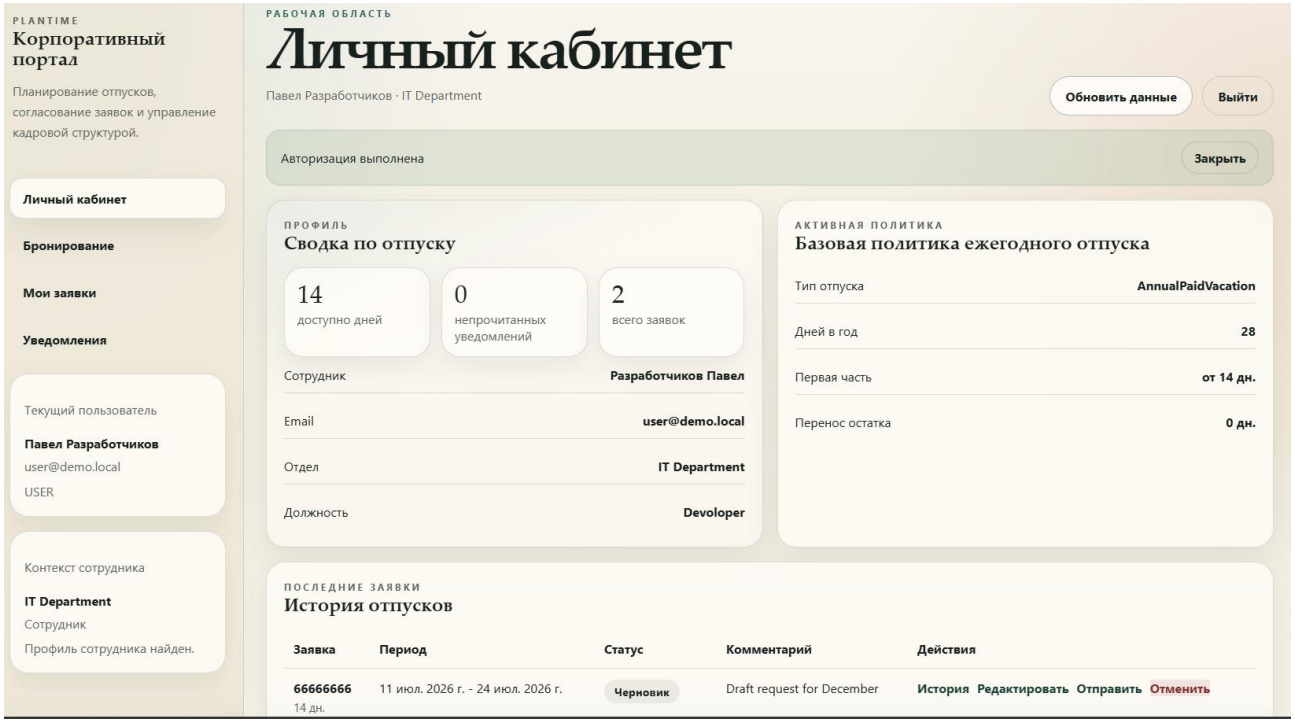

Рисунок 2 – Интерфейс личного кабинета сотрудника

Руководитель подразделения осуществляет согласование заявок сотрудников, анализирует загрузку подразделения и выявляет конфликты отпусков. Также руководителю доступен календарь отпусков команды. Интерфейс функции руководителя по просмотру календаря отпускуов представлен на рисунке 3.

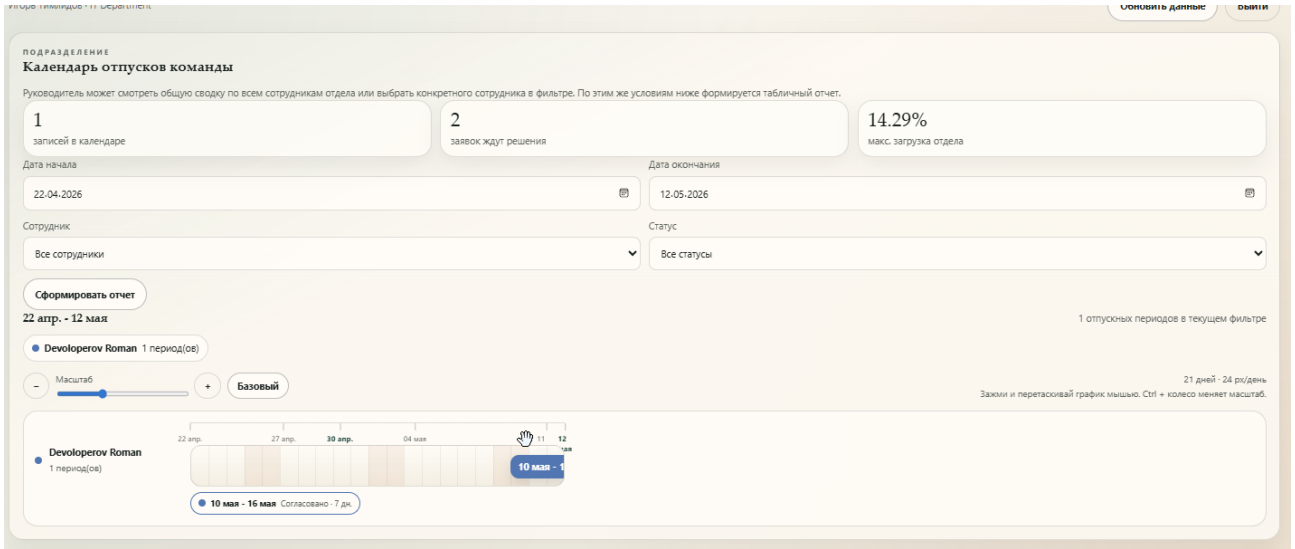

Рисунок 3 – Интерфейс функции «Календарь отпусков команды» в личном кабинете руководителя

Интерфейс остальных возможностей представлен на рисунках 4 и 5.

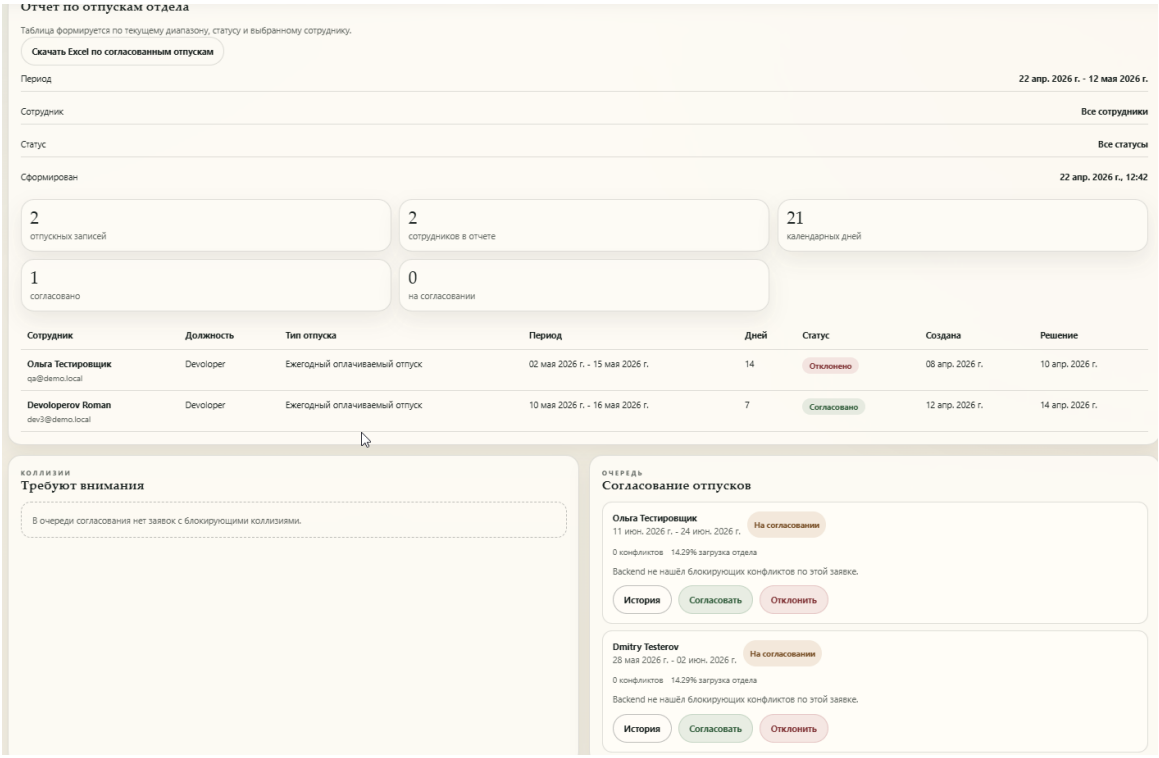

Рисунок 4 – Интерфейс функций формирования отчетов по отпускам и согласования заявок в личном кабинете руководителя

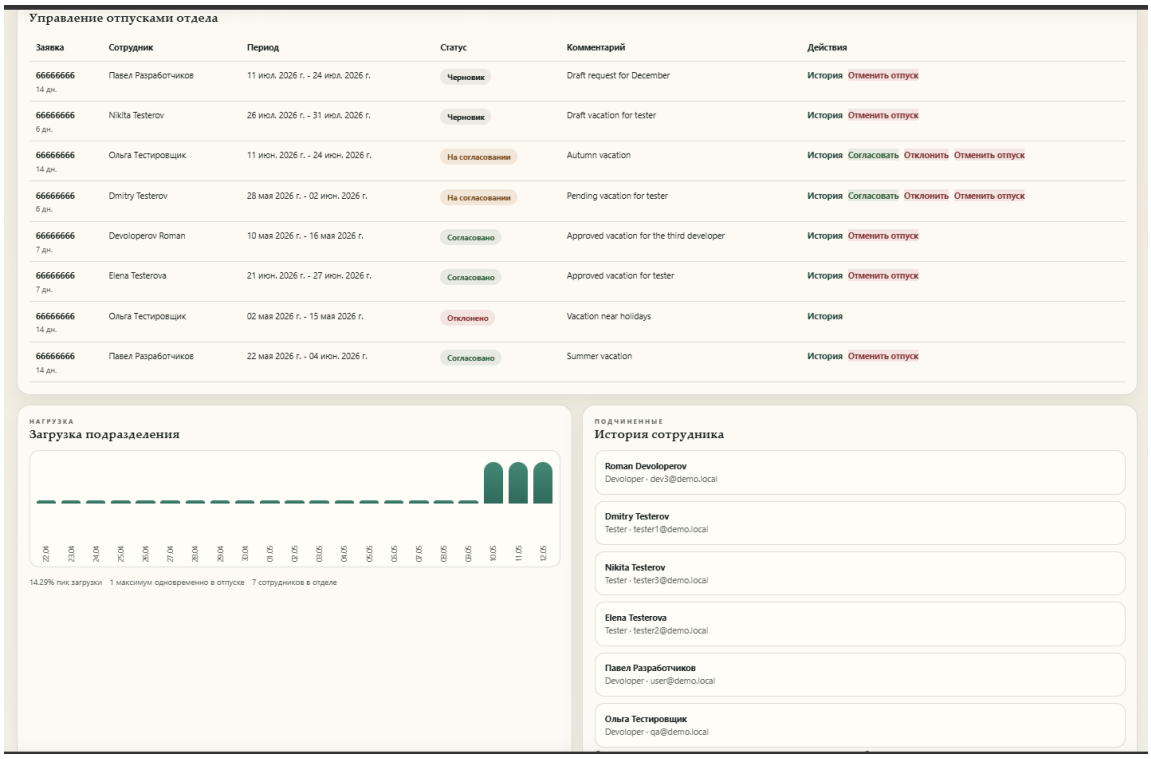

Рисунок 5 – Интерфейс функции управления отпусками и историями сотрудников в личном кабинете руководителя

Система обеспечивает автоматизацию процесса подачи и обработки заявок на отпуск. Пользователь выбирает даты отпуска, после чего система автоматически проверяет ограничения, рассчитывает доступные дни и определяет маршрут согласования. Пример интерфейса создания заявки представлен на рисунке 6.

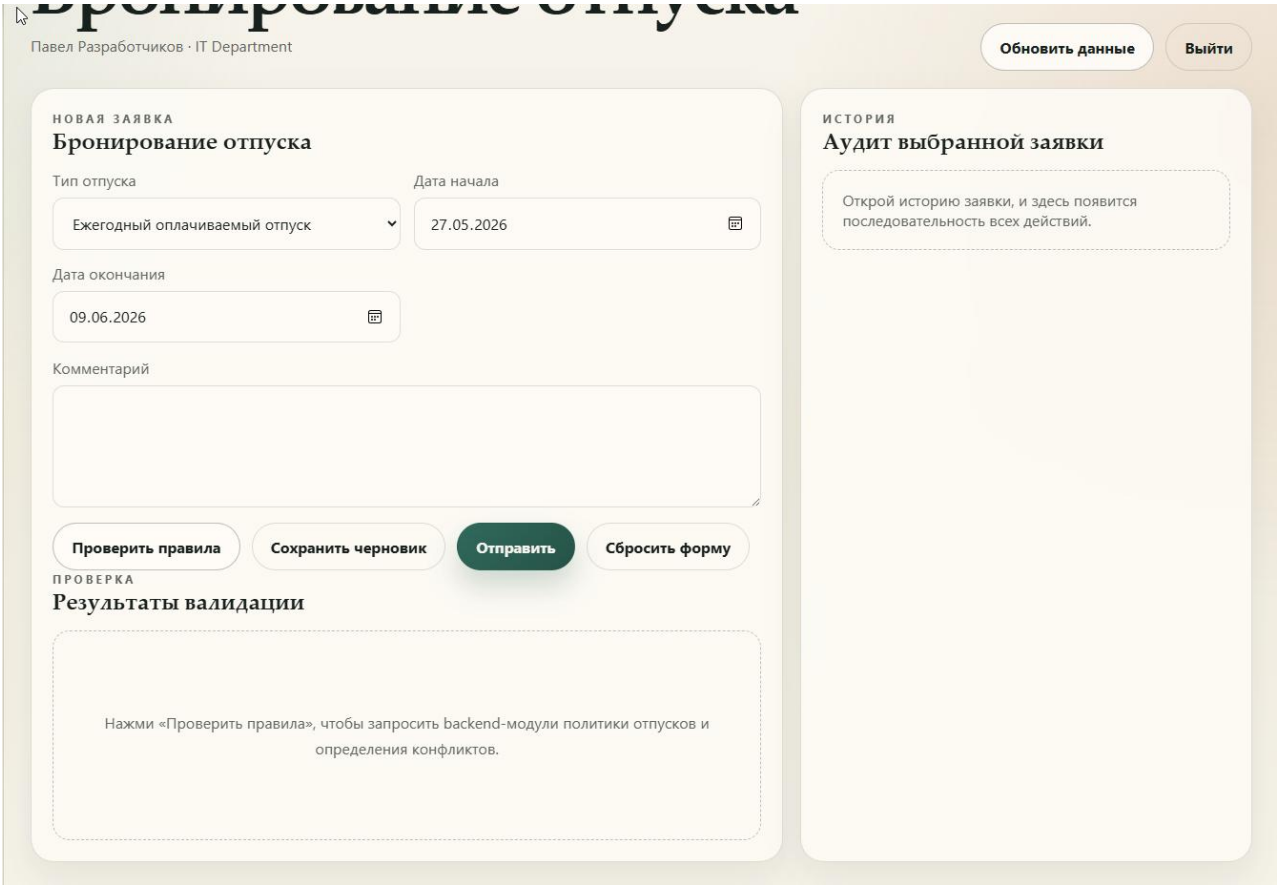

Рисунок 6 – Интерфейс создания заявки на отпуск

Одной из ключевых возможностей системы является гибкое управление отпускными политиками. Система поддерживает два режима работы организации, стандартный и круглосуточный. При выборе круглосуточного режима требуется назначение замещающего сотрудника, что обеспечивает непрерывность рабочих процессов.

HR-специалист отвечает за ведение кадровой структуры организации, управление сотрудниками, настройку отпускных политик и правил предоставления отпусков. Интерфейс управления отпускными политиками представлен на рисунке 7.

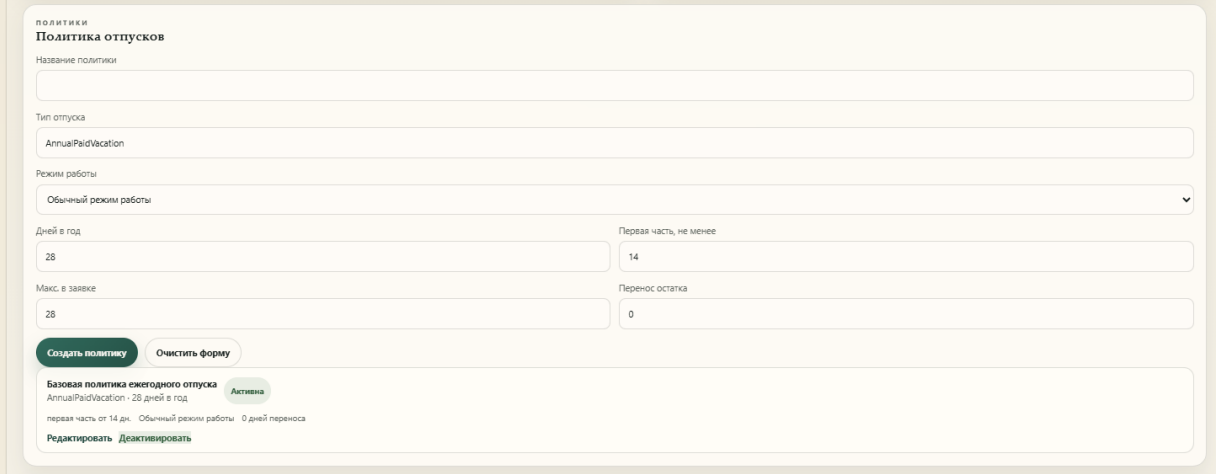

Рисунок 7 – Интерфейс управления отпускными политиками

Администратор системы выполняет функции управления пользователями, ролями и структурой организации, а также обеспечивает корректную работу системы.

Также система предоставляет инструменты аналитики и отчетности. Руководители и HR-специалисты могут формировать отчеты по отпускам сотрудников, анализировать загрузку подразделений и экспортировать данные в формате Excel. Пример формирования отчетности представлен на рисунке 8.

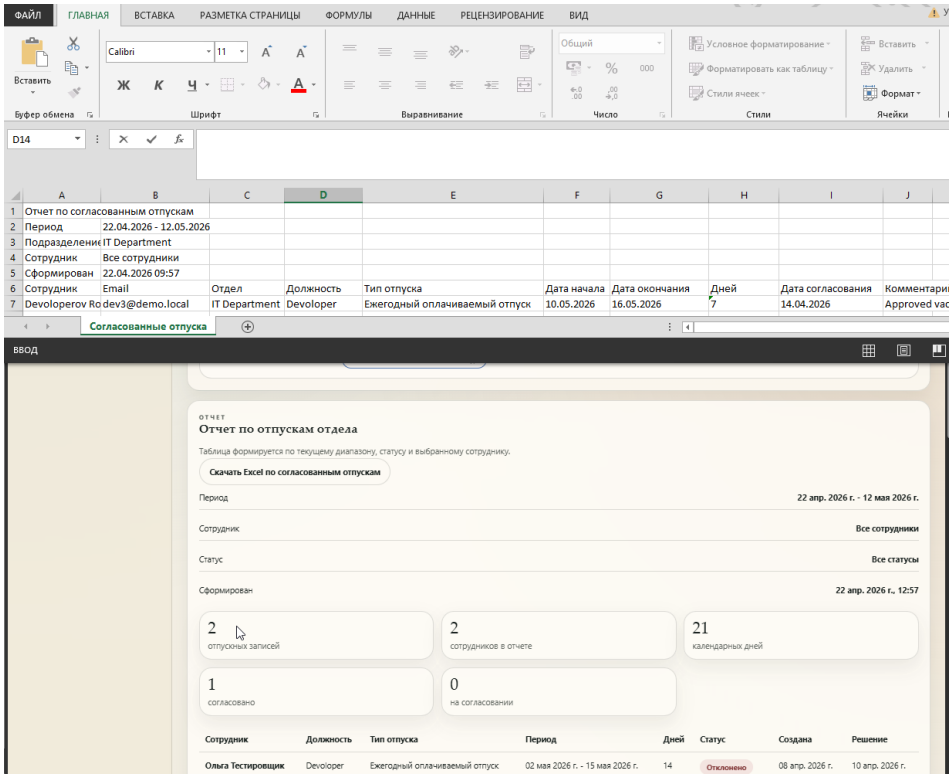

Рисунок 8 – Пример формирования отчета по отпускам

Предлагаемое решение представляет собой программный продукт, ориентированный на автоматизацию управления отпусками. Его внедрение позволит сократить временные затраты на обработку заявок, снизить количество ошибок и обеспечить гибкую настройку под особенности организации. Это делает систему универсальным решением, пригодным для использования в организациях различного масштаба и профиля деятельности.

1.3 Объем, структура и развитие рынка

Рынок HR-технологий в России является активно развивающимся сегментом IT-индустрии, демонстрирующим устойчивый рост на фоне цифровизации бизнеса и импортозамещения программного обеспечения.

Согласно оценке TAdviser объем российского рынка HR-tech в 2023 году составил около 33,6 млрд рублей, увеличившись на 15% по сравнению с предыдущим годом. Это свидетельствует о стабильном росте спроса на цифровые решения в сфере управления персоналом. Подробнее с данным исследованием можно ознакомиться в обзоре «Российский рынок HR-tech» [5].

Основываясь на данных аналитических агентств, в первом полугодии 2025 года объем рынка достиг 40,6 млрд рублей, показав рост на 12% по сравнению с аналогичным периодом предыдущего года. Это демонстрирует, что за этот период рынок продолжил рост. На рисунке 9 представлена диаграмма динамики роста рынка HR-технологий в России. Подробнее с данным исследованием можно ознакомиться в статье «Рынок HRtech вырос на 12%» [6].

Рисунок 9 – Диаграмма динамики роста рынка HR-технологий в России

На данный момент темпы роста сохраняются. Это подтверждает высокую динамику развития отрасли и наличие устойчивого спроса на решения автоматизации HR-процессов.

Структура рынка состоит из несколько ключевых сегментов:

– системы кадрового учета;
– решения для подбора персонала;
– системы управления обучением и развитием сотрудников;
– решения для автоматизации отдельных HR-процессов.

При этом наблюдается тенденция роста спроса на узкоспециализированные решения, которые позволяют автоматизировать конкретные задачи без внедрения комплексных дорогостоящих систем.

Рынок характеризуется высокой конкуренцией. На нем присутствует более 100 компаний, разрабатывающих HR-tech решения, включая как крупные платформы, так и нишевые продукты - данный показатель приведен на основе аналитических обзоров рынка HR-tech в России [7]. При этом в источниках отмечается, что значительная часть компаний демонстрирует рост выручки, что связано с расширением рынка и увеличением числа пользователей цифровых HR-систем.

Одним из ключевых факторов развития рынка является импортозамещение. Уход зарубежных решений освободил значительную долю рынка и стал стимулятором для отечественной разработки. В результате увеличился спрос на российские системы, которые могут быть адаптированы под требования законодательства и особенности бизнес-процессов.

Подводя итоги анализа рынка, можно сделать вывод, что российский рынок HR-технологий характеризуется значительным объемом, устойчивыми темпами роста и высоким уровнем конкуренции. Тенденции цифровизации и автоматизации кадровых процессов формируют благоприятные условия для вывода на рынок решения по управлению отпусками

1.4 Анализ конкурентов

На рынке программных решений для управления персоналом представлено множество различных систем, которые включает функционал управления отпусками сотрудников. На данный момент, основными конкурентами разрабатываемого продукта являются такие решения, как BambooHR, ТопФактор и 1С:Зарплата и управление персоналом.

BambooHR представляет собой зарубежную HR-платформу, ориентированную на автоматизацию кадровых процессов. К сильным сторонам системы относятся удобный пользовательский интерфейс и развитый функционал управления персоналом. Однако система ориентирована на зарубежный рынок и имеет ограниченную поддержку требований российского законодательства.

ТопФактор является российской системой управления персоналом, обеспечивающей интеграцию различных кадровых процессов. К преимуществам решения относится адаптация под российский рынок и централизованное управление данными. В то же время система обладает избыточным функционалом и не предоставляет гибкой настройки логики отпусков.

1С:Зарплата и управление персоналом – это комплексная система, ориентированная на кадровый учет и расчет заработной платы. К ее преимуществам относится полная поддержка российского законодательства и высокая точность расчетов. Однако данное решение характеризуется сложностью внедрения, низкой удобностью интерфейса и избыточным функционалом для задач управления отпусками.

Для более наглядного сравнения программных решений был проведен сравнительный анализ по ключевым критериям, отражающим потребительские свойства систем. Данный анализ отображен в таблице 1.

Таблица 1 – Сравнительный анализ программных решений

Критерий	BambooHR	ТопФактор	1С:ЗУП	Разрабатываемая система
Ориентация на рынок	Зарубежный	Российский	Российский	Российский
Удобство использования	Высокое	Среднее	Низкое	Высокое
Стоимость внедрения	Средняя	Средняя	Высокая	Низкая
Гибкость настройки	Ограниченная	Средняя	Низкая	Высокая
Поддержка сложной логики отпусков	Ограниченная	Ограниченная	Частичная	Расширенная

Проведенный анализ показывает, что существующие решения ориентированы на комплексную автоматизацию HR-процессов. Это делает их универсальными, но одновременно усложняет внедрение и использование при решении узкой задачи управления отпусками.

В отличие от конкурентов, разрабатываемая система ориентирована на решение конкретной задачи – автоматизацию управления отпусками. Основным конкурентным преимуществом является гибкая настройка отпускных политик, включая поддержку различных режимов работы организации и контроль замещения сотрудников. Система также выделяется более простой интеграцией в предприятие, удобством использования и меньшими затратами на внедрение.

Возможные конкурентные действия со стороны существующих решений связаны с расширением функциональности и адаптацией под локальные рынки.

В качестве стратегии реагирования предлагается развивать специализированный функционал с учетом потребностей российских организаций. При необходимости система может дорабатываться под конкретную компанию. Возможно добавление нового функционала или изменение логики работы под требования заказчика.

Система проста в внедрении и не требует сложной настройки. Она может быть использована в организациях различного профиля, включая предприятия с большим количеством сотрудников и круглосуточным режимом работы (производственные и военные предприятия).

Таким образом, разрабатываемый продукт обладает рядом конкурентных преимуществ и может занять нишу решений для управления отпусками сотрудников на российском рынке.

1.5 Бизнес-модель проекта

Проект предлагает программный продукт в виде облачного сервиса, работающего по модели SaaS (Software as a Service). Клиенты могут использовать систему через интернет, не требуя установки ПО на свои серверы, что упрощает доступ и снижает затраты на инфраструктуру.

Основным источником дохода является покупка подписки на использование системы. После активации лицензии клиент получает доступ к функционалу системы. Стоимость подписки может варьироваться в зависимости от количества пользователей, уровня функциональности и масштаба организации. Например, для небольших компаний может быть предложен базовый тариф, а для крупных предприятий – расширенный тариф с дополнительными возможностями и поддержкой.

Кроме того, предусмотрена возможность получения дохода за разработку дополнительного функционала по индивидуальным требованиям заказчика. Система может быть адаптирована для конкретной организации, включая модификацию логики работы, расширение модулями и, при необходимости, интеграцию с внутренними системами компании.

Основными каналами продвижения продукта являются прямые продажи организациям, размещение решения на специализированных IT-площадках и сотрудничество с интеграторами. Также для привлечения большего числа клиентов используются интернет-реклама (контекстная реклама, таргетированная реклама). Дополнительно применяется демонстрация продукта в виде презентаций и пилотных внедрений, что позволяет потенциальным клиентам оценить возможности системы на практике.

Целевая аудитория продукта включает организации с большим числом сотрудников, где важно обеспечить точное планирование отпусков и

непрерывность работы. К ним относятся коммерческие компании среднего и крупного бизнеса, государственные учреждения, образовательные организации, а также предприятия с круглосуточным режимом работы, включая производственные и военные предприятия.

Преимуществом данной бизнес-модели является возможность масштабирования. Система может внедряться как в небольших организациях, так и в крупных предприятиях с распределенной структурой. Простота подключения и гибкость настройки позволяют быстро адаптировать продукт под конкретные условия.

Таким образом, проект позволяет обеспечить стабильный поток дохода за счет подписки, а также дополнительную прибыль за счет доработки функционала. Это делает проект экономически устойчивым и перспективным для дальнейшего развития.

1.6 Команда стартап-проекта

Команда стартап-проекта состоит из двух участников, обладающих необходимыми компетенциями для разработки программного продукта и реализации всех этапов проекта.

Акчурин Али Радикович выполняет роль backend-разработчика и DevOps-инженера. Он отвечает за разработку серверной части приложения, реализацию бизнес-логики, проектирование архитектуры системы и работу с базой данных. Также в его задачи входит настройка серверной инфраструктуры и контейнеризация приложения с использованием Docker. Обладает опытом разработки на языке C# с использованием ASP.NET Core. Имеет опыт работы с СУБД PostgreSQL, применения Dapper и Entity Framework для выполнения SQL-запросов, а также разработки REST API и механизмов аутентификации. Умеет работать с брокерами сообщений и писать unit-тесты.

Васильев Дмитрий Александрович выполняет роль frontend-разработчика и дизайнера проекта. Он отвечает за разработку клиентской части приложения, создание пользовательского интерфейса и обеспечение удобства взаимодействия

с системой. В его задачи входит реализация интерфейсов личного кабинета, форм подачи заявок и визуализации графиков отпусков, а также взаимодействие с серверной частью через API. Обладает навыками разработки с использованием React и TypeScript, а также опытом проектирования пользовательских интерфейсов.

Оба участника обучаются на 4 курсе Пензенского государственного университета по направлению «Программная инженерия» и имеют опыт совместной работы над проектами. Они принимали участие в различных олимпиадах и хакатонах по программированию.

Идея разработки системы автоматизации отпусков была сформирована в рамках регионального хакатона SECON, где перед участниками стояла задача создания решения для компании «ТНС энерго», направленного на оптимизацию процессов планирования отпусков внутри компании. В рамках данного мероприятия команда разработала прототип системы и заняла 2-е место.

Полученный опыт и разработанное решение были взяты за основу текущего проекта и в дальнейшем доработаны, и расширены с учетом требований рынка и особенностей различных организаций.

Распределение ролей в команде позволяет охватить все направления в проекте: серверную часть, клиентскую часть и развертывание системы. Компетенции участников соответствуют требованиям проекта и позволяют реализовать программный продукт и обеспечить его дальнейшее развитие.

1.7 Описание финансовой модели и необходимости привлечения инвестиций

Финансовая модель проекта основана на модели подписки (SaaS), при которой пользователи оплачивают доступ к системе на регулярной основе. Основным источником дохода является ежемесячная или годовая подписка на использование программного продукта.

Планируется внедрение нескольких тарифов:

– базовый тариф для небольших организаций (до 50 пользователей);

– стандартный тариф для средних компаний;

– расширенный тариф для крупных предприятий с дополнительным функционалом и поддержкой.

Средняя стоимость подписки может составлять от 300 до 800 рублей за пользователя в месяц. Например, если подключено 10 компаний с 100 сотрудниками в каждой, ежемесячная выручка может составить около 500000 рублей. Также дополнительно предусмотрен доход от разработки индивидуального функционала под требования заказчика. В таком случае цена будет варьироваться от сложности разработки и внедрения функционала.

Расходная часть проекта включает затраты на разработку и сопровождение программного обеспечения, оплату серверной инфраструктуры и продвижение продукта. На начальном этапе основная часть разработки выполняется усилиями команды разработчиков. Это уменьшает затраты на начальную разработку. Основные расходы связаны с использованием облачных, маркетингом и прочими затратами. Примерные ежемесячные затраты на поддержание работы серверной инфраструктуры составят порядка 30000 рублей, а на маркетинг в районе 50000.

В первые полгода ожидается небольшой доход. Это связано с тестированием продукта и привлечение клиентской базы. После ввода продукта в использования, в течение 2-х лет, планируется увеличение числа клиентов и стабильный рост выручки. После того, как продукт закрепить на рынке, проект может выйти на устойчивую прибыль за счет масштабирования.

Точка безубыточности достигается при покрытии ежемесячных затрат за счет дохода от подписок. При средних затратах около 100 000 рублей в месяц, достаточно привлечь порядка 5 компаний среднего размера (около 200 пользователей в месяц), чтобы выйти по затратам в ноль.

На начальном этапе проект не требует значительных инвестиций, так как разработка выполняется силами команды. Однако для ускорения развития проекта может потребоваться финансирование в размере от 500000 до 1000000 рублей, которое будет направлено на:

– продвижение продукта;
– доработку функционала;
– увеличение размера команды разработки;
– масштабирование инфраструктуры.

Инвестиционная привлекательность проекта обусловлена ростом рынка HR-технологий, востребованностью автоматизации кадровых процессов и наличием конкурентных преимуществ разрабатываемой системы. Разрабатываемую систему, на фоне конкурентов, выделяет возможность гибкой настройки отпускных политик, поддержка предприятий с круглосуточным режимом работы, простота внедрения и возможность быстрой адаптации под требования конкретных организаций.

Таким образом, финансовая модель проекта является устойчивой и ориентирована на постепенное масштабирование. Низкий порог входа и возможность быстрого внедрения позволяют привлекать клиентов на ранних этапах, а подписочная модель обеспечивает стабильный доход. Проект имеет потенциал для дальнейшего роста и может быть успешно реализован на российском рынке.

1.8 Описание развития бизнес-проекта

На текущий момент разработан прототип программного продукта, включающий серверную часть и клиентскую части. Основные функции системы контроля отпусков на данный момент:

– массовая и одиночная регистрация пользователей;
– управление заявками отпусков;
– настройка отпускных политик;
– логика согласования;
– система уведомлений;
– формирование отчетов.

Продукт протестирован в рамках практических задач по согласованию отпусков. На данном этапе отсутствует полноценная веб-платформа для продажи

и распространения продукта, однако сама система полностью работоспособна и готова к дальнейшему развитию и внедрению.

Дальнейшее развитие проекта планируется поэтапно. Сначала предполагается создание публичного сайта для предоставления продукта, подключения клиентов, а также реализация системы подписки. Далее планируется расширение функциональности системы, включая улучшение интерфейса, а также добавление более подробной аналитики и отчетности. Затем будет рассматриваться интеграция с внешними сервисами компаний, а также развитие мобильной версии продукта.

Ключевым преимуществом проекта является используемая архитектура модульного монолита. Такой подход позволяет легко добавлять новый функционал без полной переработки системы. Каждый модуль отвечает за свою область (пользователи, заявки, политики). Выбранная архитектура упрощает поддержку и развитие продукта. При необходимости система может быть масштабирована и переведена без каких-либо затруднений на микросервисную архитектуру, что позволит повысить устойчивость при росте нагрузки.

К основным целевым показателям проекта относятся увеличение количества пользователей и рост числа подключенных организаций. На начальном этапе планируется привлечение первых клиентов и тестирование продукта в рамках настоящего производства. Далее предполагается постепенное расширение клиентской базы и выход на устойчивый уровень использования системы в большом числе предприятий.

Прогнозируемая динамика развития проекта предполагает постепенный рост. В первые месяцы основной акцент делается на доработке продукта и привлечении новых пользователей. Затем ожидается увеличение числа клиентов за счет появления отзывов на продукт, продвижения и расширения функциональности. В долгосрочной перспективе проект может масштабироваться за счет выхода на новые сегменты рынка и адаптации системы под различные типы организаций.

Подводя итог, проект обладает высоким потенциалом развития. Наличие готовой серверной и клиентской части, гибкой архитектуры и возможности масштабирования позволяет рассчитывать на успешное внедрение продукта на рынок, с возможностью дальнейшего улучшения проекта.

1.9 Описание рисков

В процессе реализации бизнес-проекта были выявлены основные группы рисков, способные повлиять на разработку и внедрение системы. Для каждого риска определены вероятность возникновения, степень влияния и меры по снижению. Анализ рисков проекта приведен в таблице 2.

Таблица 2 – Анализ рисков проекта

Вид риска	Описание риска	Вероятность	Влияние	Меры минимизации
Рыночный	Высокая конкуренция, нежелание компаний внедрять новое ПО	Средняя	Высокое	Узкая специализация продукта, низкая стоимость, адаптация под РФ
Технологический	Ошибки в системе, сбои, рост нагрузки	Средняя	Высокое	Использование проверенных технологий, тестирование, масштабирование
Операционный	Срывы сроков, сложности внедрения	Средняя	Среднее	Agile-подход, поэтапная разработка, тестирование
Юридический	Несоблюдение требований законодательства РФ	Низкая	Высокое	Соблюдение требований, защита персональных данных
Финансовый	Недостаток средств на развитие и продвижение	Средняя	Среднее	Подписочная модель, постепенное развитие
Кадровый	Недостаток специалистов при росте проекта	Средняя	Среднее	Расширение команды, привлечение специалистов

После проведения анализа можно сделать вывод, что наиболее значимыми являются рыночные и технологические риски, так как они напрямую влияют на успешность внедрения продукта и его стабильную работу.

Для снижения рисков в проекте используются современные технологии разработки, гибкая методология и поэтапное развитие системы. Также предусмотрена возможность масштабирования продукта и его адаптация под требования заказчиков.

Так, проведенный анализ показывает, что команда проекта учитывает потенциальные угрозы и готова к их минимизации, что повышает устойчивость и перспективность стартап-проекта.

2 Анализ предметной области и постановка задачи на разработку средств автоматизации контроля отпусков

2.1 Анализ предметной области

Предметная область работы связана с планированием, учетом, согласованием и контролем отпусков сотрудников организации. В большинстве компаний данные процессы выполняются вручную или с использованием таких средств, как электронные таблицы или почта. Зачастую это приводит к ошибкам, конфликтам в графиках и усложняет процесс согласования. Наиболее данная проблема становится актуальной в организациях с большим числом сотрудников, где ручное управление отпусками становится неэффективным.

Разрабатываемая система направлена на автоматизацию этих процессов и обеспечивает централизованное хранение данных о сотрудниках, организационной структуре, отпускных политиках и заявках на отпуск. Система позволяет автоматизировать подачу и согласование заявок, контролировать остаток отпускных дней, проверять ограничения и выявлять конфликты при планировании. Система разрабатывается с учетом требований действующего трудового законодательства Российской Федерации, в частности норм Трудового кодекса РФ, регламентирующих порядок предоставления отпусков, их продолжительность и правила учета.

При планировании отпусков необходимо учитывать несколько факторов:

- структуру организации;
- наличие руководителя; правила отпускной политики;
- доступные дни; загрузку подразделения;
- возможные пересечения с отпусками других сотрудников.

В рамках данной работы разрабатывается серверная часть системы, которая отвечает за реализацию бизнес-логики, хранение и обработку данных, управление доступом пользователей к определенному функционалу, в зависимости от его роль, а также обработку заявок на отпуск.

Особое внимание уделяется гибкому управлению отпускными политиками, позволяющему изменять правила предоставления отпусков в организации. В системе предусмотрены два режима работы: обычный и круглосуточный. В круглосуточном режиме при оформлении отпуска требуется наличие сотрудника на замену с аналогичной должностью. Такой подход особого актуален на предприятиях военного назначения, где станок должен работать безостановочно. В обычном режиме допускается оформление отпуска без обязательной замены, даже при наличии пересечений по датам.
Серверная часть предоставляет API для взаимодействия с клиентским приложением и обеспечивает корректность работы всей системы.

К основным функциям системы относятся:

- регистрация и авторизация пользователей;
- управление пользователями и организационной структурой;
- создание и обработка заявок на отпуск, их согласование и контроль;
- настройка отпускных политик;
- расчет доступных дней, выявление конфликтов;
- формирование отчетности;
- импорт данных сотрудников.

Система ориентирована на сотрудников, руководителей, HR-специалистов и администраторов.

Таким образом, в ходе анализа предметной области, можно сделать вывод, что автоматизация управления отпусками позволяет повысить прозрачность кадровых процессов, снизить количество возникаемых ошибок и, как следствие, увеличить эффективность работы компании. Полученные результаты анализа будут применены для дальнейшего проектирования и реализации серверной части информационной системы в рамках бизнес-проекта.

2.2 Анализ программных решений аналогичного назначения

В рамках разработки информационной системы контроля отпусков был проведен анализ существующих программных решений аналогичного

назначения. Далее будут рассмотрены популярные решения и проведен сравнительный анализ конкурентов с разрабатываемым приложением.

2.2.1 Популярные программные решения управления отпусками

BambooHR – это американская система управления персоналом, ориентированная на автоматизацию кадровых процессов в организациях малого и среднего бизнеса. Данное решение предоставляет большой функционал для ведения кадрового учета и управление отпусками персонала.

BambooHR дает возможность сотрудникам в личном кабинете оставлять заявки на отпуск, а руководителям согласовывать их в цифровом виде. Система автоматически отслеживает отпускных дней и хранит историю заявок.

К особенностям BambooHR относится удобный пользовательский интерфейс. Интерфейс личного кабинета представлен на рисунке 10 [8]. Система обеспечивает централизованное хранение данных и доступ к ним из любого устройства.

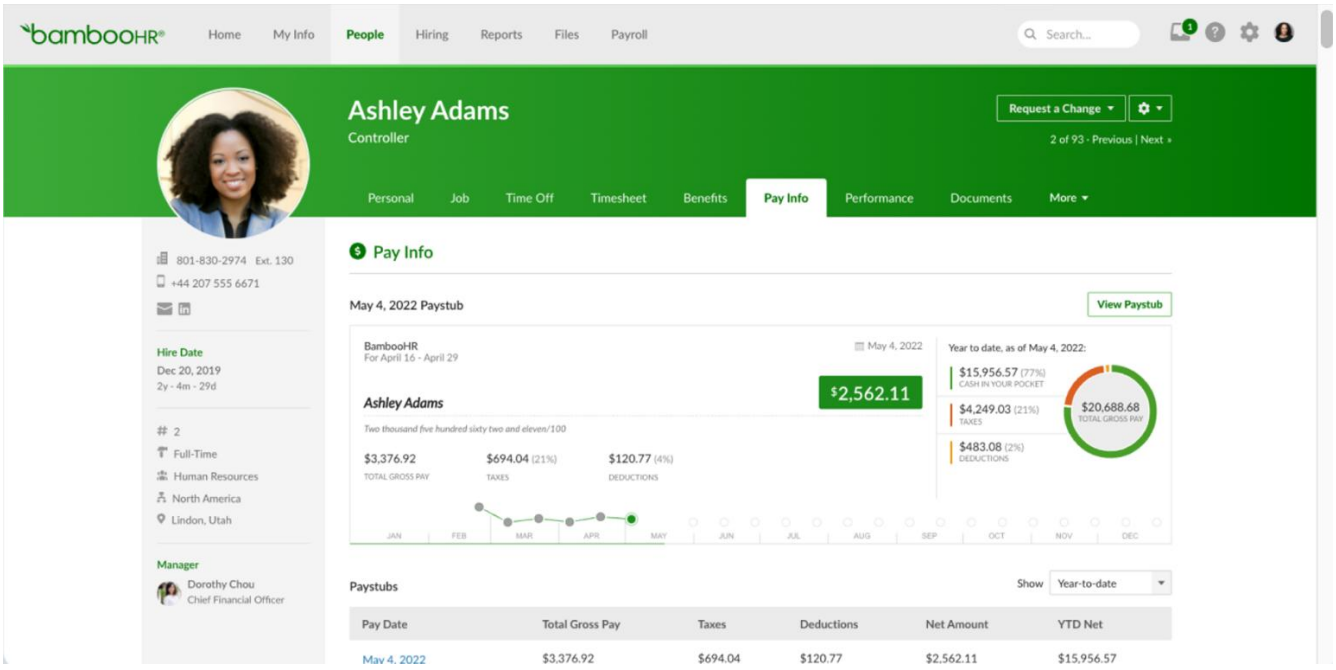

Рисунок 10 – Интерфейс программного обеспечения «BambooHR»

ТопФактор – российская информационная система управления персоналом, предназначенная для автоматизации кадровых процессов, включая учет рабочего времени и управление отпусками сотрудников.

Ключевой особенностью ТопФактор является его направленность на интеграцию кадровых процессов в единую цифровую среду. Система позволяет централизованно управлять информацией о сотрудниках и обеспечивает прозрачность взаимодействия между сотрудниками, руководителями и HR-службой. Интерфейс просмотра графика отпусков представлен на рисунке 11 [9].

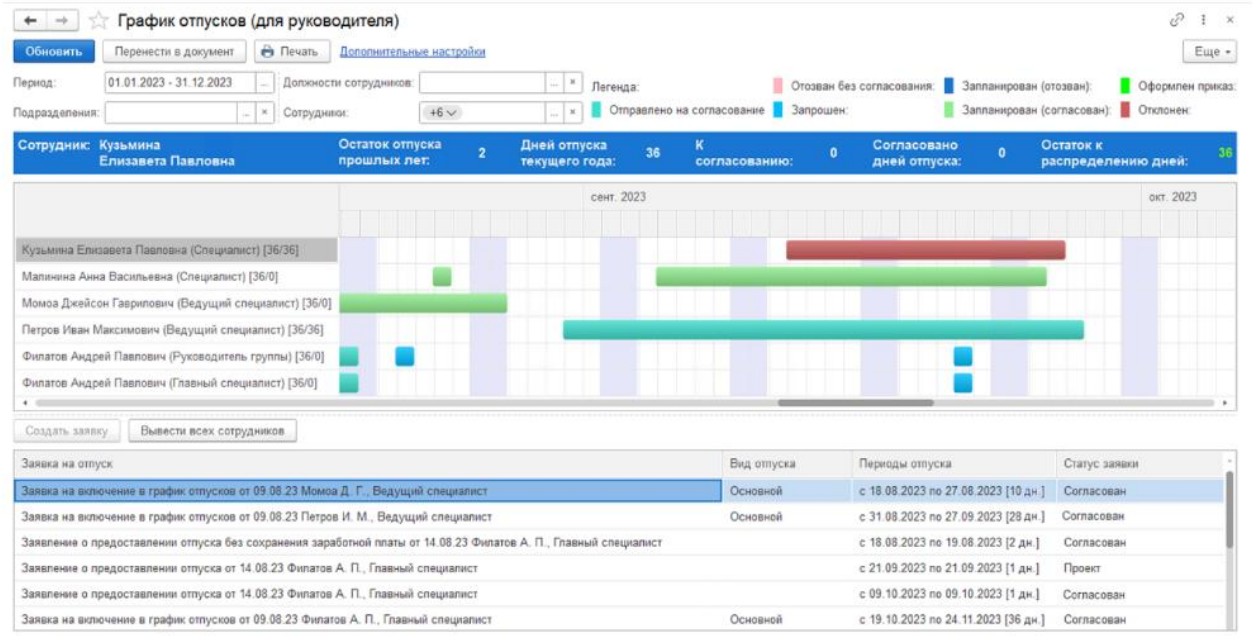

Рисунок 11 – Интерфейс программного обеспечения «ТопФактор»

1С:Зарплата и управление персоналом является комплексной системой, ориентированной на автоматизацию кадрового учета и расчета заработной платы с учетом требований законодательства. Управление отпусками в данной системе рассматривается как часть общего кадрового учета.

Главной особенностью решения является его глубокая интеграция с бухгалтерскими и расчетными процессами. Система обеспечивает точный учет отпускных дней, автоматический расчет выплат и формирование регламентированной отчетности.

Интерфейс системы представлен на рисунке 12 [10]. Решение обеспечивает централизованное хранение кадровых данных, их обработку и интеграцию с расчетными и бухгалтерскими процессами организации.

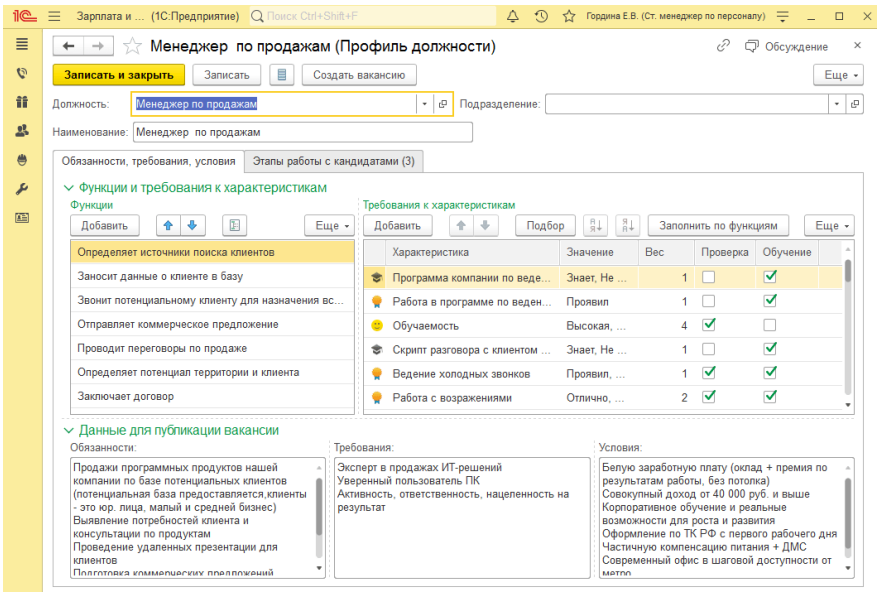

Рисунок 12 – Интерфейс программного обеспечения «1С:Зарплата и управление персоналом»

В данном разделе были рассмотрены существующие программные решения для управления отпусками сотрудников и их функциональные возможности. Далее будет проведен сравнительный анализ рассмотренных систем и разрабатываемого решения с целью выявления их преимуществ недостатков. Это послужит основой для последующего выбора архитектурных и функциональных особенностей проектируемой системы.

2.2.2 Сравнительный анализ программных обеспечений

На основе рассмотренных в предыдущем разделе решений был проведен сравнительный анализ систем, для выявления преимуществ и недостатков каждого программного обеспечения. Особое внимание уделяется удобству использования, затратам на внедрение, ориентированность на российский рынок, гибкость настройки отпускных политик для легко внедрения продукта в различные компании и наличие избыточного функционала.

Результаты сравнительного анализа представлены в таблице 3.

Таблица 3 – Сравнительный анализ программных решений для управления отпусками

Критерий	BambooHR	ТопФактор	1С:ЗУП	Разрабатываемая система
Ориентация на рынок	Зарубежный	Российский	Российский	Российский
Основное назначение	HR-платформа	HR-платформа	Кадровый учет и бухгалтерия	Управление отпусками
Удобство использования	Высокое	Среднее	Низкое	Высокое
Стоимость внедрения	Средняя	Средняя	Высокая	Низкая
Избыточность функционала	Высокая	Высокая	Очень высокая	Низкая
Гибкость настройки политик	Ограниченная	Средняя	Низкая	Высокая
Поддержка российского законодательства	Ограниченная	Полная	Полная	Полная
Поддержка и сопровождение	Высокая	Средняя	Высокая	Средняя
Поддержка сложной логики отпусков	Ограниченная	Ограниченная	Частично	Расширенная
Круглосуточный режим работы	Нет	Нет	Частично	Полная
Контроль замещения сотрудников	Нет	Нет	Есть	Есть
Простота внедрения	Средняя	Средняя	Низкая	Высокая
Масштабируемость	Высокая	Высокая	Высокая	Высокая

Сравнение показывает, что существующие решения в большинстве случаев ориентированы на комплексную автоматизацию HR-процессов, что делает их более универсальными, но в то же время избыточными для задач

управления отпусками. Такие системы эффективны в крупных организациях, где требуется полный кадровый учет и интеграция с бухгалтерией.

Выбор конкретного программного решения будет зависеть от сценария использования. Комплексные HR-системы, такие как 1С:ЗУП, подходят для организаций с необходимостью полного кадрового и бухгалтерского учета. Универсальные HR-платформы (BambooHR, ТопФактор) целесообразны при необходимости интеграции различных кадровых процессов в единую систему.

В свою очередь, разрабатываемое решение для управления отпусками позволяет снизить сложность внедрения и повысить удобство. Основной отличительной особенностью разрабатываемого приложения является гибкая система управления отпускными политиками с поддержкой различных режимов работы организации. Имеется возможность выбора между обычным и круглосуточным режимами, при котором учитывается необходимость обязательного замещения сотрудника на период его отсутствия.

Таким образом, разрабатываемая система ориентирована на решение конкретной задачи управления отпусками, сочетая удобство использования, гибкость настройки и соответствие требованиям российского рынка, что позволяет выделить ее на фоне остальных решений, представленных на рынке.

2.3 Анализ целевой аудитории программного продукта

Целевая аудитория разрабатываемого приложения включает организации с большим числом сотрудников, где процессы планирования и согласования отпусков требуют высокой точности и автоматизации. В таких организациях ручное управление отпусками неэффективно и может привести к нарушению рабочего процесса. Особенно это заметно на предприятиях с непрерывной деятельностью.

Разрабатываемая система ориентирована на следующие категории клиентов:

– коммерческие предприятия среднего и крупного бизнеса (производство, энергетика, логистика, ИТ-компании), где требуется управление графиками большого количества сотрудников;

– предприятия с непрерывным циклом работы (производственные и военные организации), где необходимо учитывать круглосуточный режим и обязательное замещение сотрудников на период отпуска;

– государственные и муниципальные учреждения;

– образовательные организации, в которых планирование отпусков связано со сложным распределением нагрузки;

– холдинги и крупные корпоративные структуры с распределенной организационной структурой.

В рамках бизнес-проекта предполагается реализация двух моделей распространения программного продукта.

B2B и B2G – внедрение системы в коммерческих компаниях, государственных и муниципальных организациях [11]. Основной акцент делается на предоставлении решения «под ключ», настройке под структуру предприятия и последующей поддержке по модели SaaS.

B2C – предоставление упрощенной версии продукта для индивидуальных пользователей или небольших команд с базовым функционалом планирования отпусков [11]. Доступ к системе осуществляется по подписке с минимальными требованиями к инфраструктуре.

Таким образом, разрабатываемый продукт ориентирован на организации, в которых критически важно обеспечить непрерывность работы, прозрачность кадровых процессов и эффективное управление отпусками сотрудников.

2.4 Анализ требований

В рамках разработки был проведен анализ требований к разрабатываемому программному продукту. В данном разделе рассматриваются основные функциональные и нефункциональные требования, предъявляемые к системе, а также особенности их реализации в рамках серверной части приложения.

Далее будут рассмотрены функциональные возможности системы, а также требования к архитектуре и используемым технологиям.

2.4.1 Анализ функциональных требований

В техническом задании на разработку информационной системы контроля отпусков определены следующие функциональные требования:

– система должна обеспечивать регистрацию и авторизацию пользователей;

– предоставление пользователям информации о своем профиле и остатке отпускных дней;

– возможность создания, редактирования и отправки заявок на отпуск;

– согласование, отклонение и отмена заявок на отпуск;

– управление кадровой структурой организации;

– настройка отпускных политик;

– формирование отчетности и контроль загрузки подразделений.

Диаграммы вариантов использования описывают, какая функциональность должна быть реализована в системе и какие действия она должна выполнять [12]. Они представляют систему в виде актеров, взаимодействующих с системой через варианты использования.

Актер – это внешняя по отношению к системе сущность, которая использует функциональные возможности системы для выполнение определенных задача [12]. Вариант использования описывает сервисы, предоставляемые системой актеру. Между элементами диаграммы могут существовать различные виды связей, такие как ассоциации, включения, расширения и обобщения, отражающие зависимости и взаимодействие между сценариями использования.

В рассматриваемой системе выделяются следующие основные роли пользователей:

– пользователь системы – роль, обеспечивающая доступ к общим функциям, таким как авторизация, просмотр профиля и уведомлений;

– сотрудник – пользователь, который выполняет операции, связанные с оформлением заявок на отпуск и отправкой их на согласование своему руководителю;

– руководитель подразделения – пользователь, обладающий функционалом сотрудника, но с возможностью согласования заявок подчиненных, контроля загрузки подразделения, анализа конфликтов и формирования отчетности;

– HR-специалист – пользователь, отвечающий за ведение кадровой структуры, управление сотрудниками и настройку отпускных политик;

– администратор – пользователь, осуществляющий управление пользователями, ролями, организационной структурой и политикой компании.

Стоит отметить, что роль руководителя не является отдельной ролью внутри система, а определяется положением сотрудника внутри организации.

На рисунке 13 представлена диаграмма вариантов использования для пользователя, сотрудника и руководителя подразделения.

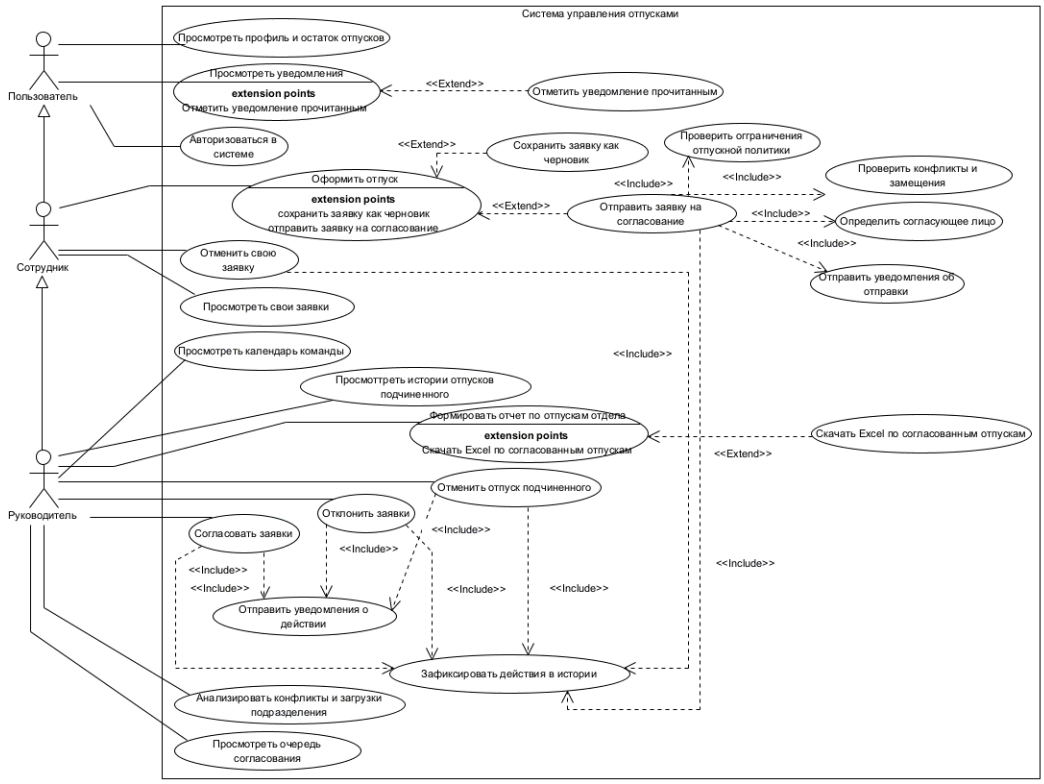

Рисунок 13 – Диаграмма вариантов использования, отражающая возможности ролей пользователя, сотрудника и руководителя

На рисунке 14 показана диаграмма вариантов использования для HR-специалиста и администратора.

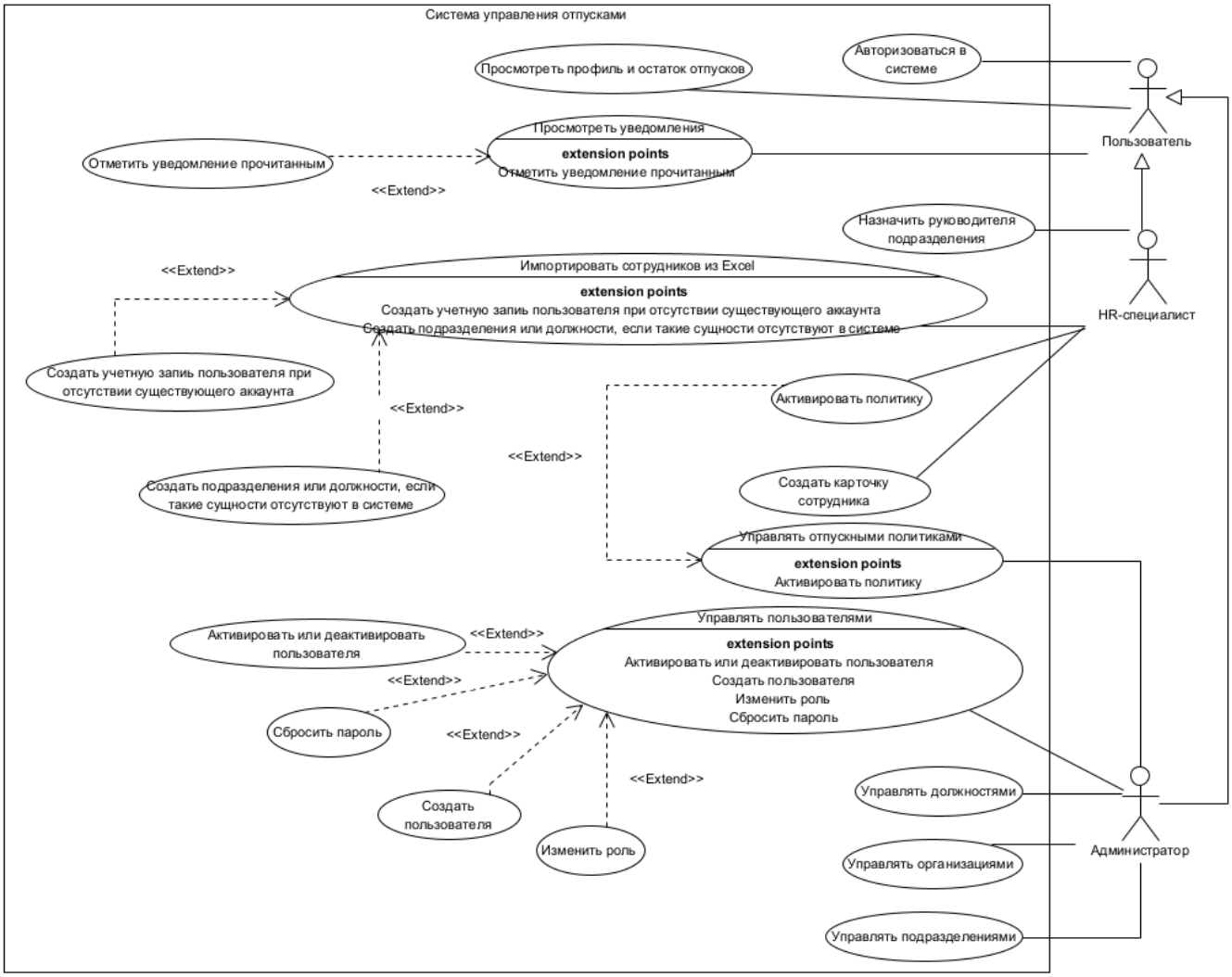

Рисунок 14 – Диаграмма вариантов использования, отражающая возможности ролей HR-специалиста и администратора

Спецификации основных вариантов использования системы управления отпусками для каждой роли приведены в приложении А.

В таблицах А.1 – А.3 приведена спецификация сценариев использования для роли «Пользователь», которая является родительской сущностью для остальных актеров, представленных на обеих диаграммах.

В таблицах А.4 – А.6 приведена спецификация вариантов использования для роли «Сотрудник».

Спецификации основных сценариев использования для роли «Руководитель» приведены в таблицах А.7 – А.9.

Помимо рассмотренных основных вариантов использования, руководитель подразделения обладает дополнительным набором возможностей, связанных с управлением отпусками сотрудников и анализом деятельности подразделения. К таким функциям относятся отмена отпусков подчиненных, формирование отчетности и анализ загрузки сотрудников.

В таблицах А.10 – А.12 приведены спецификации вариантов использования для роли «Руководитель», связанных с управлением отпусками сотрудников и анализом информации по своему отделу.

Руководитель может просмотреть очереди заявок на согласование и истории отпусков конкретного сотрудника. Также у него есть возможность выгрузки данных в формате Excel.

Спецификации оставшихся сценариев использования для роли «Руководитель» представлены в таблицах А.13 – А.15.

В таблицах А.16 – А.18 представлены спецификации вариантов использования для роли HR-специалиста.

После рассмотрения вариантов использования для роли HR-специалиста перейдем к анализу функционала администратора. В таблицах А.19 – А.21 представлены основные варианты использования администратора, связанные с управлением пользователями и структурой системы.

Администратору также доступны функции управления системы и поддержание ее работоспособности. В таблицах А.22 – А.24 представлены спецификации оставшихся вариантов использования администратора, отражающие операции по сбросу пароля, изменению статуса пользователя и управлению организационной структурой.

Таким образом, система должна обеспечивать централизованное управление отпусками, гибкость настройки бизнес-правил и разграничение прав доступа пользователей. Результаты полученные, в ходе анализа функциональных требований послужат основой для дальнейшего

проектирования системы и реализации серверной части продукта в рамках бизнес-проекта.

2.4.2 Анализ нефункциональных требований

На данном этапе были определены нефункциональные требования к разрабатываемому программному продукту.

Серверная часть приложения реализуется с использованием языка программирования C# на платформе ASP.NET Core. Разработка ведется в среде JetBrains Rider 2025. Данная платформа поддерживает кроссплатформенную разработку и обеспечивает стабильную работу приложения под управлением операционной системы Windows 10 и выше.

В качестве системы управления базами данных используется PostgreSQL 17. Ее преимуществами является удобство масштабирование и надежность хранения данных. Для работы с базой данных применяется ORM Dapper, позволяющая выполнять SQL-запросы с минимальными накладными расходами. Такой подход обеспечит высокую скорость обработки данных по сравнению с другими ORM-решениями.

Архитектура системы построена по принципу модульного монолита. Система разделена на логические модули, такие как пользователи, сотрудники, заявки на отпуск и отпускные политики. Каждый модуль отвечает за свою область функциональности. Такой подход позволяет упростить разработку и снизить затраты на внедрение системы.

Модульный монолит легче поддерживать и изменять, так как вся система находится в одном приложении. Внесение изменений не требует сложной настройки взаимодействия между сервисами. Преимуществом данной архитектуры является возможность легкого перехода к микросервисной модели. Отдельные модули могут быть выделены в самостоятельные сервисы без полной переработки системы.

Взаимодействие клиентской части с сервером осуществляется через REST API с использованием протокола HTTP и формата данных JSON. Вся бизнес-

логика реализована на сервере, что обеспечивает централизованную обработку данных и повышает безопасность приложения Клиентская часть выступает в роли потребителя и не содержит бизнес-логики.

Система должна обеспечивать надежность хранения и обработки данных с соблюдением принципов транзакционности ACID (атомарность, согласованность, изолированность и долговечность). Это гарантирует целостность данных при работе с базой данных

Для обеспечения безопасности применяется механизм аутентификации и авторизации на основе JWT-токенов. Такой подход позволяет разграничить доступ пользователей к функционалу продукта и защитить систему от несанкционированного доступа.

Также система должна обеспечивать устойчивую работу при высокой нагрузке и поддерживать возможность масштабирования при увеличении количества пользователей и объема обрабатываемых данных.

2.5 Планирование разработки и оценка бюджета

В рамках разработки серверной части информационной системы контроля отпусков был сформирован перечень задач, необходимых для реализации программного продукта. Разработка серверной части включает проектирование архитектуры системы, разработку базы данных, реализацию бизнес-логики, настройку механизмов аутентификации и авторизации, обработку запросов через API, а также тестирование и оптимизацию работы приложения.

При планировании разработки и оценке бюджета важно учесть последовательность выполнения работ, их длительность, а также распределение ресурсов между этапами разработки.

На рисунке 15 представлен перечень задач разработки серверной части информационной системы с указанием этапов, подзадач и исполнителей.

	Задача	Исполнитель	Статус
1	**Анализ и проектирование**		
1.1	Анализ требований	Backend Разработчик	Открыт
1.2	Проектирование архитектуры	Архитектор	Открыт
2	**Проектирование базы данных**		
2.1	Проектирование структуры БД	Backend Разработчик	Открыт
2.2	Связи и ограничения	Backend Разработчик	Открыт
3	**Настройка среды разработки**		
3.1	Настройка проекта ASP.NET	Backend Разработчик	Открыт
3.2	Подключение PostgreSQL и Dapper	Backend Разработчик	Открыт
4	**Модуль аунтефикации**		
4.1	Реализация регистрации и авторизации	Backend Разработчик	Открыт
4.2	JWT-токены	Backend Разработчик	Открыт
5	**Модуль пользователей и ролей**		
5.1	CRUD пользователей	Backend Разработчик	Открыт
5.2	Роли и права доступа	Backend Разработчик	Открыт
6	**Модуль сотрудников и структуры**		
6.1	Сотрудники	Backend Разработчик	Открыт
6.2	Подразделения и должности	Backend Разработчик	Открыт
7	**Модуль отпускных политик**		
7.1	Создание и настройка политик	Backend Разработчик	Открыт
7.2	Активация и настройки	Backend Разработчик	Открыт
8	**Модуль отпускных заявок**		
8.1	Создание и редактирование заявки	Backend Разработчик	Открыт
8.2	Согласование заявок	Backend Разработчик	Открыт
9	**Уведомления**		
9.1	Реализация уведомлений	Backend Разработчик	Открыт
10	**Отчетность и экспорт**		
10.1	Формирование отчетов	Backend Разработчик	Открыт
10.2	Экспорт в Excel	Backend Разработчик	Открыт
11	**Тестирование**		
11.1	Тестирование API	Тестировщик	Открыт
11.2	Исправление ошибок	Backend Разработчик	Открыт
12	**Завершение разработки**		
12.1	Оптимизация	Backend Разработчик	Открыт
12.2	Подготовка дкоументации	Backend Разработчик	Открыт

Рисунок 15 – Список задач на разработку серверной части системы контроля отпусков

На основе сформированного перечня задач была построена диаграмма Ганта, наглядно отображающая процесс разработки серверной части системы.

Диаграмма Ганта представляет собой инструмент планирования, который используется для визуализации последовательности выполнения задач, их длительности и взаимосвязей [13]. Она позволяет определить сроки выполнения отдельных этапов разработки и эффективно распределить ресурсы проекта.

Диаграмма Ганта для системы контроля отпусков, представлена на рисунке 16. Она отражает этапы разработки серверной части информационной системы контроля отпусков с указанием временных интервалов выполнения задач.

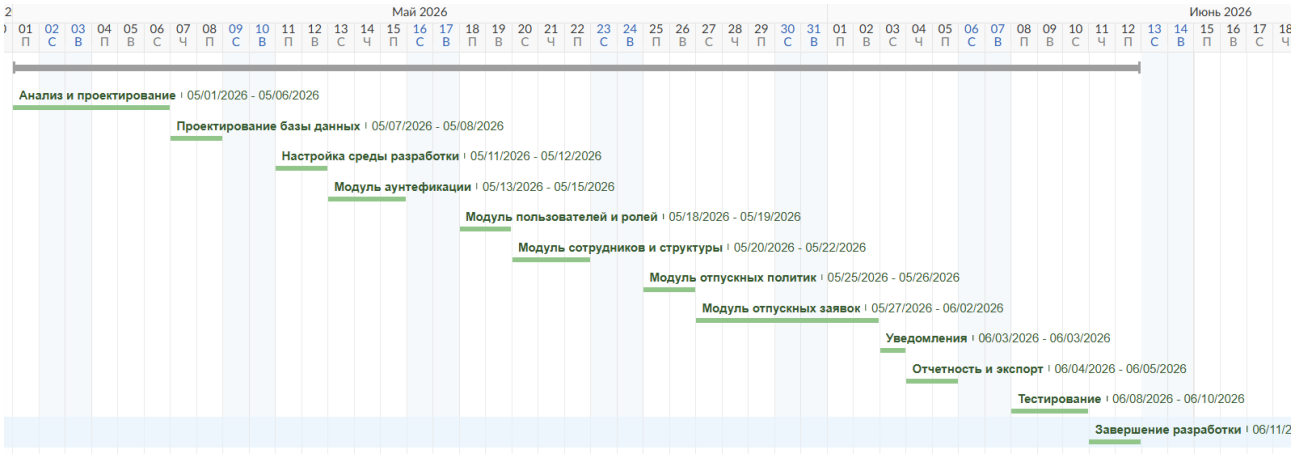

Рисунок 16 – Диаграмма Ганта системы контроля отпусков

На начальных этапах работы выполняются анализ и проектирование, после чего осуществляется разработка основных модулей системы. Наиболее трудоемкими этапами являются реализация модулей сотрудников, отпускных заявок и бизнес-логики. Завершающими этапами являются тестирование, оптимизация и подготовка документации.

После формирования диаграммы Ганта была проведена оценка трудозатрат на разработку серверной части системы.

На рисунке 17 представлена сводка по количеству затраченных часов для каждого исполнителя, участвующего в разработке.

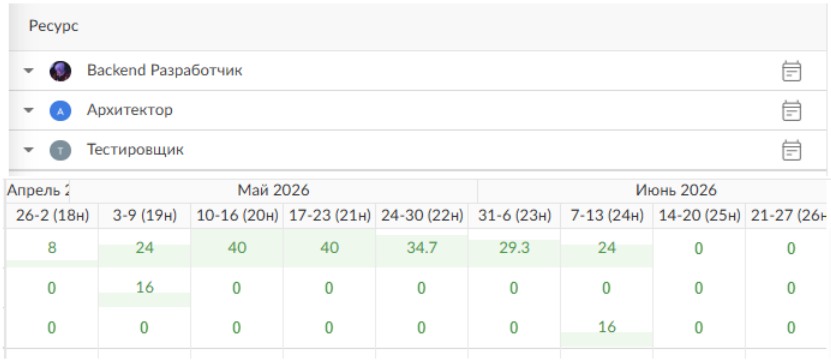

Рисунок 17 – Количество затраченных часов на разработку

Полученные данные позволяют определить общий объем трудозатрат и перейти к расчету стоимости разработки программного продукта. Приблизительный расчет разработки серверной части системы контроля отпусков приведен в таблице 4.

Таблица 4 – Расчет затрат на разработку

Исполнитель	Макс. часы	Обычная ставка (руб./час)	Сверхурочная ставка (руб./час)	Общая стоимость (руб.)
Backend-разработчик	210	1200	1500	252 000
Архитектор	16	1500	1800	24 000
Тестировщик	16	800	1000	12 800
Итого	242	-	-	288 800

Основной объем работ выполняется backend-разработчиком, на которого приходится большая часть трудозатрат. Архитектор задействован на начальных этапах проектирования, а тестировщик на этапе проверки и валидации системы.

С учетом средних ставок на рынке разработки программного обеспечения, ориентировочная стоимость разработки серверной части составит около 300000 рублей.

Таким образом, разработка серверной части системы контроля отпусков не требует значительных финансовых затрат и большого количества времени, что делает данный проект целесообразным и выгодным для внедрения в организациях различного масштаба.

3 Проектирование серверной части приложения системы контроля отпусков

3.1 Выбор технологий разработки

В ходе проектирование программного обеспечения было решено использовать методологию разработки Agile [14]. Данный подход предполагает разбиение процесса разработки на короткие итерации, или спринты, в рамках которых реализуется определенная часть функциональности системы. Каждая итерация включает этапы анализа требований, проектирования, реализации и тестирования, что позволяет оперативно выявлять и исправлять ошибки, а также учитывать изменения требований в процессе разработки.

В рамках Agile используется подход Scrum, предполагающий планирование задач на спринт, контроль их выполнения и регулярную оценку результатов [15]. Основными элементами Scrum являются список задач (backlog), спринты фиксированной длительности, а также регулярный контроль выполнения задач.

В рамках разработки серверной части системы контроля отпусков основным преимуществом такого решения является возможность поэтапной реализации модулей системы без полной переработки проекта. Такой подход позволяет контролировать сроки разработки и упрощает внесение изменений.

Анализ требований в рамках выбранной методологии разработки осуществляется с использованием языка моделирования UML [16]. UML применяется для моделирования структуры и поведения системы. Для построения диаграмм, описывающих работу разрабатываемой системы, используется инструмент Visual Paradigm, позволяющий наглядно представить взаимодействие пользователей с системой и затрагиваемые бизнес-процессы.

Основным языком программирования для разработки серверной части выступает C#. Серверная часть реализуется на платформе ASP.NET Core, которая славиться тем, что позволяет разрабатывать высокопроизводительные веб-приложения с использованием актуальных подход. Разработка ведется в

среде JetBrains Rider 2025. Серверная часть проекта реализуется на основе архитектуры модульного монолита.

Для хранения данных используется система управления базами данных PostgreSQL 17. Взаимодействие с базой данных осуществляется с использованием ORM Dapper, что позволяет выполнять SQL-запросы напрямую. Использование Dapper обусловлено тем, что его альтернатива Entity Framework обладает меньшей скоростью обработки данных и гибкостью самих запросов к БД, так как SQL-запросы формируются автоматически. Формирование запросов вручную, используя Dapper, позволяет использовать только необходимые операции без дополнительной избыточности, что делает работу с данными более эффективной и безопасной. Для проектирования базы данных приложения используется программное средство Visual Paradigm 17.1 [17].

Для развертывания и управления окружением применяется технология Docker. Она позволяет изолировать компоненты системы, упростить процесс запуска приложения и обеспечить переносимость между различными средами.

3.2 Проектирование общей архитектуры программного обеспечения

В качестве принципа работы приложения была выбрана клиент-серверная архитектура [18]. На 2026 год, данный подход является одним из наиболее распространенных для построения веб-приложения. Данная архитектура предполагает разделение системы на клиентскую и серверную части. Клиентская часть отвечает за взаимодействие с пользователем, а серверная содержит всю бизнес-логику и занимается обработкой данных, полученных от клиента.

Преимуществами выбранного подхода являются:

– централизованная обработка данных на серверной части;

– повышение безопасности за счет хранения всех данных на серверной стороне;

– возможность масштабирования системы при увеличении нагрузки;

– независимость клиентской части от серверной.

Принцип работы приложения на основе клиент-серверной архитектуры представлен на рисунке 18.

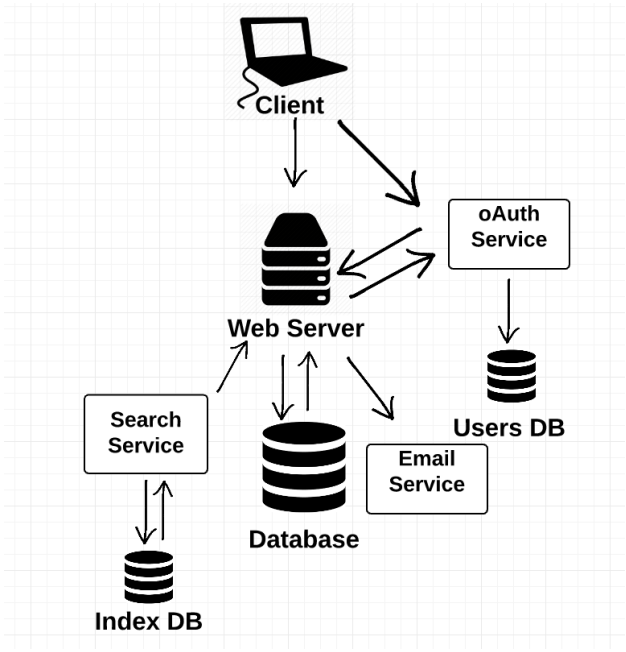

Рисунок 18 – Принцип работы клиент-серверной архитектуры

Взаимодействие в данной архитектуре, заключается в том, что клиентское приложение отправляет HTTP-запросы на сервер. Сервер принимает запросы, обрабатывает их согласно заданной ему логики и, при необходимости, обращается к базе данных для получения или изменения информации. После выполнения всех операций сервер формирует ответ и отправляет его обратно клиенту в формате JSON. Клиентская часть, получив ответ, отображает данные пользователю или выполняет какое-то другое действие, в соответствии с принятым запросом.

Таким образом, суть клиент-серверной архитектуры - это создание двух обособленных друг от друга частей, которые могут взаимодействовать между собой с помощью запросов. Выбор такого подхода для разработки программного обеспечения системы управления отпусками, обусловлен необходимостью обеспечения безопасной обработки данных, поддержки большого числа пользователей и возможности эффективного расширения как серверной, так и клиентской частей приложения.

3.3 Проектирование базы данных

3.3.1 Концептуальное проектирование базы данных

Одним из ключевых этапов проектирования системы контроля отпусков является разработка концептуальной схемы базы данных [19]. Основной задачей данного этапа является выделение ключевых сущностей системы, определение их атрибутов и установка связей между ними.

Концептуальная модель позволяет определить, какие данные необходимо хранить и как они взаимосвязаны между собой. В рамках проектирования были выделены основные сущности, связанные с пользователями, сотрудниками, организационной структурой, а также процессами оформления и согласования отпусков.

На рисунке 19 представлена диаграмма концептуальной модели базы данных системы контроля отпусков.

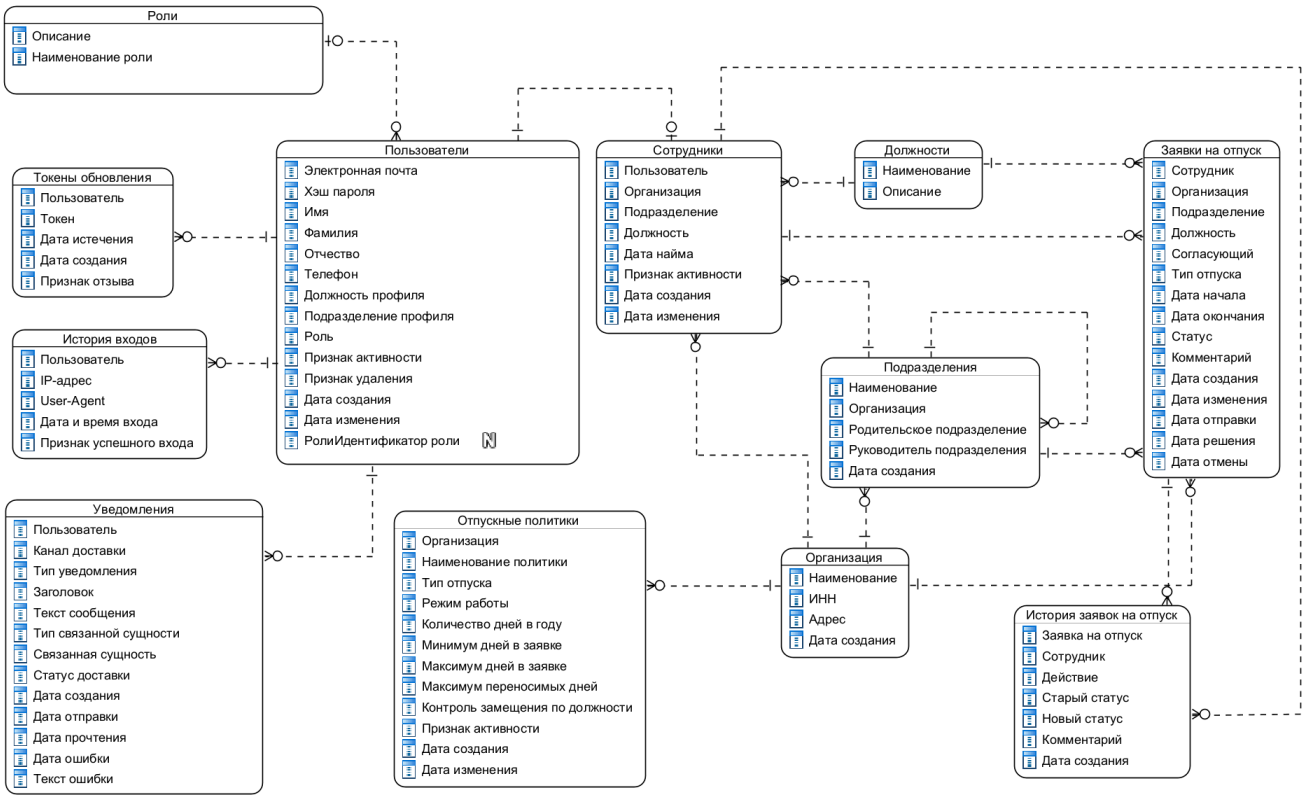

Рисунок 19 – Диаграмма концептуальной модели (схемы) базы данных системы контроля отпусков

В таблице 5 приведено краткое описание сущностей, представленных на концептуальной схеме базы данных проекта.

Таблица 5 – Описание сущностей концептуальной схемы базы данных

№ п/п	Наименование	Описание
1	Роли	Содержит перечень ролей пользователей системы и определяет уровень доступа
2	Пользователи	Хранит учетные записи пользователей и их персональные данные
3	Сотрудники	Описывает сотрудников организации, связанных с пользователями системы
4	Организации	Содержит информацию об организациях
5	Подразделения	Отражает структуру организации с поддержкой иерархии
6	Должности	Справочник должностей сотрудников
7	Отпускные политики	Определяет правила предоставления отпусков
8	Заявки на отпуск	Содержит информацию о заявках сотрудников на отпуск
9	История заявок на отпуск	Хранит изменения статусов заявок
10	Уведомления	Содержит информацию об уведомлениях пользователей
11	Токены обновления	Используется для поддержки механизма авторизации
12	История входов	Хранит информацию о попытках входа в систему

Ключевыми сущностями концептуальной модели являются «Сотрудники» и «Заявки на отпуск. Сущность «Сотрудники» хранит информацию о работниках и связывает пользователя с организацией, подразделением и должностью. Иерархия организации реализуется через сущность «Подразделения», в которой предусмотрено хранение ссылки на родительское подразделение. Руководитель подразделения определяется путем установки конкретного сотрудника руководителем внутри подразделения. Сущность «Заявки на отпуск» фиксирует данные о заявке и содержит ссылки на сотрудника, подающего заявку на отпуск, его подразделение и организацию.

Подробное описание связей между сущностями представлено в таблице 6.

Таблица 6 – Описание связей концептуальной схемы базы данных

№ п/п	Сущность 1	Сущность 2	Тип связи	Описание
1	Роли	Пользователи	1 : N	Одна роль может быть назначена нескольким пользователям
2	Пользователи	Сотрудники	1 : 0..1	Пользователь может быть связан с одним сотрудником
3	Пользователи	Токены обновления	1 : N	Один пользователь может иметь несколько токенов
4	Пользователи	История входов	1 : N	Один пользователь имеет множество записей входа
5	Пользователи	Уведомления	1 : N	Пользователь получает множество уведомлений
6	Организации	Подразделения	1 : N	Организация включает множество подразделений
7	Подразделения	Подразделения	1 : N	Формируется иерархия подразделений
8	Организации	Сотрудники	1 : N	В организации работает множество сотрудников
9	Подразделения	Сотрудники	1 : N	В подразделении работает множество сотрудников
10	Должности	Сотрудники	1 : N	Должность назначается нескольким сотрудникам
11	Организации	Отпускные политики	1 : N	Организация определяет несколько политик
12	Сотрудники	Заявки на отпуск	1 : N	Сотрудник может создавать несколько заявок
13	Сотрудники	Заявки на отпуск (согласование)	1 : 0..N	Сотрудник может выступать согласующим
14	Организации	Заявки на отпуск	1 : N	Заявки относятся к организации
15	Подразделения	Заявки на отпуск	1 : N	Заявки формируются в подразделениях

Продолжение таблицы 6

№ п/п	Сущность 1	Сущность 2	Тип связи	Описание
16	Должности	Заявки на отпуск	1 : N	В заявке фиксируется должность
17	Заявки на отпуск	История заявок	1 : N	Каждая заявка имеет историю изменений
18	Сотрудники	История заявок	1 : 0..N	Сотрудник выполняет действия по заявкам

Разработанная концептуальная модель базы данных отражает структуру предметной области системы контроля отпусков и взаимосвязи между ее основными компонентами. Полученная модель будет служить основой для разработки логического схемы базы данных.

3.3.2 Логическое проектирование базы данных

На основе концептуальной схемы базы данных была построена логическая модель. Логической моделью данных называют усовершенствованную версию концептуальной модели. В ней схематически представлены ограничения данных, имена сущностей и связи, которые нужно будет реализовать независимо от платформы [20].

Логическая модель базы данных отражает структуру хранения данных с учетом выбранной системы управления базами данных. Она включает в себя описание атрибутов сущностей, типов данных, а также ограничений целостности (первичные и внешние ключи). На фоне концептуальной модели, на данном этапе уточняются характеристики полей и формализуются связи между таблицами.

На рисунке 20 представлена диаграмма логической модели базы данных системы контроля отпусков. Разработанная логическая модель соответствует требованиям третьей нормальной формы. Нормализация направлена на устранение избыточности и предотвращение аномалий при взаимодействии с информацией информации [20].

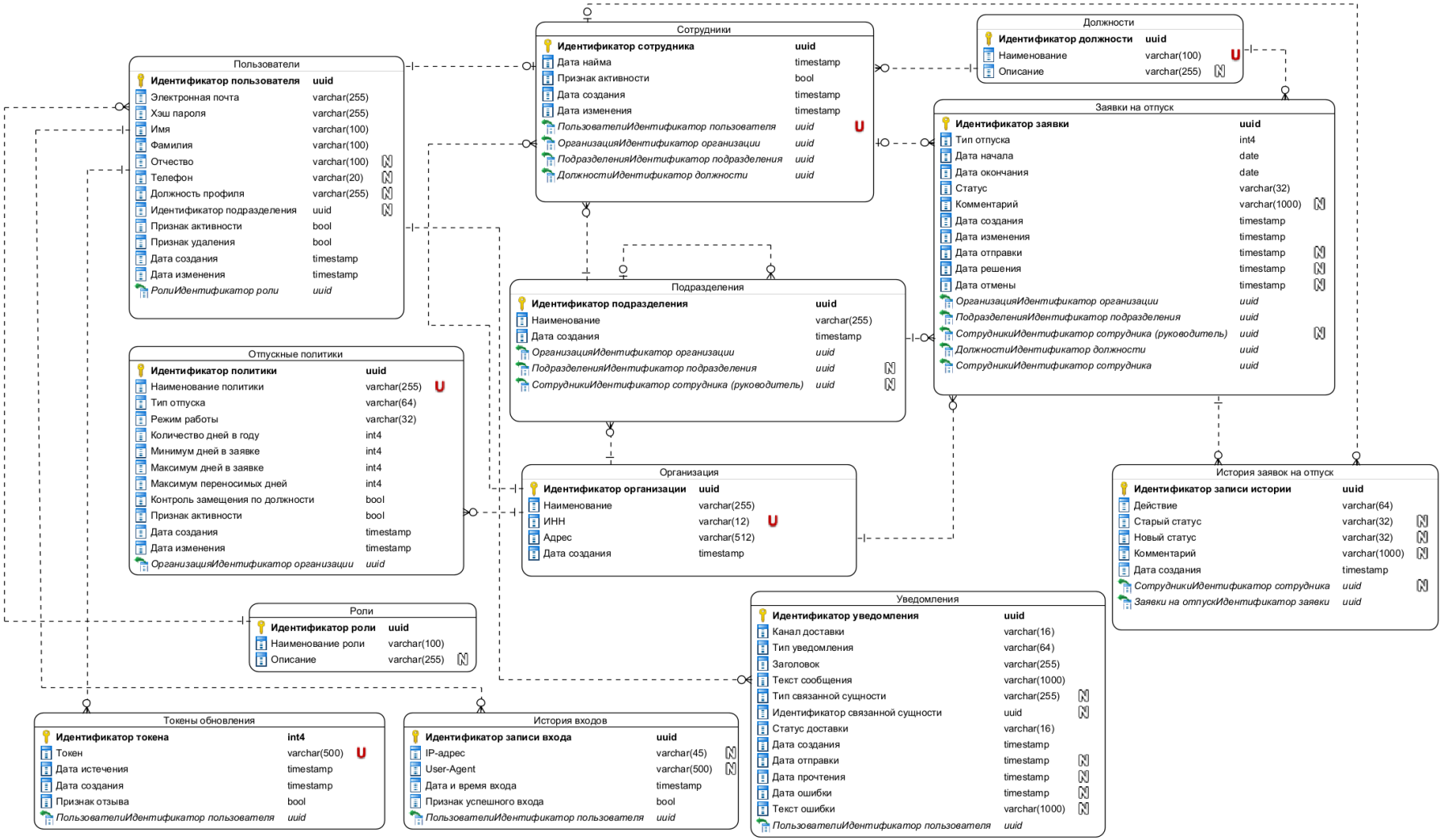

Рисунок 20 – Диаграмма логической модели базы данных системы контроля отпусков

Подробное описание атрибутов и ограничений сущностей представлено в приложении Б.

Таким образом, логическая модель базы данных детализирует структуру хранения данных системы контроля отпусков, включая типы данных и ограничения целостности. Это позволяет обеспечить корректное хранение информации и сформировать основу для последующей реализации физической модели базы данных.

3.4 Проектирование основных алгоритмов системы

На серверной части будет реализован весь основной функционал системы. Далее приведено описание будущих возможностей системы, которое послужит основой для дальнейшей реализации бизнес-логики системы контроля отпусков.

Серверная часть будет отвечает за создание учетных записей, проверку логина и пароля, выдачу JWT-токенов, обновление refresh-токенов и фиксацию истории входов. Также будут учитываться роли пользователей и активность учетной записи.

Система хранит и обрабатывает данные об организациях, подразделениях и должностях. Подразделения могут иметь иерархию, то есть родительские и дочерние отделы, а также назначенного руководителя.

Сервер связывает сотрудника с пользователем, организацией, подразделением и должностью. Также будет хранится дата приема на работу, статус активности и служебные данные, необходимые для проверки прав и отпускных ограничений.

Система позволяет задавать правила отпусков для организации. Можно настроить тип отпуска, годовой лимит дней, минимальную и максимальную длительность, перенос неиспользованных дней и режим работы организации. Для круглосуточного режима включаются дополнительные проверки наличия замены.

Сервер проверяет, нет ли у сотрудника пересекающихся отпусков, хватает ли доступных дней, соблюдены ли лимиты политики и не нарушается ли работа

подразделения. Для круглосуточного режима дополнительно контролируется наличие сотрудника той же должности на замену.

Также система создает уведомления для сотрудников и руководителей о важных событиях. К таким событиям относятся поступление заявки, изменение статуса, согласования или отклонения. Уведомления могут сохраняться в базе и иметь статус доставки или прочтения.

Для заявок фиксируются создание, отправка на согласование, смена статуса, отмена и комментарии. История позволяет отследить, кто и когда выполнил действие.

Центральной бизнес-операцией системы является процесс оформления и согласования заявки на отпуск. Он включает в себя проверку корректности вводимых данных, применение правил отпускной политики, анализ пересечений, а также определение необходимости согласования руководителем. На рисунке 21 представлена диаграмма деятельности [21], которая отображает принцип выполнения операции бронирования отпусков.

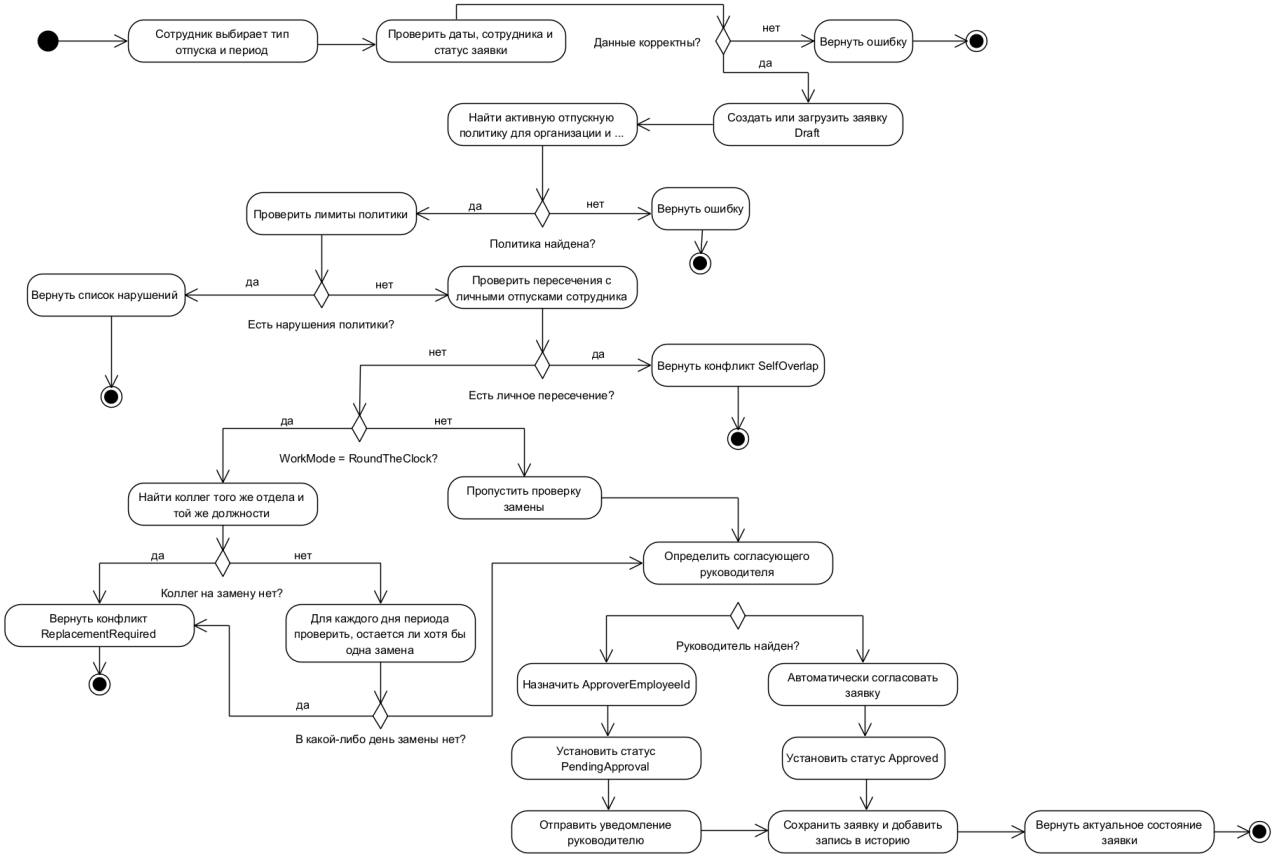

Рисунок 21 – Диаграмма деятельности алгоритма бронирования отпуска

Диаграмма отражает последовательность выполнения операций при создании заявки на отпуск. Процесс начинается с ввода пользователем параметров отпуска. Далее система выполняет валидацию данных: проверяется корректность дат, наличие сотрудника и допустимость текущего состояния заявки. В случае обнаружения ошибок выполнение процесса прекращается с возвратом соответствующего сообщения.

После успешной проверки создаётся новая заявка. Затем система определяет действующую отпускную политику для организации и выбранного типа отпуска. На основании найденной политики выполняется проверка соответствия годовым лимитам, ограничениям по длительности и правилам переноса дней. При нарушении хотя бы одного условия пользователю отображается уведомление об ошибке.

Далее происходит проверка пересечений с уже существующими отпусками сотрудника. При обнаружении конфликта заявка отклоняется. Если для политики установлен круглосуточный режим работы, система выполняет проверку наличия сотрудников для замещения на период отсутствия.

Если все проверки пройдены успешно, система определяет руководителя, ответственного за согласование заявки. При наличии такого сотрудника заявка переводится в статус ожидания подтверждения, и руководителю отправляется уведомление.

После успешного выполнения всех шагов алгоритма заявка сохраняется в базе данных, а также фиксируется запись в истории изменений. После сохранения, сотруднику возвращается текущий статус заявки.

На рисунке 22 представлен интерфейс создания заявки на отпуск, через который пользователь инициирует выполнение данного алгоритма. При заполнении полей формы. Пользователь заполняет следующие поля: тип отпуска, даты начала и окончания, комментарий. Таким образом, фактически, сотрудник формирует входные данные для алгоритма обработки заявки.

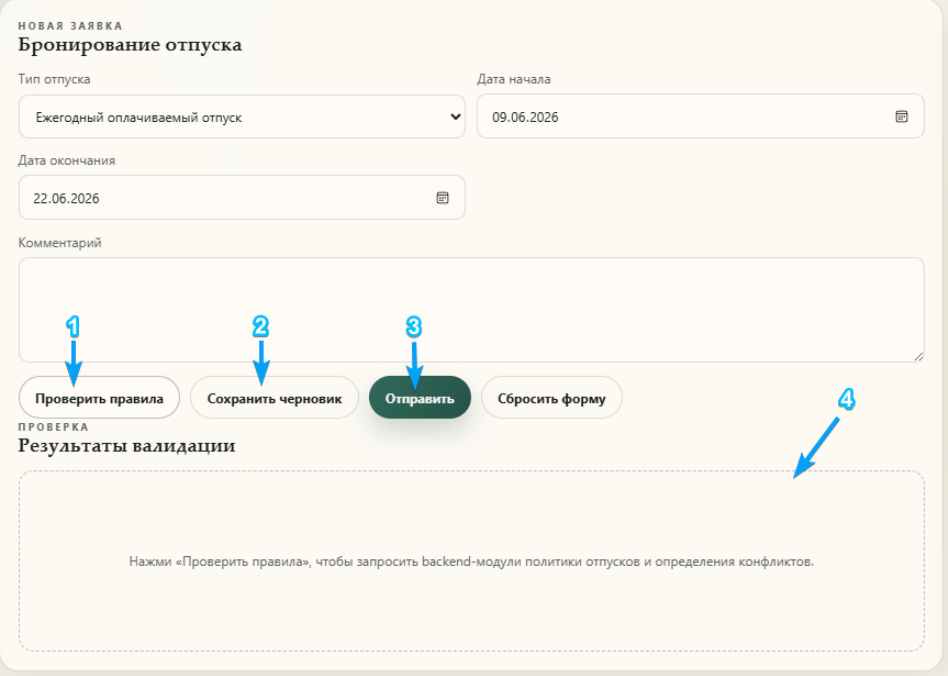

Рисунок 22 – Интерфейс создания заявки на отпуск

Кнопка под цифрой 1 запускает этап предварительной валидации, в ходе которого сервер обращается к модулям отпускных политик и определения конфликтов. В результате пользователь получает информацию о возможных нарушениях, не отправляя заявку на согласование.

Кнопка под цифрой 2 инициирует выполнение части алгоритма, связанной с созданием заявки в статусе «Draft» без прохождения полного цикла проверок и согласования.

Кнопка под цифрой 3 запускает полный алгоритм, представленный на диаграмме деятельности. В этом случае выполняется последовательная проверка данных, применение правил отпускной политики, анализ пересечений и определение согласующего лица. При успешном прохождении всех этапов заявка переводится в статус ожидания согласования.

Результаты выполнения алгоритма отображаются в блоке под номером 4, где пользователь видит сообщения об ошибках или подтверждение успешной обработки.

Таким образом, была успешно спроектирована логика обработки пользовательских запросов. Данные наработки будут служить основой для дальнейшей реализации бизнес-операций.

3.5 Проектирование взаимодействия компонентов системы

На данном этапе проектирования будет определен порядок взаимодействия клиентской и серверной частей системы. Целью является формирование представления о том, каким образом данные будут передаваться между серверной и клиентской частями.

На рисунке 23 представлена диаграмма взаимодействия, сценарии обмена данными между пользователем, клиентским приложением и серверной частью системы контроля отпусков [22].

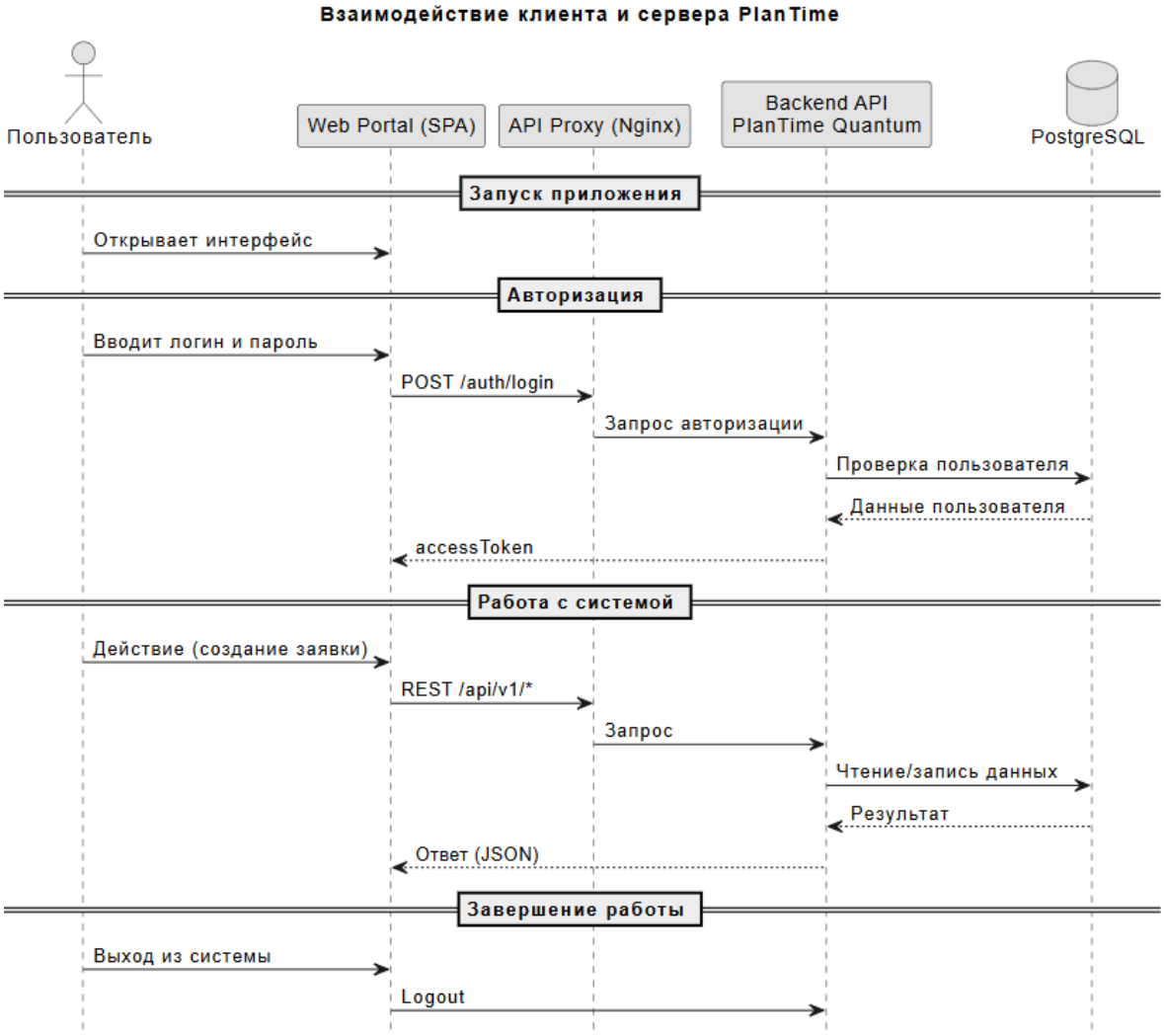

Рисунок 23 – Диаграмма взаимодействия компонентов системы

Взаимодействие системы начинается с открытия сайта. После запуска клиентская часть загружается в браузере пользователя. Далее пользователь выполняет авторизацию. Заданные данные передаются на сервер через API Proxy, который перенаправляет запрос в серверное приложение.

Серверная часть обрабатывает запрос авторизации, обращаясь к базе данных для сверки учетных данных пользователя. В случае успешной аутентификации формируется ответ, содержащий токен доступа, который возвращается обратно в клиентское приложение.

После авторизации клиент получает доступ к определенным возможностям, которые зависят от его роли. Каждое действие инициирует HTTP-запрос с клиентской стороны, который через прокси-сервер поступает на серверную часть. Сервер обрабатывает запрос, выполняя необходимые операции с данными, после чего возвращает результат выполнения операции в виде ответа.

Полученные данные используются клиентским приложением для предоставления информации и отображения актуального состояния системы пользователю.

Таким образом, взаимодействие между компонентами системы реализуется по модели «клиент–сервер» с использованием REST-запросов. Такой подход позволяет разграничить возможности обеих частей системы. Хранение бизнес-логики на сервере позволит упростить поддержку клиентского приложения и обеспечить надежную обработку пользовательских запросов. Разработанная на данном этапе схема взаимодействия будет служить основой для реализации функциональности системы и ее дальнейшего развития.

3.6 Проектирование архитектуры серверной части приложения

При проектировании серверной части приложения была выбрана архитектура модульного монолита. Данный подход описан в книге Марка Ричардса «Фундаментальный подход к программной архитектуре» [23]. Данный архитектурный стиль предполагает разработку единого приложения, внутри

которого система разделена на независимые модули, соответствующие предметным областям. Такой подход является компромиссом между простотой монолита и гибкостью микросервисной архитектуры.

Выбор модульного монолита обусловлен сложностью внедрения микросервисов на ранних этапах разработки и необходимостью обеспечения управляемости системы.

В отличие от классического монолита, где компоненты тесно связаны между собой, модульный монолит предполагает строгое разделение системы на функциональные части.

Ключевым принципом архитектуры является разделение системы по предметным областям. Каждый модуль реализует конкретную часть бизнес-логики. Модуль должен соответствовать следующим принципам:

– модуль минимально зависит от других модулей;

– модуль содержит всю необходимую бизнес-логику;

– взаимодействие осуществляется только через публичные интерфейсы (контракты).

Каждый модуль должен иметь четко определённый контракт, который является точкой входа. Контракт должен быть минимальным, непротиворечивым и стабильным, чтобы избежать нарушения работы зависимых компонентов.

Архитектура построена с учетом возможности дальнейшего масштабирования.

Несмотря на то, что система реализована как единое приложение, логическое разделение на модули позволяет при необходимости выделить отдельные компоненты в независимые микросервисы без значительных изменений.

Сигнатура каждого разрабатываемого модуля приведена на рисунке 24.

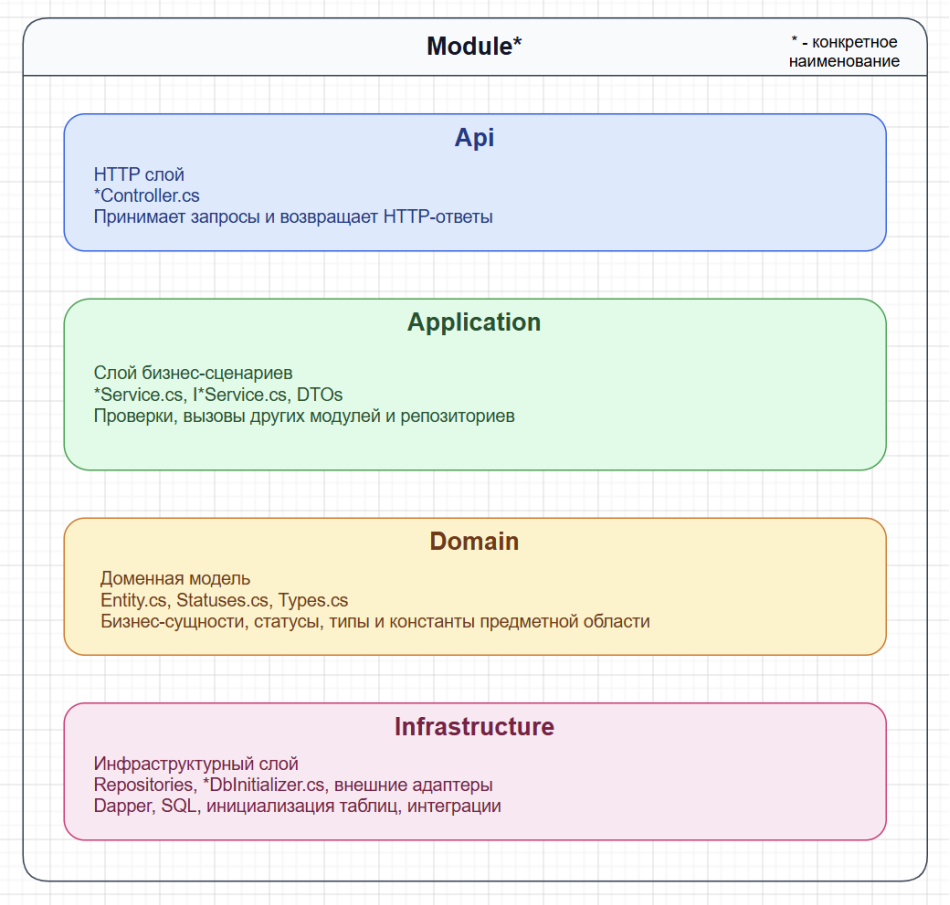

Рисунок 24 – Структура каждого модуля системы

Каждый слой выполняет строго определенную роль:

– Api отвечает за обработку HTTP-запросов, маршрутизацию и формирование ответов;

– Application содержит реализацию бизнес-сценариев и прикладной логики;

– Domain хранит описание предметной области (сущности, статусы, типы);

– Infrastructure отвечает за взаимодействие с внешними системами (БД, SQL, Dapper, уведомления).

Поток выполнения запроса внутри модуля представлен на рисунке 25.

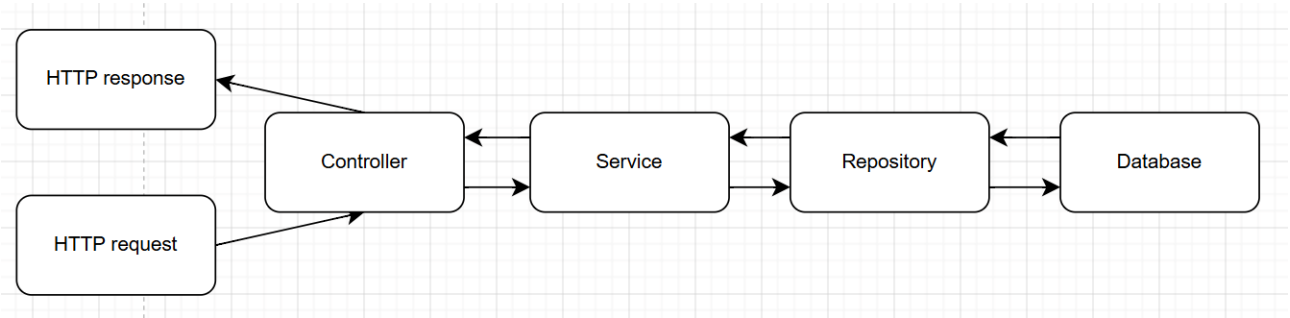

Рисунок 25 – Поток выполнения запроса внутри модуля

Таким образом, выбор модульного монолита в качестве основы разработки серверной части приложения обеспечивает логическое разделение системы на независимые функциональные модули при сохранении простоты развертывания и сопровождения. По сравнению с популярной микросервисной архитектурой данный подход более выгоден с финансовой точки зрения, так как не требует сложной инфраструктуры, распределённого взаимодействия и дополнительных затрат на поддержку отдельных сервисов.

Логическое разделение системы на модули позволяет сохранить гибкость архитектуры. Также при увеличении нагрузки или усложнении системы, отдельные модули, без особых затрат, могут быть выделены в самостоятельные микросервисы с минимальными изменениями.

4 Программная реализация

4.1 Реализация базы данных

В ходе реализации была разработана физическая модель базы данных, которая представляет собой отображение логической модели в выбранной системе управления базами данных [24]. В отличие от логической модели, физическая модель учитывает особенности используемой СУБД, включая типы данных, ограничения целостности и механизмы хранения данных [25].

Физическая модель базы данных реализована в СУБД PostgreSQL и включает в себя набор таблиц, соответствующих ранее разработанным сущностям предметной области системы управления отпусками.

Для сопоставления логической и физической моделей в таблице 7 приведено соответствие наименований сущностей обеих схем.

Таблица 7 – Соответствие логических и физических наименований сущностей

Логическое наименование	Физическое наименование
Роли	Roles
Пользователи	Users
Токены обновления	RefreshTokens
История входов	LoginHistory
Организации	Organizations
Подразделения	Departments
Должности	Positions
Сотрудники	Employees
Отпускные политики	VacationPolicies
Заявки на отпуск	VacationRequests
История заявок	VacationRequestHistory
Уведомления	Notifications

На рисунке 26 представлена разработанная физическая модель базы данных.

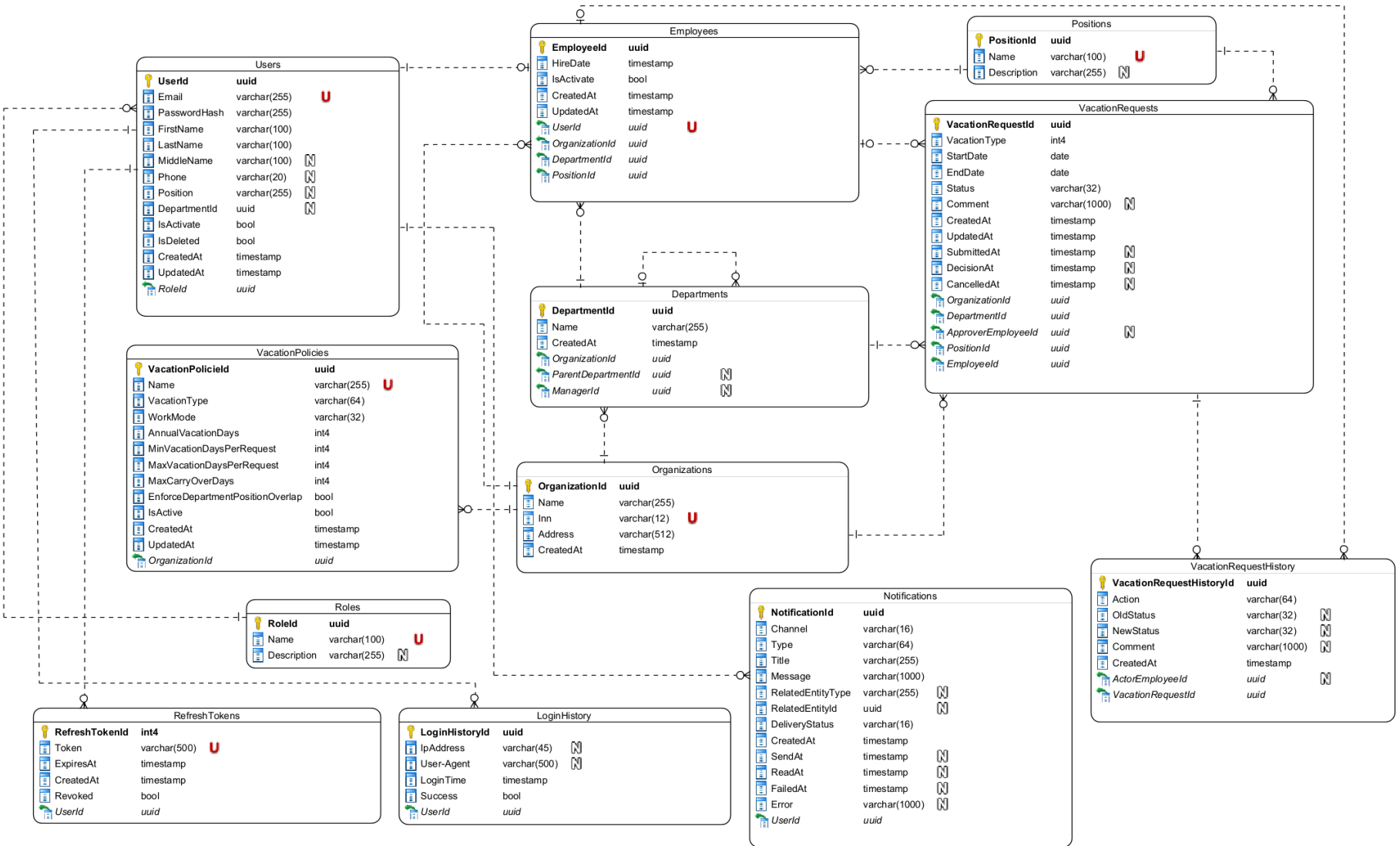

Рисунок 26 – Диаграмма физической модели базы данных системы контроля отпусков

Полные спецификации всех сущностей физической модели, включая атрибуты, ограничения и описание структур, приведены в приложении В.

Для работы с базой данных в серверной части приложения используется слой репозиториев. Данный слой инкапсулирует SQL-запросы и обеспечивает абстракцию доступа к данным. Это дает возможность отделить бизнес-логику от деталей работы с СУБД. Подключение к базе данных осуществляется с использованием вспомогательного класса «DapperContext», который отвечает за создание соединения с PostgreSQL. Непосредственное выполнение запросов реализовано с помощью библиотеки Dapper.

Каждая сущность системы имеет соответствующий репозиторий, реализующий операции получения, создания, обновления и удаления данных. Например, для работы с таблицей пользователей используется класс «UserRepository», содержащий методы для поиска пользователя по электронной почте, создания новой записи и выполнения других операций.

В листинге 1 приведен фрагмент класса «UserRepository», демонстрирующий подход к реализации доступа к данным.

Листинг 1 – Фрагмент класса UserRepository для работы с таблицей Users

```
public class UserRepository : IUserRepository
{
    private readonly DapperContext _context;
    public UserRepository(DapperContext context)
    {
        _context = context;
    }
    public async Task<User?> GetByEmailAsync(
        string email,
        bool includeDeleted = false,
        CancellationToken ct = default)
    {
        var sql = """
            SELECT u."Id", u."Email", u."PasswordHash",
                   u."FirstName", u."LastName", u."MiddleName",
                   u."Phone", u."Position", u."DepartmentId",
                   u."RoleId", u."IsActive", u."IsDeleted",
                   u."CreatedAt", u."UpdatedAt",
                   r."Name" AS "RoleName"
            FROM "Users" u
            INNER JOIN "Roles" r ON r."Id" = u."RoleId"
            WHERE u."Email" = @Email
            """ + (includeDeleted ? "" : " AND u.\"IsDeleted\" = false");
        using var conn = _context.CreateConnection();
        var row = await conn.QuerySingleOrDefaultAsync<UserWithRole>(
            new CommandDefinition(
```

```
                sql,
                new { Email = email },
                cancellationToken: ct));

        return row == null ? null : ToUser(row);
    }
    public async Task<int> CreateAsync(User user, CancellationToken ct =
default)
    {
        var sql = """
            INSERT INTO "Users"
            ("Id", "Email", "PasswordHash", "FirstName", "LastName",
             "MiddleName", "Phone", "Position", "DepartmentId",
             "RoleId", "IsActive", "IsDeleted", "CreatedAt", "UpdatedAt")
            VALUES
            (@Id, @Email, @PasswordHash, @FirstName, @LastName,
             @MiddleName, @Phone, @Position, @DepartmentId,
             @RoleId, @IsActive, @IsDeleted, @CreatedAt, @UpdatedAt)
            """;
        using var conn = _context.CreateConnection();

        return await conn.ExecuteAsync(
            new CommandDefinition(sql, user, cancellationToken: ct));
    }
}
```

Аналогично реализованы репозитории для остальных сущностей системы. Стоит отметить, что каждый репозиторий содержит методы, соответствующие своей предметной области. Использование слоя репозиториев в сочетании с Dapper позволяет обеспечить эффективность работы с базой данных, а также упрощает сопровождение системы.

Таким образом, в данном разделе была разработана физическая модель базы данных системы управления отпусками, представляющая собой реализацию логической модели с учетом СУБД PostgreSQL. Она будет служить основой для реализации серверной части приложения. Также был рассмотрен способ взаимодействия серверной части с базой данных через слой репозиториев, обеспечивающий изоляцию SQL-запросов и возможность удобного расширения функционала системы.

4.2 Реализация серверной части

4.2.1 Описание архитектуры серверной части в нотации C4

Описание архитектуры серверной части система было выполнено в нотации C4 [26]. Она позволяет последовательно представить систему на нескольких уровнях детализации. Данный подход позволяет наглядно

отобразить взаимодействие пользователей с системой, структуры приложений и организации внутренней логики.

Диаграммы архитектуры в нотации C4 были разработаны с использованием инструмента «Structurizr», который предоставляет возможность декларативного описания архитектуры с помощью языка Structurizr DSL и автоматической генерации визуальных схем.

Модель C4 включает четыре уровня.

Контекстная диаграмма показывает систему как «чёрный ящик» и её взаимодействие с внешними пользователями и системами. На этом уровне отображаются акторы и основные сценарии взаимодействия. Это верхний уровень, отображающий назначение системы и её место в окружении.

Диаграмма контейнеров раскрывает внутреннюю структуру системы, показывая основные контейнеры системы и их взаимодействие. Данный уровень показывает из каких технологических частей состоит система.

Диаграмма компонентов раскрывает отдельный контейнер и показывает его внутренние модули или компоненты. На этом уровне отображаются бизнес-модули, их зависимости и взаимодействие.

Диаграмма кода наиболее детализированный уровень, которые отображает классы, интерфейсы и их связи. Используется для глубокой проработки реализации. В рамках данного раздела представлены три уровня архитектурного представления из четырех.

На рисунке 27 приведена легенда контекстной диаграммы системы контроля отпусков. Сама диаграмма представлена на рисунке 28.

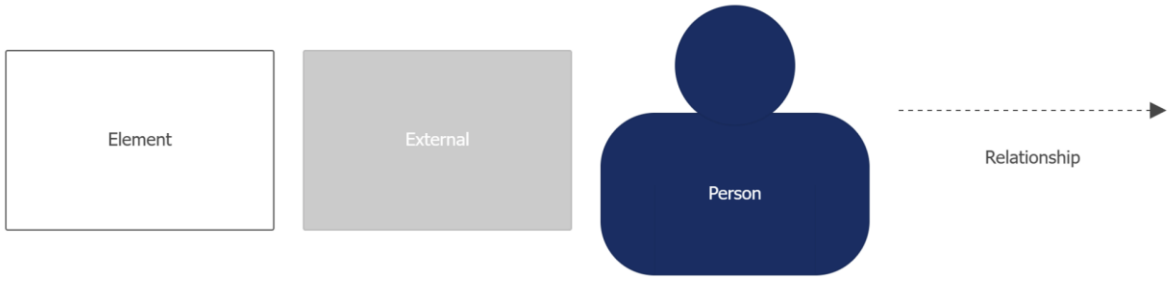

Рисунок 27 – Легенда к диаграмме контекста системы контроля отпусков

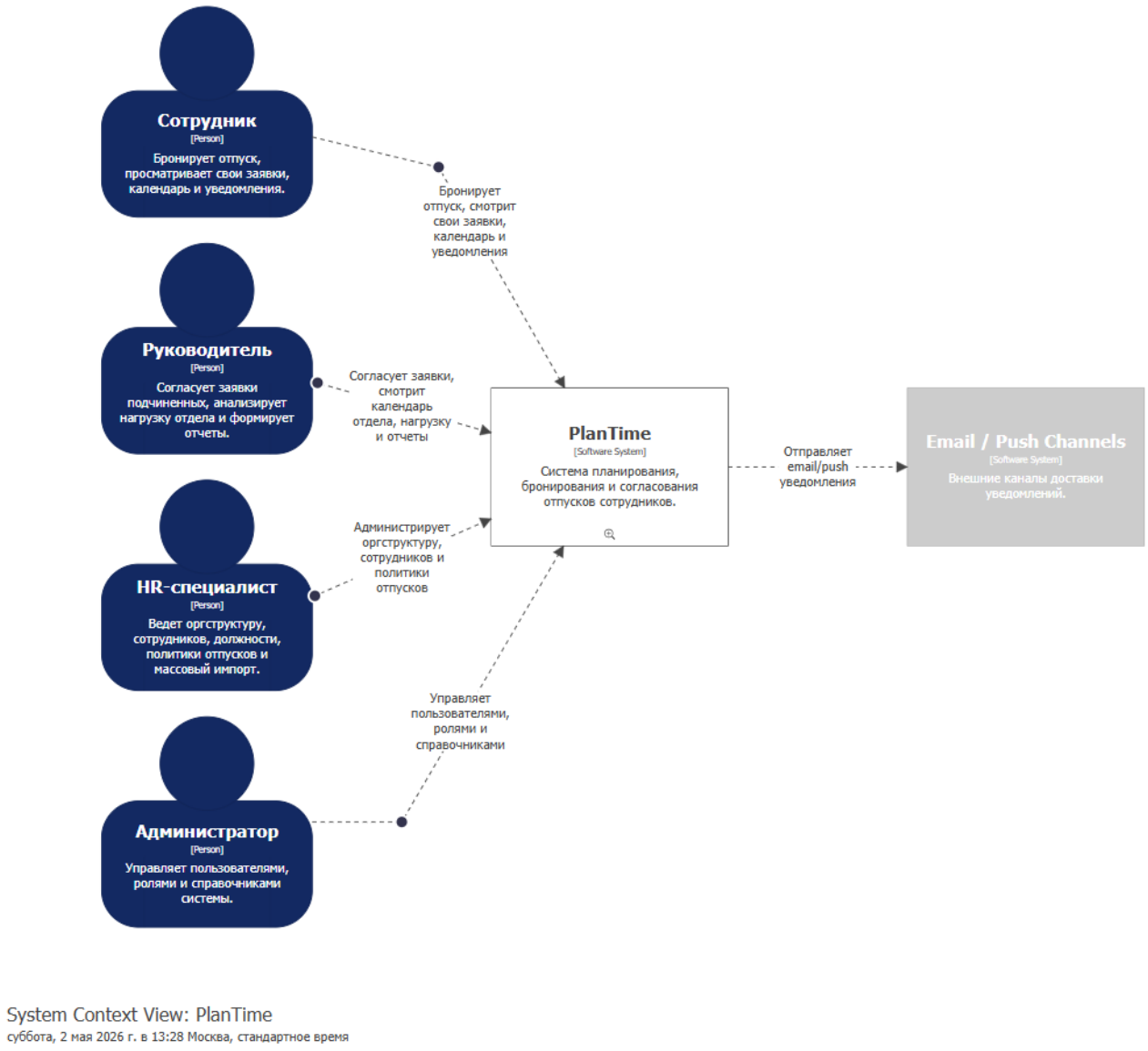

Рисунок 28 – Диаграмма контекста системы контроля отпусков

Диаграмма контекста отражает взаимодействие системы контроля отпусков с основными типами пользователей и внешними сервисами.

В системе выделены следующие роли пользователей:

– сотрудник – формирует заявки на отпуск, просматривает календарь и уведомления;

– руководитель – согласует заявки и анализирует загрузку подразделения;

– HR-специалист – управляет организационной структурой и политиками отпусков;

– администратор – управляет пользователями, ролями и справочниками.

Также система взаимодействует с внешними каналами доставки уведомлений, через которые осуществляется информирование пользователей.

На рисунке 29 приведена легенда диаграммы контейнеров системы контроля отпусков. Сама диаграмма представлена на рисунке 30.

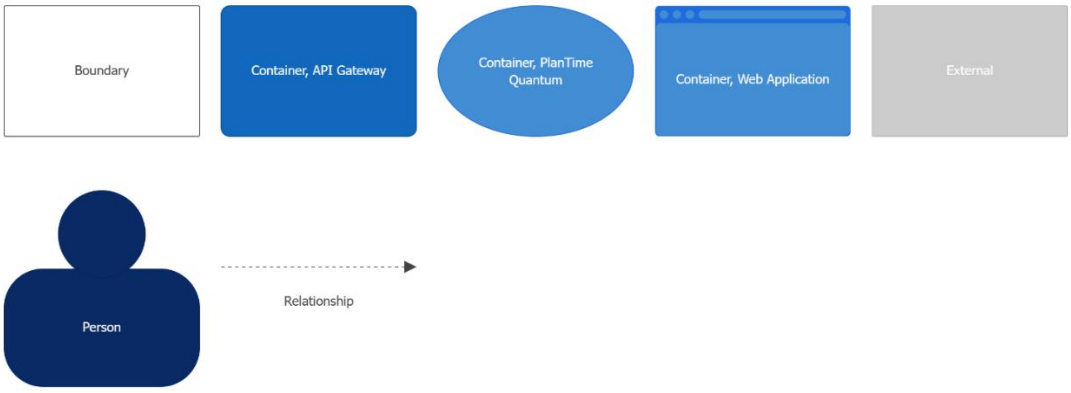

Рисунок 29 – Легенда к диаграмме контейнеров системы контроля отпусков

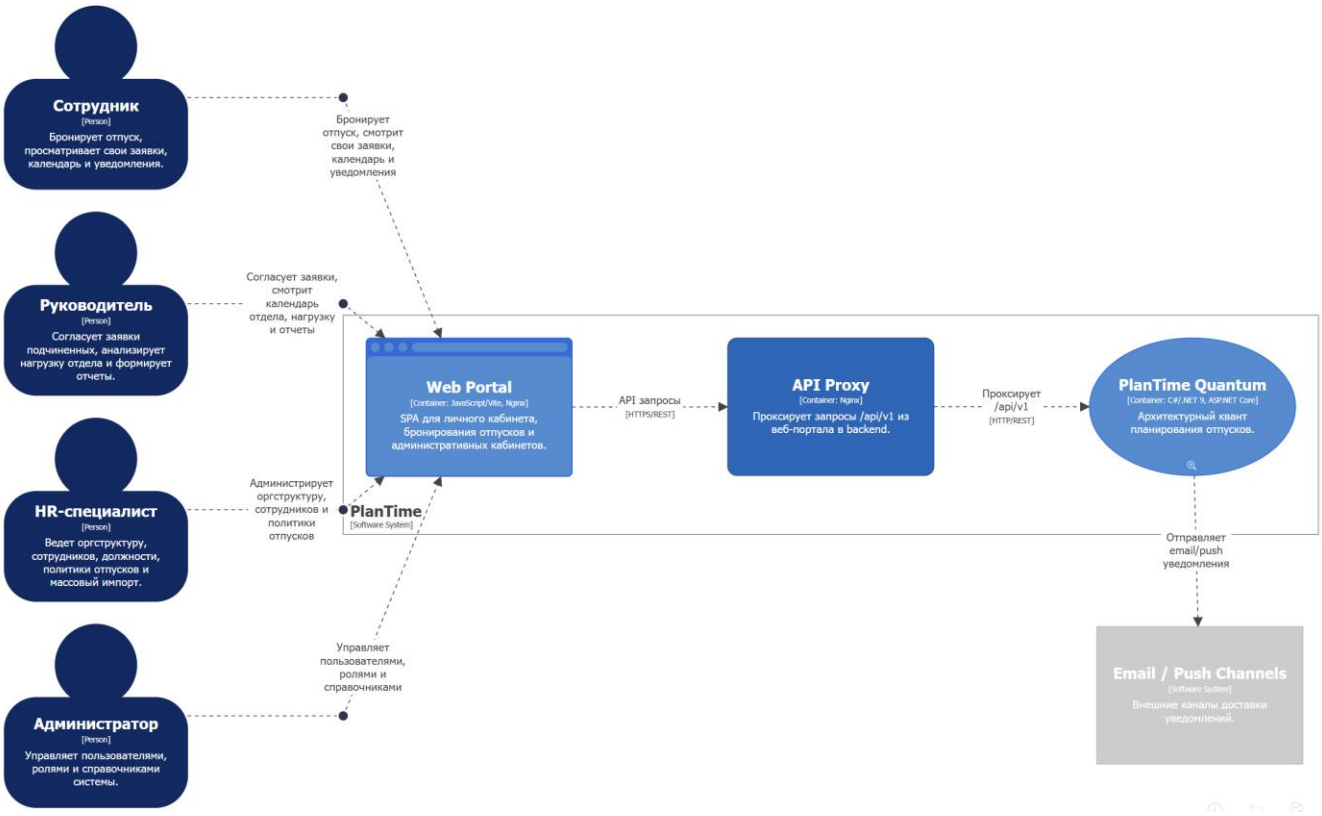

Рисунок 30 – Диаграмма контейнеров системы контроля отпусков

Диаграмма контейнеров раскрывает внутреннюю структуру системы и показывает основные программные элементы.

Система состоит из следующих контейнеров:

– Web Portal – клиентское веб-приложение (SPA), через которое пользователи взаимодействуют с системой;

– API Proxy – промежуточный слой (gateway), принимающий HTTP-запросы и перенаправляющий их в backend;

– PlanTime Quantum – серверное приложение, реализующее бизнес-логику системы.

Взаимодействие между контейнерами осуществляется следующим образом:

– пользователь работает с Web Portal;

– Web Portal отправляет HTTP-запросы через API Proxy;

– API Proxy маршрутизирует запросы в PlanTime Quantum;

– серверная часть обрабатывает запрос и возвращает результат.

Также сервер взаимодействует с внешними сервисами уведомлений для отправки сообщений пользователям.

На рисунке 31 приведена легенда диаграммы компонентов системы контроля отпусков. Сама диаграмма представлена на рисунке 32.

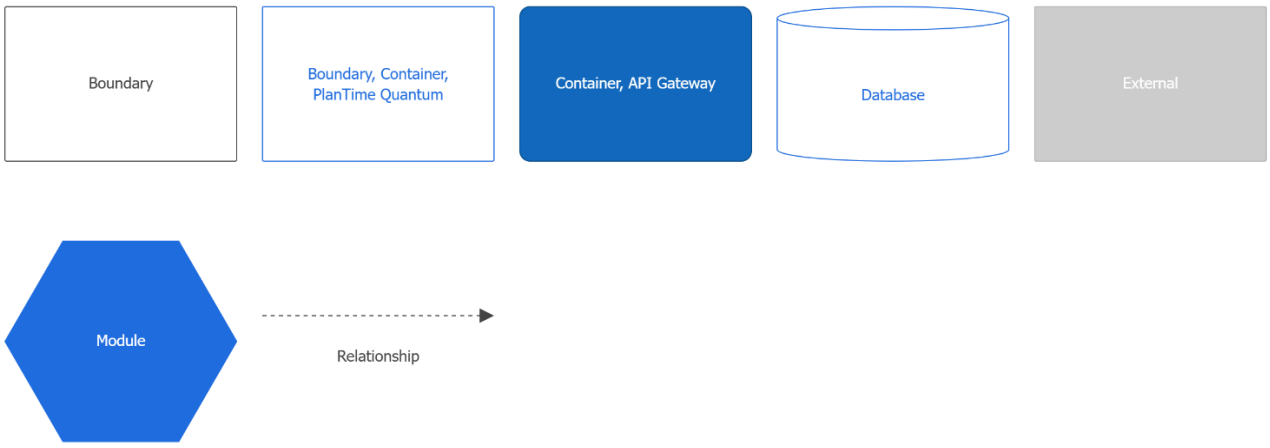

Рисунок 31 – Легенда к диаграмме компонентов системы контроля отпусков

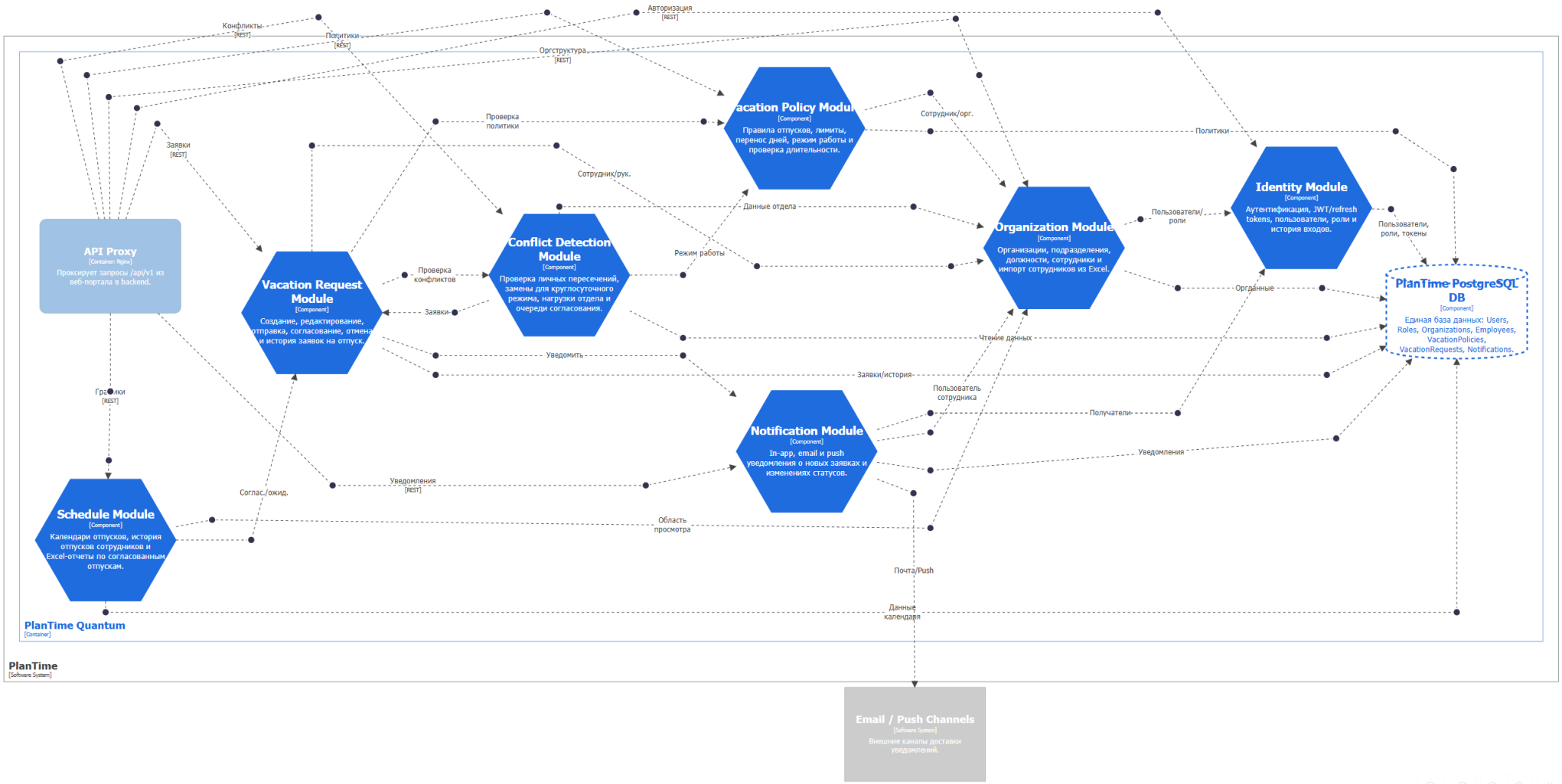

Рисунок 32 – Диаграмма компонентов системы контроля отпусков

Диаграмма компонентов детализирует структуру серверного приложения, реализованного в виде модульного монолита.

Серверная часть разделена на функциональные модули:

– Identity Module – управление пользователями, ролями, аутентификация и история входов;

– Organization Module – управление организациями, подразделениями, должностями и сотрудниками;

– Vacation Policy Module – управление правилами предоставления отпусков;

– Vacation Request Module – работа с заявками на отпуск (создание, изменение, согласование);

– Conflict Detection Module – проверка пересечений отпусков и наличия замещающих сотрудников;

– Schedule Module – формирование календарей отпусков и отчетов;

– Notification Module – управление уведомлениями.

Все модули взаимодействуют с общей базой данных, которая хранит информацию о пользователях, сотрудниках, заявках, политиках и уведомлениях.

Взаимодействие между модулями организовано через четко определенные контракты. Модуль заявок использует модуль политик для проверки ограничений. Модуль заявок обращается к модулю проверки конфликтов для анализа пересечений. Модуль уведомлений используется для информирования пользователей о событиях. Модуль организации предоставляет данные о сотрудниках и структуре компании

Модули серверной части системы совместно образуют единый архитектурный квант. Все модули функционируют в рамках одного процесса, обеспечивая целостную реализацию бизнес-логики системы.

Архитектурный квант представляет собой минимально независимую единицу развертывания, внутри которой обеспечивается высокая согласованность компонентов и низкая связанность [23]. В рамках

разрабатываемой системы таким квантом является серверное приложение «PlanTime Quantum», включающее все функциональные модули системы, объединенные общей инфраструктурой и базой данных.

Взаимодействие между модулями организовано через контракты, что позволяет изолировать внутреннюю реализацию каждого модуля и снизить число зависимостей частей друг от друга. Каждый модуль инкапсулирует свою предметную область и реализует отдельный набор операций.

Таким образом, архитектура системы обеспечивает высокую связность внутри модулей и низкую связанность между ними. Это позволяет повысить управляемость кода, упростить сопровождение и создать возможность для возможного перехода к микросервисной архитектуре.

4.2.2 Структура модулей серверной части

Серверная часть системы реализована в виде модульной архитектуры, в которой каждый модуль отвечает за отдельную предметную область и инкапсулирует соответствующую бизнес-логику. Такой подход позволяет разделить функциональность системы на независимые компоненты, повысить удобство сопровождения, тестирования и дальнейшего развития приложения.

Каждый модуль имеет унифицированную внутреннюю структуру и содержит три основных слоя: контроллеры, сервисы и репозитории.

Взаимодействие между модулями осуществляется через сервисный слой. Например, модуль заявок на отпуск использует модуль отпускных политик для проверки ограничений, модуль анализа конфликтов - для выявления пересечений, а модуль уведомлений - для информирования пользователей о событиях. Такой подход обеспечивает слабую связанность между модулями и высокую связность внутри каждого модуля.

На рисунке 33 представлена диаграмма классов, отражающая структуру модулей серверной части и взаимосвязи между их основными элементами [27]. Диаграмма отображает как внутреннюю организацию каждого модуля, так и взаимодействие между различными модулями системы.

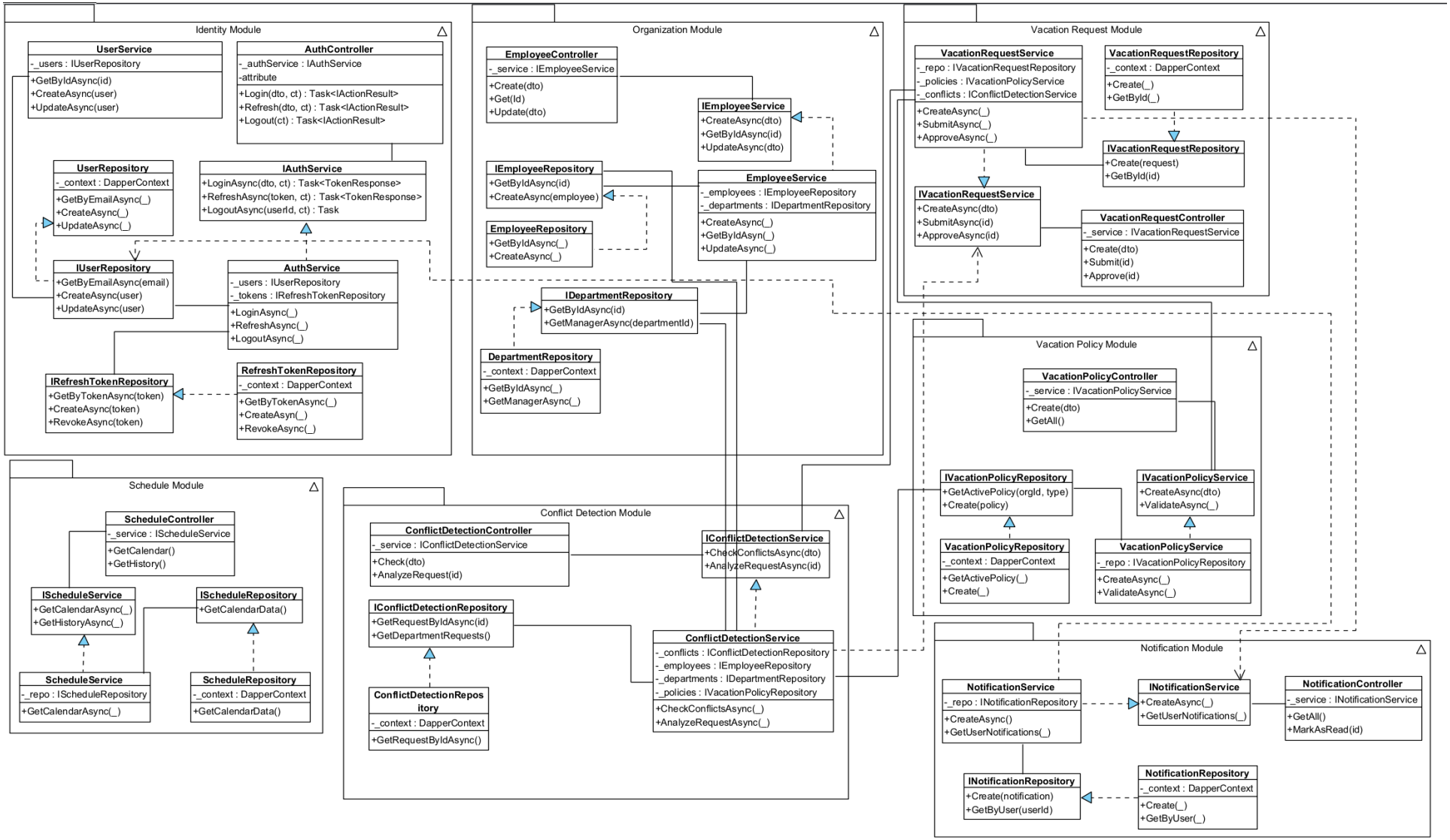

Рисунок 33 – Диаграмма классов системы контроля отпусков

Сервисы реализуют основную бизнес-логику системы. В них выполняются проверки правил, координация работы модулей и обработка пользовательских сценариев. Сервисы взаимодействуют между собой через интерфейсы. Репозитории предназначены для работы с базой. Они выполняют операции чтения, добавления, изменения и удаления записей в базе данных.

Контроллеры представляют внешний уровень взаимодействия с системой и обеспечивают обработку HTTP-запросов от клиента. Они принимают входные данные, выполняют первичную проверку и передают управление в слой сервисов. Например, AuthController обрабатывает запросы, связанные с аутентификацией и управлением сессиями пользователей. EmployeeController отвечает за операции управления сотрудниками. VacationRequestController реализует функциональность создания и согласования заявок на отпуск. VacationPolicyController используется для работы с отпускными политиками. ConflictDetectionController выполняет анализ конфликтов при планировании отпусков. ScheduleController формирует календарные представления и отчеты, а NotificationController управляет уведомлениями пользователей.

В таблице 8 приведено описание основных HTTP-запросов, реализованных в контроллерах системы.

Таблица 8 – Описание основных HTTP-запросов контроллеров системы

Модуль	Контроллер	HTTP метод	URL	Описание	Требуется аутентификация	Роли
Identity	AuthController	POST	/api/v1/auth/login	Аутентификация пользователя	Нет	-
Identity	AuthController	POST	/api/v1/auth/refresh	Обновление access-токена	Нет	-
Identity	AuthController	POST	/api/v1/auth/logout	Выход из системы	Да	Пользователь
Identity	UserController	POST	/api/v1/users	Создать пользователя	Да	Администратор
Identity	UserController	PUT	/api/v1/users	Обновить пользователя	Да	Администратор

Продолжение таблицы 8

Модуль	Контроллер	HTTP метод	URL	Описание	Требуется аутентификация	Роли
Identity	UserController	GET	/api/v1/users/{id}	Получить пользователя по ID	Да	Администратор
Organization	EmployeeController	GET	/api/v1/employees/{id}	Получить сотрудника	Да	Пользователь
Organization	EmployeeController	POST	/api/v1/employees	Создать сотрудника	Да	Администратор
Organization	EmployeeController	PUT	/api/v1/employees	Обновить сотрудника	Да	Администратор
Organization	DepartmentController	GET	/api/v1/departments	Получить список подразделений	Да	Пользователь
Vacation Request	VacationRequestController	POST	/api/v1/vacation-requests	Создать заявку	Да	Пользователь
Vacation Request	VacationRequestController	POST	/api/v1/vacation-requests/{id}/submit	Отправить на согласование	Да	Пользователь
Vacation Request	VacationRequestController	POST	/api/v1/vacation-requests/{id}/approve	Подтвердить заявку	Да	Руководитель
Vacation Request	VacationRequestController	GET	/api/v1/vacation-requests/{id}	Получить заявку	Да	Пользователь
Vacation Policy	VacationPolicyController	POST	/api/v1/vacation-policies	Создать политику	Да	Администратор
Vacation Policy	VacationPolicyController	GET	/api/v1/vacation-policies	Получить политики	Да	Администратор
Conflict Detection	ConflictDetectionController	POST	/api/v1/conflicts/check	Проверка конфликтов	Да	Пользователь
Conflict Detection	ConflictDetectionController	GET	/api/v1/conflicts/requests/{id}	Анализ заявки	Да	Пользователь
Schedule	ScheduleController	GET	/api/v1/schedules/calendar	Получить календарь отпусков	Да	Пользователь
Schedule	ScheduleController	GET	/api/v1/schedules/history	История отпусков	Да	Пользователь
Notification	NotificationController	GET	/api/v1/notifications	Получить уведомления	Да	Пользователь
Notification	NotificationController	POST	/api/v1/notifications/{id}/read	Отметить как прочитанное	Да	Пользователь

Таким образом, в данном разделе была разработана структура модулей серверной части системы. Каждый модуль включает контроллеры, сервисы и репозитории, что обеспечивает четкое разделение ответственности между уровнями приложения. Представленная структура обеспечивает масштабируемость, расширяемость и удобство сопровождения системы. Использование модульного подхода позволяет изолировать изменения в отдельных частях системы и упрощает внедрение новых функций без нарушения существующей логики.

4.2.3 Спецификация модулей серверной части

В ходе реализации серверной части была сформирована спецификация модулей серверной части системы. Спецификация описывает назначение каждого модуля, его основные функции, используемые классы, структуру данных и взаимодействие с другими компонентами системы. Это позволяет формализовать архитектуру и составить сводку функциональности системы.

Спецификация одного из модулей приведена в таблице 9. Исходный код данного модуля представлен в приложении Д. Полный исходный код всей серверной части был размещен в репозитории GitHub пользователя «jkrezz» [28].

Таблица 9 – Спецификация модуля Identity Module

Характеристика	Описание
Назначение	Обеспечивает аутентификацию и авторизацию пользователей, управление учетными записями и токенами
Основные функции	Вход в систему, обновление JWT-токена, выход, управление пользователями и ролями
Основные классы	AuthService, UserService, RoleService, UserRepository, RefreshTokenRepository
Таблицы БД	Users, Roles, RefreshTokens, LoginHistory
Основные API	/api/v1/auth/, /api/v1/users/
Входные данные	Логин, пароль, refresh token, данные пользователя
Выходные данные	JWT access token, refresh token, данные пользователя
Связи с другими модулями	Используется всеми модулями для определения пользователей и ролей

Спецификации остальных модулей представлены в приложении Г.

Таким образом, в данном разделе была описана спецификация модулей серверной части системы. Для каждого модуля определены его назначение, функции, используемые классы, структура данных и взаимодействие с другими компонентами. Представленная спецификация позволяет формализовать архитектуру системы и служит основой для реализации, тестирования и сопровождения программного продукта.

4.3 Структурная организация системы

В ходе реализации разработанного веб-приложения была сформирована структурная модель системы, отражающая состав программных компонентов, их размещение на вычислительных узлах и характер взаимодействия между ними.

На рисунке 34 представлена диаграмма развертывания системы, отражающая физическую архитектуру разработанного приложения и распределение его компонентов по вычислительным узлам [29].

Рисунок 34 – Диаграмма развертывания системы контроля отпусков

В структуре системы выделяются три основных узла. Первый – это клиентское устройство пользователя, на котором функционирует браузер и веб-приложение. Данный компонент отвечает за взаимодействие с пользователем, включая ввод данных, отображение интерфейса и отправку запросов к серверной части.

Второй узел – сервер приложений, на котором размещены веб-сервер и API. Веб-сервер (Nginx) выполняет функции проксирования и обработки HTTP-запросов, а серверное приложение на платформе ASP.NET Core реализует

бизнес-логику системы. Именно на этом уровне происходит обработка пользовательских запросов, выполнение алгоритмов, работа с модулями системы и формирование ответов.

Последний узел – это сервер базы данных, на котором развернута СУБД PostgreSQL. Данный компонент обеспечивает хранение и управление данными системы, включая учетные записи пользователей, организационную структуру, заявки на отпуск, политики и уведомления. Сервер приложений взаимодействует с базой данных посредством отправки и получения запросов.

Взаимодействие между узлами осуществляется по сети. Клиент отправляет HTTP-запросы на сервер приложений, сервер обрабатывает их и при необходимости обращается к базе данных, после чего возвращает результат клиенту. Таким образом обеспечивается разделение ответственности между уровнями системы.

Таким образом, разработанная организация системы реализует клиент-серверное приложений. Выбранный подход способствует изоляции компонентов, упрощает изменение отдельных частей системы и повышает надежность всей архитектуры в целом.

Подводя итог, в рамках раздела была выполнена программная реализация системы управления отпусками, включающая разработку серверной части, клиентского приложения и базы данных. Была спроектирована и реализована модульная архитектура серверной части, обеспечивающая разделение функциональности на независимые компоненты, взаимодействующие через строго определенные контракты. Реализованы основные бизнес-процессы системы. Для работы с данными разработан слой репозиториев, обеспечивающий изоляцию логики доступа к базе данных.

В результате была создана целостная программная система, удовлетворяющая поставленным требованиям и обеспечивающая масштабируемость, расширяемость и удобство дальнейшего внедрения в эксплуатацию.

5 Тестирование системы контроля отпусков

5.1 Обоснование выбора тестирования

Правильно проведенное тестирование позволит проверить корректность работы системы и убедиться в соответствии реализованного функционала заданным требованиям. Для системы контроля отпусков особое значение имеет проверка бизнес-операций, корректности обработки данных полученных с клиентской части, а также устойчивость серверной части приложения.

В рамках разработки системы были выбраны два основных вида тестирования: юнит-тестирование и функциональное тестирование методом черного ящика.

Юнит-тесты используется для проверки компонентов системы отдельно от остальных частей приложения [30]. Основной целью такого подхода является проверка корректности работы методов, сервисов логики серверной части. Это позволяет контролировать корректность вычислений и упростить дальнейшее сопровождение программного кода.

Функциональное тестирование методом черного ящика применяется для проверки работы системы с точки зрения пользователя [31]. Суть данного подхода заключается в анализе реакции системы на различные входные данные без учета внутренней реализации системы. Функциональное тестирование позволяет убедиться в корректности работы различных пользовательских сценариев.

Выбор данных методов тестирования обусловлен архитектурой разработанной системы и необходимостью проверки логики взаимодействия приложения с потребителем. Использование юнит-тестирования и функционального тестирования позволяет значительно повысить конечную надежность программного обеспечения и обеспечить стабильную работу системы контроля отпусков внутри компании.

5.2 Функциональное тестирование черным ящиком

Метод тестирования черным ящиком предполагает проверку работы системы без анализа её внутренней структуры и реализации алгоритмов. В процессе тестирования анализируются входные данные и результаты работы программы.

Функциональное тестирование – один из видов тестирования методом черного ящика. Его основная задача заключается в проверке корректности реализации функций системы. В рамках функционального тестирования выполняется проверка реакции системы на корректные и некорректные входные данные. На рисунке 35 представлен принцип тестирования черным ящиком [32].

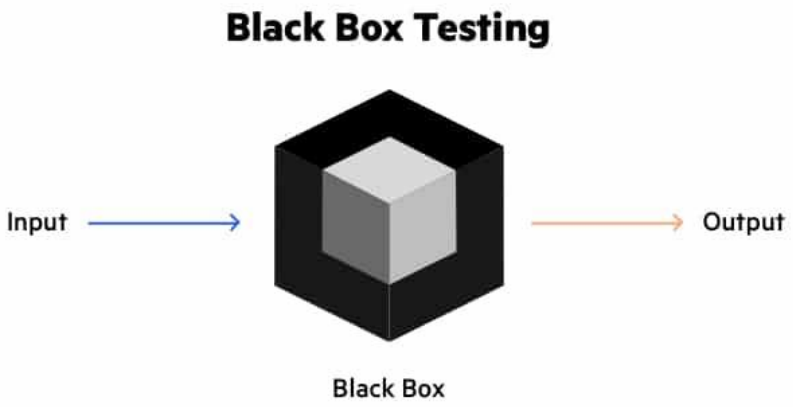

Рисунок 35 – Схема тестирования черным ящиком

Рассмотрим данный подход на примере проверки поля ввода максимальной продолжительности отпуска в одной заявке. Данное поле используется при создании отпускной политики и определяет максимально допустимое количество дней отпуска, которое сотрудник может указать в одной заявке.

В программной реализации данное поле представлено параметром «maxVacationDaysPerRequest». Значение вводится пользователем через интерфейс создания отпускной политики и далее проходит проверку на серверной стороне в методе «ValidatePolicyValues».

При функциональном тестировании выделяются классы эквивалентности. Выделяя классы эквивалентности, мы заявляем, что если на одном элементе класса не находится ошибка, то маловероятно, что она найдется на любом другом представителе этого класса [33]. Классы эквивалентности для данного поля представлены в таблице 10.

Таблица 10 – Классы эквивалентности для поля «maxVacationDaysPerRequest»

	Правильные КЭ	Неправильные КЭ
Количество	14–28 (1)	0-13 (2)
		29 и более (3)

Графическое изображение классов эквивалентности представлено на рисунке 36.

Рисунок 36 – Классы эквивалентности для поля максимальной продолжительности отпуска

Для проверки классов эквивалентности были составлены тест-кейсы. Результаты тестирования данных сценариев приведены в таблице 11.

Таблица 11 – Тест-кейсы эквивалентности

№ теста	Класс эквивалентности	Входные данные	Ожидаемый результат	Фактический результат
1	1	maxVacationDaysPerRequest = 14	Успешное создание политики	Успешное создание политики (Рисунок Д.3)
2	2	maxVacationDaysPerRequest = 13	Отображение ошибки	Ошибка: минимальная продолжительность отпуска не может превышать максимальную (Рисунок Д.2)
3	3	maxVacationDaysPerRequest = 29	Отображение ошибки	Ошибка: максимальная продолжительность отпуска не может превышать годовой лимит (Рисунок Д.5)

После выделения классов эквивалентности были определены граничные условия. В таблице 12 приведено описание возможных границ.

Таблица 12 – Описание граничных условий

Границы	Описание
Физические	То, что не может существовать
Логические	Не подходит по смыслу
Технологические	Перестает работать по техническим причинам
Произвольные	Техническое задание

На основе рассмотренных типов границ были определены граничные значения для поля «maxVacationDaysPerRequest». Граничные значения для рассматриваемого поля представлены в таблице 13.

Таблица 13 – Границы тестирования поля максимальной продолжительности отпуска

Тип границы	Описание	Значение
Физические	Значение не может быть меньше или равно нулю	0
Логические	Значение меньше минимально допустимого	13
Логические	Значение больше годового лимита	29
Технологические	Значение выходит за пределы типа int	больше MaxInt
Произвольные	Нижняя допустимая граница	14
Произвольные	Верхняя допустимая граница	28

Для проверки корректности обработки граничных значений поля были сформированы тест-кейсы функционального тестирования.

Во всех тестовых сценариях остальные поля формы заполнены корректными значениями. Название политики имеет значение «Тестовая политика», тип отпуска – «AnnualPaidVacation», режим работы – Normal, количество дней отпуска в год – 28, минимальная длительность отпуска – 14, перенос остатка – 0.

Результаты функционального тестирования представлены в таблице 14.

Таблица 14 – Тест-кейсы поля «maxVacationDaysPerRequest»

№ теста	Граничное условие	Входные данные	Ожидаемый результат	Фактический результат
1	Ф1	maxVacationDaysPerRequest = 0	Отображение ошибки	Ошибка: максимальная продолжительность отпуска должна быть больше нуля (Рисунок Д.1)
2	Л1	maxVacationDaysPerRequest = 13	Отображение ошибки	Ошибка: минимальная продолжительность отпуска не может превышать максимальную (Рисунок Д.2)
3	П1	maxVacationDaysPerRequest = 14	Успешное создание политики	Политика отпуска успешно создана (Рисунок Д.3)
4	П2	maxVacationDaysPerRequest = 28	Успешное создание политики	Политика отпуска успешно создана (Рисунок Д.4)
5	Л2	maxVacationDaysPerRequest = 29	Отображение ошибки	Ошибка: максимальная продолжительность отпуска не может превышать годовой лимит (Рисунок Д.5)

Таким образом, функциональное тестирование черным ящиком подтвердило корректную работу проверки вводимых данных при создании отпускной политики. Система корректно обрабатывает допустимые и недопустимые значения поля.

5.3 Тестирование с помощью юнит-тестов

Юнит-тестирование – это метод проверки отдельных частей программы, абстрагировано от остальных компонентов системы [30]. При таком подходе тестируется работа отдельных методов, классов или модулей без обращения к базе данных, внешним сервисам и пользовательскому интерфейсу. Для этого используются заглушки и моки, позволяющие имитировать поведение реальных компонентов. Такой подход дает возможность проверить логику тестируемого компонента без зависимости от других сервисов.

В рамках данной работы юнит-тесты были применены для проверки корректности работы контроллеров серверной части системы. Тесты были написаны для основных сценариев обработки запросов на отпуск. В таблице 15 отражен перечень выполненных тестов и их краткое описание.

Таблица 15 – Перечень юнит-тестов

Название теста	Описание
Create_ValidRequest_ReturnsCreatedAtAction	Проверяет, что при успешном создании заявки контроллер возвращает CreatedAtAction
GetById_RequestNotFound_ReturnsNotFound	Проверяет, что при отсутствии заявки контроллер возвращает NotFound
Create_ValidPolicy_ReturnsCreatedAtAction	Проверяет успешное создание отпускной политики
CalculateAvailableDays_ReturnsOk_WithAvailableDays	Проверяет корректный расчет доступных дней отпуска
Check_ReturnsOk_WithConflictAnalysis	Проверяет возврат результата анализа конфликтов при успешной проверке
AnalyzeRequest_RequestNotFound_ReturnsNotFound	Проверяет ответ NotFound при анализе несуществующей заявки
Login_ValidCredentials_ReturnsOk	Проверяет успешную авторизацию и возврат токенов
Refresh_InvalidToken_ReturnsBadRequest	Проверяет обработку невалидного refresh-токена
GetUnreadCount_ReturnsOk_WithUnreadCount	Проверяет корректный возврат количества непрочитанных уведомлений
MarkAsRead_Success_ReturnsNoContent	Проверяет успешную отметку уведомления как прочитанного

На рисунке 37 представлено успешное выполнение всех юнит-тестов, что подтверждает корректность работы основных сценариев серверной части системы. Исходный код каждого теста приведен в приложении Ж.

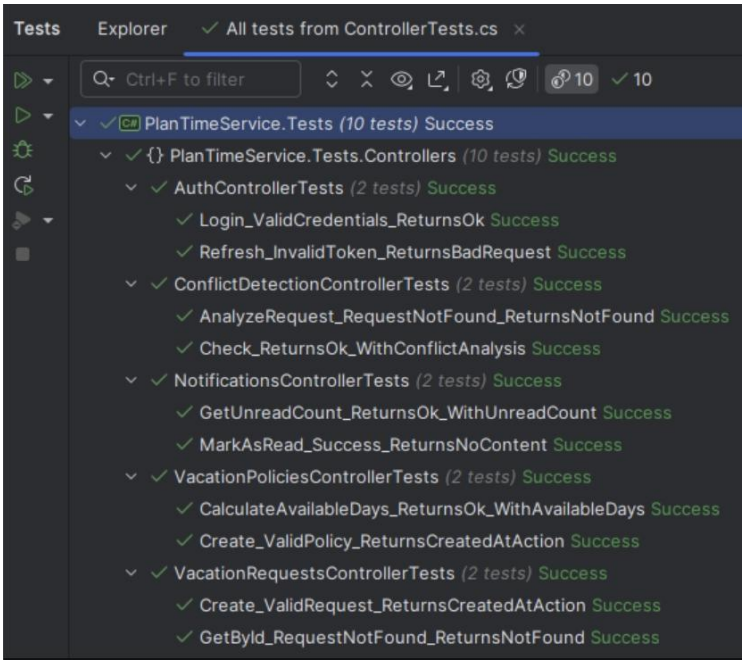

Рисунок 37 – Результат успешного выполнения юнит-тестов

Таким образом, выполненные юнит-тесты подтвердили корректность работы основного функционала системы. Основные сценарии выполняются без нарушения логики. Результаты тестирования показали, что обработка запросов осуществляется корректно.

5.4 Оценка кода с помощью метрик

Метрики – это средства для оценка качества реализации программы [34]. Метрики позволяют количественно оценить характеристики исходного кода, определить сложность реализации, читаемость, связанность компонентов и удобство сопровождения продукта.

В качестве примера, для анализа был выбран один из ключевых методов системы – «SubmitForApprovalAsync». Он реализует процесс отправки заявки на отпуск на согласование. Метод содержит основную бизнес-логику системы и взаимодействует с несколькими модулями одновременно. Взаимодействие осуществляется с модулем заявки на отпуск, отпускных политик, проверки конфликтов и уведомлений.

Реализация данного метода представлен в листинге 2.

Листинг 2 – Метод отправки заявки на отпуск

```
public async Task<Result<VacationRequest>> SubmitForApprovalAsync(Guid id,
SubmitVacationRequestDto dto, CancellationToken ct = default)
    {
        // Загружаем заявку и проверяем, что отправить ее может только автор.
        var request = await _requests.GetByIdAsync(id, ct);
        if (request == null) return Result<VacationRequest>.Fail("Заявка на
отпуск не найдена.");
        if (request.EmployeeId != dto.EmployeeId) return
Result<VacationRequest>.Fail("Отправить заявку может только её автор.");
        if (request.Status != VacationRequestStatuses.Draft)
            return Result<VacationRequest>.Fail("На согласование можно отправить
только черновик заявки.");

        // Получаем актуальные данные сотрудника для дальнейших проверок и
определения согласующего.
        var employee = await _employees.GetByIdAsync(dto.EmployeeId, ct);
        if (employee == null) return Result<VacationRequest>.Fail("Сотрудник не
найден.");

        // Проверяем ограничения отпускной политики: доступные дни, лимиты и
длительность отпуска.
        var policyCheck = await _vacationPolicies.CheckRestrictionsAsync(new
CheckVacationPolicyRestrictionsDto(
            dto.EmployeeId,
```

```
            request.StartDate,
            request.EndDate,
            request.VacationType,
            request.Id), ct);

        if (!policyCheck.IsSuccess) return
Result<VacationRequest>.Fail(policyCheck.Error!);
        if (policyCheck.Value is { IsAllowed: false })
            return Result<VacationRequest>.Fail(string.Join(" ",
policyCheck.Value.Violations));

        // Проверяем конфликты с уже существующими отпусками и требованиями к
замене.
        var conflictCheck = await _conflicts.CheckConflictsAsync(new
CheckVacationConflictDto(
            dto.EmployeeId,
            request.StartDate,
            request.EndDate,
            request.VacationType,
            request.Id), ct);

        if (!conflictCheck.IsSuccess) return
Result<VacationRequest>.Fail(conflictCheck.Error!);
        if (conflictCheck.Value is { HasBlockingConflicts: true })
            return Result<VacationRequest>.Fail(string.Join(" ",
conflictCheck.Value.Conflicts.Select(x => x.Message).Distinct()));

        // Определяем руководителя, который должен согласовать заявку.
        var approverResolution = await ResolveApproverAsync(employee.Id,
employee.DepartmentId, ct);
        if (!approverResolution.IsSuccess) return
Result<VacationRequest>.Fail(approverResolution.Error!);

        // Обновляем служебные поля заявки перед изменением ее статуса.
        var now = DateTime.UtcNow;
        var previousStatus = request.Status;
        request.OrganizationId = employee.OrganizationId;
        request.DepartmentId = employee.DepartmentId;
        request.PositionId = employee.PositionId;
        request.SubmittedAt = now;
        request.UpdatedAt = now;

        if (!string.IsNullOrWhiteSpace(dto.Comment))
            request.Comment = NormalizeComment(dto.Comment);

        // Если согласующий не требуется, заявка утверждается автоматически.
        if (approverResolution.Value!.AutoApprove)
        {
            request.ApproverEmployeeId = null;
            request.Status = VacationRequestStatuses.Approved;
            request.DecisionAt = now;

            await _requests.UpdateAsync(request, ct);
            await AddHistoryAsync(request.Id, dto.EmployeeId,
"ApprovedAutomatically", previousStatus, request.Status, dto.Comment, now, ct);
            return Result<VacationRequest>.Ok(request);
        }

        // Иначе переводим заявку в ожидание согласования и уведомляем
руководителя.
        request.ApproverEmployeeId =
approverResolution.Value.ApproverEmployeeId;
        request.Status = VacationRequestStatuses.PendingApproval;
```

```
        request.DecisionAt = null;

        await _requests.UpdateAsync(request, ct);
        await AddHistoryAsync(request.Id, dto.EmployeeId,
"SubmittedForApproval", previousStatus, request.Status, dto.Comment, now, ct);
        await _notifications.NotifyManagerAboutNewVacationRequestAsync(request,
approverResolution.Value.ApproverEmployeeId!.Value, ct);
        return Result<VacationRequest>.Ok(request);
    }
```

Метод «SubmitForApprovalAsync» реализует сценарий отправки заявки на отпуск на согласование. Метод выполняет проверку существования заявки, принадлежности сотруднику и корректности её статуса. Далее осуществляется проверка отпускной политики и анализ конфликтов с другими отпусками. Затем определяется согласующий руководитель, изменяется статус заявки и сохраняется история изменений.

В последующих разделах для данного метода будут рассмотрены некоторые программные метрики, позволяющие оценить сложность и качество реализации кода.

5.4.1 Обоснование выбора метрик кода

Для оценки качества реализации выбранного метода системы были выбраны метрика SLOC и метрика Холстеда. Программные метрики позволяют количественно оценить характеристики исходного кода и определить сложность реализации алгоритмов.

Метрика SLOC (Source Lines of Code) используется для оценки объема программного кода путем подсчета строк исходного кода [35]. Как правило, в расчет идут только содержательные строки программы без комментариев и пустых строк. Метрика SLOC позволяет определить размер программного компонента, а также оценить уровень структурной сложности метода. Данная метрика широко используется при анализе качества программного кола, благодаря простоте вычисления и наглядности результатов анализа.

Метрика Холстеда позволяет оценить сложность программного обеспечения [34]. Она основывается на анализе операторов и операндов исходного кода. В основе метрики используются 4 основных показателя. Это количество уникальных операторов, количество уникальных операндов, общее

количество операторов и общее количество операндов. На основе этих значений вычисляются словарь программы, длина программы и объем программы. Данная метрика позволяет оценить алгоритмическую сложность кода и трудоемкость его понимания и сопровождения.

Таким образом, выбор рассмотренных метрик обусловлен тем, что рассматриваемый метод «SubmitForApprovalAsync» содержит большое количество условий, проверок и взаимодействий между модулями системы. Использование данных метрик позволит выполнить количественную оценку сложности реализации данного алгоритма и определить уровень сложности метода.

5.4.2 Метрика SLOC

Для оценки объема кода был выбран метод «SubmitForApprovalAsync» из класса «VacationRequestService». Данный метод реализует ключевой сценарий системы, связанный с отправкой заявки на отпуск на согласование.

Метрика Source Lines of Code показывает количество строк исходного кода без учета пустых строк и комментариев. Данная метрика позволяет оценить размер программного метода и уровень его структурной сложности.

Расчет метрики выполнялся для кода метода «SubmitForApprovalAsync». Исходные данные для расчета приведены в таблице 15.

Таблица 15 – Исходные данные для расчета метрики SLOC

Характеристика	Значение
Всего строк метода	72
Количество пустых строк	9
Количество строк комментариев	8

Значение SLOC = 72 – 9 – 8 = 55 строк. При расчете не учитывались пустые строки и комментарии, так как они используются для визуального разделения логических блоков метода, а также повышения читаемости исходного кода. Комментарии дают пояснение отдельным этапам обработки заявок. Пустые

строки позволяют структурировать бизнес-логику по функциональным участкам.

Полученное значение показывает, что метод имеет средний объём программного кода. Основная часть строк связана с проверкой ограничений отпускной политики и обработкой различных сценариев согласования. Таким образом, рассмотренная метрика SLOC показала, что выбранный метод имеет небольшой размер и хорошо структурированную реализацию бизнес-логики.

5.4.3 Метрика Холстеда

Для оценки сложности метода «SubmitForApprovalAsync» была применена метрика Холстеда. Данная метрика позволяет количественно оценить объем и сложность программного кода на основе числа операторов и операндов.

При подсчете будут использоваться следующие правила:

– комментарии, пустые строки, скобки, запятые, точки с запятой, точки и угловые скобки учитываться не будут;

– вызов методов считается как оператор;

– идентификаторы, типы и строковые литералы считаются как операнды.

Для расчета метрики Холстеда определены следующие показатели:

– n1 – число уникальных операторов;

– n2 – число уникальных операндов;

– N1 – общее число операторов;

– N2 – общее число операндов.

В качестве объекта анализа был выбран метод «SubmitForApprovalAsync» из класса «VacationRequestService». В таблице 16 приведено количество операторов и операндов метода «SubmitForApprovalAsync».

Таблица 16 – Подсчет операторов метода «SubmitForApprovalAsync»

Оператор	Количество	Операнд	Количество
=	20	request	34
if	11	Result	12
return	11	VacationRequest	12
await	10	dto	11
Fail	9	ct	11
!	9	Value	8
var	7	EmployeeId	7
==	2	employee	7
!=	2	Status	6
new	2	approverResolution	6
is	2	now	6
Join	2	policyCheck	5
UpdateAsync	2	Id	5
AddHistoryAsync	2	conflictCheck	5
Ok	2	Comment	5
public	1	null	4
async	1	ApproverEmployeeId	4
default	1	_requests	3
CheckRestrictionsAsync	1	VacationRequestStatuses	3
CheckConflictsAsync	1	IsSuccess	3
ResolveApproverAsync	1	Error	3
IsNullOrWhiteSpace	1	string	3
NormalizeComment	1	DepartmentId	3
NotifyManagerAboutNewVacationRequestAsync	1	previousStatus	3
Select	1	IsAllowed	2
Distinct	1	StartDate	2
=>	1	EndDate	2
-	-	VacationType	2

Продолжение таблицы 16

Оператор	Количество	Операнд	Количество
-	-	" "	2
-	-	HasBlockingConflicts	2
-	-	Conflicts	2
-	-	x	2
-	-	OrganizationId	2
-	-	PositionId	2
-	-	DecisionAt	2
-	-	_employees	1
-	-	_vacationPolicies	1
-	-	_conflicts	1
-	-	_notifications	1
-	-	Draft	1
-	-	Approved	1
-	-	PendingApproval	1
-	-	AutoApprove	1
-	-	SubmittedAt	1
-	-	UpdatedAt	1
-	-	Прочие единоразовые операнды (Task, Guid, SubmitVacationRequestDto, CancellationToken, DateTime, UtcNow)	6

Подсчет количества операторов и операндов из таблиц дал следующие результаты: n1 = 28, n2 = 59, N1 = 107, N2 = 219.

На основании подсчета операторов и операндов далее выполняется расчет метрики Холстеда. Полученные значения позволяют определить словарь программы, длину программы, объем, сложность и трудоемкость реализации рассматриваемого метода.

Расчет словаря программы производится по формуле:

$$n = n1 + n2.$$

Подставив полученные ранее значения получаем:

$$n = 28 + 59 = 87.$$

Расчет длины программы производится по формуле:

$$N = N1 + N2.$$

Подставив полученные ранее значения получаем:

$$N = 107 + 219 = 326.$$

Расчет объема программы производится по формуле:

$$V = N \times \log_2(n).$$

Подставив полученные ранее значения получаем:

$$V = 326 \times \log_2 87 \approx 326 \times 6{,}4429 = 2100{,}7.$$

Расчет сложности программы производится по формуле:

$$D = \frac{n1}{2} \times \frac{N2}{n2}.$$

Подставив полученные ранее значения получаем:

$$D = \frac{28}{2} \times \frac{219}{59} \approx 51{,}97.$$

Расчет усилия разработки производится по формуле:

$$E = D \times V.$$

Подставив полученные ранее значения получаем:

$$E = 51{,}97 \times 2100{,}7 \approx 109165{,}43.$$

Общая сводка полученных результатов представлена в таблице 17.

Таблица 17 – Результаты расчета метрики Холстеда

Характеристика	Значение
Словарь программы	87
Длина программы	326
Объем программы	2100,7
Сложность программы	51,97
Усилия разработки	109165,43

Полученные значения показывают, что метод содержит большое количество операторов и операндов. Это связано с тем, что он реализует

несколько последовательных проверок и сценариев обработки заявки. Наибольший вклад в сложность вносят проверки отпускной политики, анализ конфликтов, изменение статуса заявки, а также фиксация истории изменений.

Таким образом, метрика Холстеда показывает, что рассматриваемый метод является достаточно сложным, но при этом обладает хорошей структурированностью и уровнем восприятия.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана серверная часть информационной системы контроля отпусков сотрудников.

В рамках стартап-части был выполнен анализ рынка HR-технологий, рассмотрены конкуренты, определена целевая аудитория, сформирована бизнес-модель проекта, описаны риски, финансовая модель, этапы развития проекта и состав команды. Анализ подтвердил востребованность разработанного командой решения на российском рынке.

В рамках анализа была изучена предметная область управления отпусками, выделены основные проблемы ручного планирования, определены роли пользователей и сформированы функциональные и нефункциональные требования к системе. По результатам анализа области был сделан вывод о необходимости автоматизации контроля отпусков, особенно в организациях с большим числом сотрудников и круглосуточным режимом работы.

Общей архитектурой системы была выбрана клиент-серверная архитектура. Серверная часть приложения реализована с учетом принципов архитектуры модульного монолита. Были спроектированы основные алгоритмы системы. Также разработаны концептуальная и логическая модели базы данных, определены сущности и их взаимосвязи. Каждая сущность была приведена к требованиям третьей нормальной формы. Нормализация гарантирует минимизацию избыточности данных и позволяет обеспечить целостность информации.

В ходе программной реализации была создана база данных и реализована серверная часть приложения. Были разработаны основные контроллеры, сервисы и репозитории, обеспечивающие работу с пользователями, сотрудниками, заявками на отпуск, политиками отпусков, конфликтами и уведомлениями.

В разделе тестирования было проведено юнит-тестирование и функциональное тестирование методом черного ящика. Проверка показала

корректность работы основных бизнес-сценариев системы. Дополнительно были рассчитаны метрики качества кода SLOC и Холстеда, результаты которых показали умеренную сложность реализованных алгоритмов.

Таким образом, цели выпускной квалификационной работы были достигнуты. Разработанная система автоматизирует процессы управления отпусками и может служить основой для дальнейшего развития и внедрения в организациях различного профиля.

Список использованных источников

1. «Трудовой кодекс Российской Федерации» от 30.12.2001 N 197-ФЗ (ред. от 29.12.2025, с изм. от 06.02.2026) [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW_34683/ (дата обращения 12.04.2026).

2. Клиент-сервер - о технологии простыми словами [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/klient-servernaya-arhitektura/ (дата обращения 12.04.2026).

3. HR Automation Statistics [Электронный ресурс]. URL: https://wifitalents.com/hr-automation-statistics/ (дата обращения 13.04.2026).

4. 120 Automation Statistics: AI, Machine Learning, and More [Электронный ресурс]. URL: https://flair.hr/en/blog/automation-statistics/ (дата обращения 14.04.2026).

5. Российский рынок HR-tech [Электронный ресурс]. URL: https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%80%D1%8B%D0%BD%D0%BE%D0%BA_HR-tech (дата обращения 15.04.2026).

6. В I полугодии 2025 года рынок HRtech вырос на 12% - до 40,6 млрд рублей [Электронный ресурс]. URL: https://smartranking.ru/ru/analytics/hrtech/v-i-polugodii-2025-goda-rynok-hrtech-vyros-na-12-do-406-mlrd-rublej-odnako-tempy-rosta-ostayutsya/ (дата обращения 16.04.2026).

7. HR Tech в 2026 году: как технологии трансформируют управление персоналом в России [Электронный ресурс]. URL: https://vc.ru/hr/2791570-hr-tech-v-2026-godu-transformatsiya-upravleniya-personalom-v-rossii (дата обращения 17.04.2026).

8. Анализ продукта BambooHR. [Электронный ресурс]. URL: https://www.bamboohr.com/ (дата обращения 12.04.2026).

9. Анализ продукта TopFactor. [Электронный ресурс]. URL: https://topfactor.pro/ (дата обращения 12.04.2026).

10. Анализ продукта 1C:HRM. [Электронный ресурс]. URL: https://v8.1c.ru/hrm/ (дата обращения 12.04.2026).

11. Клиент, бизнес, государство и торговые отношения. Как работают разные модели продаж. [Электронный ресурс]. URL: https://www.uplab.ru/blog/b2b-b2g-b2c-d2c-obzor-modelej-prodazh/ (дата обращения 12.04.2026).

12. Арлоу Дж., Нейштадт А. UML 2 и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. - СПб.: Символ-Плюс, 2007. - 624 с.

13. Что такое диаграмма Ганта и как её построить [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/diagramma-ganta/ (дата обращения 12.05.2026).

14. Agile: что это такое и где используется, принципы методологии [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/metodology-agile/ (дата обращения 23.04.2026).

15. Кен Швабер, Джефф Сазерленд. Руководство по Scrum. Исчерпывающее руководство по Scrum: правила игры [Электронный ресурс]. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Russian.pdf (дата обращения 24.04.2026).

16. Проектирование программного обеспечения [Электронный ресурс]. URL: https://habr.com/ru/articles/74330/ (дата обращения 25.04.2026).

17. Visual Paradigm - AI-Powered Visual Modeling Platform [Электронный ресурс]. URL: https://www.visual-paradigm.com/ (дата обращения 26.04.2026).

18. Трутнев Д. Р. Архитектуры информационных систем. Основы проектирования: учебное пособие [Электронный ресурс]. URL: https://books.ifmo.ru/file/pdf/919.pdf (дата обращения 27.04.2026).

19. Токмаков Г. П. Базы данных. Концепция баз данных, реляционная модель данных, языки SQL и XML: учебное пособие [Электронный ресурс].

URL: https://www.andpop.ru/courses/db_books/Tokmakov.pdf (дата обращения 28.04.2026).

20. Голицына О. Л., Партыка Т. Л., Попов И. И. Основы проектирования баз данных: учебное пособие. - 2-е изд. - М.: Форум, НИЦ ИНФРА-М, 2016. - 416 с.

21. Иванов Д., Новиков Ф. Моделирование на UML: учебно-методическое пособие [Электронный ресурс]. URL: https://books.ifmo.ru/file/pdf/722.pdf (дата обращения 29.04.2026).

22. Общее введение, типы данных и средства определения структур SQL [Электронный ресурс]. URL: https://intuit.ru/studies/courses/75/75/lecture/28057 (дата обращения 30.04.2026).

23. Ричардс М., Форд Н. Фундаментальный подход к программной архитектуре. 2-е межд. изд. - СПб.: Питер, 2024. - 448 с.

24.Голицына О. Л., Максимов Н. В., Попов И. И. Базы данных: учебное пособие. - 2-е изд., испр. и доп. - М.: Форум, ИНФРА-М, 2009. - 400 с. [Электронный ресурс]. URL: http://znanium.com/bookread2.php?book=182482 (дата обращения 02.05.2026).

25. What's the Difference Between a Logical and Physical Data Model? [Электронный ресурс]. URL: https://aws.amazon.com/ru/compare/the-difference-between-logical-and-physical-data-model/ (дата обращения 03.05.2026).

26. C4 model [Электронный ресурс]. URL: https://c4model.com/ (дата обращения 04.05.2026).

27. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя [Электронный ресурс]. URL: http://lib.ysu.am/disciplines_bk/90b613116c1e0df8426ecf8e43caa35a.pdf (дата обращения 05.05.2026).

28. PlanTimeProject. Код серверной части проекта контроля отпусков [Электронный ресурс]. URL: https://github.com/jkrezz/PlanTimeProject (дата обращения (05.05.2026).

29. Проектирование программного обеспечения [Электронный ресурс]. URL: https://habr.com/ru/articles/74330/ (дата обращения 06.05.2026).

30. Хориков В. Принципы юнит-тестирования [Электронный ресурс]. URL: https://www.amazon.com/ (дата обращения 07.05.2026).

31. Тестирование методом «черного ящика» [Электронный ресурс]. URL: https://tquality.ru/blog/testirovanie-metodom-chernogo-yashchika/ (дата обращения 08.05.2026).

32. Что такое Black Box Testing? [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/700858/ (дата обращения 09.05.2026).

33. Майерс Г., Сандлер К., Бэджетт Т. Искусство тестирования программного обеспечения [Электронный ресурс]. URL: https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf (дата обращения 09.05.2026).

34. Метрики качества ПО и целевые показатели проекта: за чем следить и как ими управлять [Электронный ресурс]. URL: https://surf.ru/metrics-of-program-development/ (дата обращения 10.05.2026).

35. Программный код и его метрики [Электронный ресурс]. URL: https://habr.com/ru/companies/intel/articles/106082/ (дата обращения 11.05.2026).

Приложение А

(рекомендуемое)

Спецификация диаграммы вариантов использования

Таблица А.1 – Спецификация варианта использования «Авторизоваться»

Поле	Описание
Наименование	Авторизоваться
ID	1
Краткое описание	Пользователь вводит учетные данные и получает доступ к системе
Действующие лица	Пользователь
Предусловие	Пользователь зарегистрирован в системе
Основной поток	1. Пользователь вводит логин и пароль 2. Система проверяет данные 3. При успешной проверке предоставляется доступ
Постусловие	Пользователь авторизован

Таблица А.2 – Спецификация варианта использования «Просмотреть профиль»

Поле	Описание
Наименование	Просмотреть профиль
ID	2
Краткое описание	Пользователь просматривает информацию о себе и остаток отпуска
Действующие лица	Пользователь
Предусловие	Пользователь авторизован
Основной поток	1. Пользователь открывает профиль 2. Система отображает данные сотрудника и остаток отпуска
Постусловие	Данные успешно отображены

Таблица А.3 – Спецификация варианта использования «Просмотреть уведомления»

Поле	Описание
Наименование	Просмотреть уведомлений
ID	3
Краткое описание	Пользователь просматривает список уведомлений
Действующие лица	Пользователь
Предусловие	Пользователь авторизован

Продолжение таблицы А.3

Поле	Описание
Основной поток	1. Пользователь открывает раздел уведомлений 2. Система отображает список уведомлений
Постусловие	Уведомления просмотрены

Таблица А.4 – Спецификация варианта использования «Оформить отпуск»

Поле	Описание
Наименование	Оформить отпуск
ID	4
Краткое описание	Сотрудник создает заявку на отпуск
Действующие лица	Сотрудник
Предусловие	Пользователь авторизован, есть карточка сотрудника
Основной поток	1. Сотрудник вводит даты отпуска 2. Указывает параметры 3. Сохраняет заявку
Постусловие	Заявка создана

Таблица А.5 – Спецификация варианта использования «Отправить заявку»

Поле	Описание
Наименование	Отправить заявку
ID	5
Краткое описание	Сотрудник отправляет заявку руководителю
Действующие лица	Сотрудник
Предусловие	Заявка создана
Основной поток	1. Пользователь нажимает «Отправить» 2. Система проверяет ограничения 3. Определяет руководителя 4. Отправляет уведомление
Постусловие	Заявка в статусе «На согласовании»

Таблица А.6 – Спецификация варианта использования «Отменить заявку»

Поле	Описание
Наименование	Отменить заявку
ID	6
Краткое описание	Сотрудник отменяет свою заявку
Действующие лица	Сотрудник
Предусловие	Заявка существует
Основной поток	1. Сотрудник выбирает заявку 2. Нажимает «Отменить» 3. Система фиксирует действие
Постусловие	Заявка отменена

Таблица А.7 – Спецификация варианта использования «Согласовать заявку»

Поле	Описание
Наименование	Согласовать заявку
ID	7
Краткое описание	Руководитель одобряет отпуск
Действующие лица	Руководитель
Предусловие	Есть заявка на согласовании
Основной поток	1. Руководитель открывает заявку 2. Нажимает «Согласовать» 3. Система сохраняет решение
Постусловие	Заявка согласована

Таблица А.8 – Спецификация варианта использования «Отклонить заявку»

Поле	Описание
Наименование	Отклонить заявку
ID	8
Краткое описание	Руководитель отклоняет заявку
Действующие лица	Руководитель
Предусловие	Есть заявка
Основной поток	1. Руководитель открывает заявку 2. Нажимает «Отклонить» 3. Указывает причину
Постусловие	Заявка отклонена

Таблица А.9 – Спецификация варианта использования «Просмотреть календарь команды»

Поле	Описание
Наименование	Просмотреть календарь команды
ID	9
Краткое описание	Руководитель просматривает отпуска сотрудников
Действующие лица	Руководитель
Предусловие	Руководитель авторизован
Основной поток	1. Открывается календарь 2. Отображаются отпуска сотрудников
Постусловие	Календарь просмотрен

Таблица А.10 – Спецификация варианта использования «Отменить отпуск подчиненного»

Поле	Описание
Наименование	Отменить отпуск подчиненного
ID	10
Краткое описание	Руководитель отменяет отпуск сотрудника
Действующие лица	Руководитель
Предусловие	Существует активная заявка или утвержденный отпуск
Основной поток	1. Руководитель выбирает сотрудника 2. Открывает заявку и нажимает «Отменить»
Постусловие	Отпуск отменен

Таблица А.11 – Спецификация варианта использования «Сформировать отчет»

Поле	Описание
Наименование	Сформировать отчет
ID	11
Краткое описание	Руководитель формирует отчет по отпускам подразделения
Действующие лица	Руководитель
Предусловие	Руководитель авторизован
Основной поток	1. Выбирается период 2. Система формирует данные 3. Отображается отчет
Постусловие	Отчет сформирован

Таблица А.12 – Спецификация варианта использования «Анализировать конфликты и загрузки подразделения»

Поле	Описание
Наименование	Анализировать конфликты и загрузки подразделения
ID	12
Краткое описание	Руководитель анализирует занятость сотрудников
Действующие лица	Руководитель
Предусловие	Есть данные по отпускам
Основной поток	1. Руководитель открывает аналитику 2. Система отображает конфликты 3. Показывает загрузку подразделения
Постусловие	Информация проанализирована

Таблица А.13 – Спецификация варианта использования «Просмотреть очередь согласования»

Поле	Описание
Наименование	Просмотреть очередь согласования
ID	13
Краткое описание	Руководитель просматривает список заявок, ожидающих согласования
Действующие лица	Руководитель
Предусловие	Руководитель авторизован
Основной поток	1. Руководитель открывает раздел заявок 2. Система отображает список заявок на согласовании
Постусловие	Очередь заявок просмотрена

Таблица А.14 – Спецификация варианта использования «Просмотреть очередь согласования»

Поле	Описание
Наименование	Просмотреть очередь согласования
ID	14
Краткое описание	Руководитель просматривает историю отпусков сотрудника

Продолжение таблицы А.14

Поле	Описание
Действующие лица	Руководитель
Предусловие	Руководитель авторизован
Основной поток	1. Выбирается сотрудник 2. Система отображает историю отпусков
Постусловие	История просмотрена

Таблица А.15 – Спецификация варианта использования «Скачать отчет в Excel»

Поле	Описание
Наименование	Скачать отчет в Excel
ID	15
Краткое описание	Руководитель выгружает отчет по отпускам
Действующие лица	Руководитель
Предусловие	Отчет сформирован
Основной поток	1. Руководитель формирует отчет 2. Нажимает «Скачать» 3. Система формирует Excel-файл
Постусловие	Файл загружен

Таблица А.16 – Спецификация варианта использования «Импортировать сотрудников из Excel»

Поле	Описание
Наименование	Импортировать сотрудников из Excel
ID	16
Краткое описание	HR загружает сотрудников в систему через Excel-файл
Действующие лица	HR-специалист
Предусловие	HR авторизован, файл подготовлен
Основной поток	1. HR загружает Excel-файл 2. Система обрабатывает данные 3. Создаются сотрудники
Постусловие	Сотрудники добавлены в систему

Таблица А.17 – Спецификация варианта использования «Управлять отпускными политиками»

Поле	Описание
Наименование	Управлять отпускными политиками
ID	17
Краткое описание	HR настраивает правила отпусков
Действующие лица	HR-специалист
Предусловие	HR авторизован
Основной поток	1. HR открывает политики 2. Создает или редактирует правила 3. Сохраняет изменения
Постусловие	Политика обновлена

Таблица А.18 – Спецификация варианта использования «Создать карточку сотрудника»

Поле	Описание
Наименование	Создать карточку сотрудника
ID	18
Краткое описание	HR создает нового сотрудника
Действующие лица	HR-специалист
Предусловие	HR авторизован
Основной поток	1. Вводятся данные сотрудника 2. Назначается подразделение и должность 3. Сохраняется запись
Постусловие	Карточка сотрудника создана

Таблица А.19 – Спецификация варианта использования «Управлять пользователями»

Поле	Описание
Наименование	Управлять пользователями
ID	19
Краткое описание	Администратор управляет учетными записями
Действующие лица	Администратор
Предусловие	Администратор авторизован

Продолжение таблицы А.19

Поле	Описание
Основной поток	1. Открывается список пользователей 2. Выполняются действия (создание, редактирование)
Постусловие	Данные пользователей обновлены

Таблица А.20 – Спецификация варианта использования «Создать пользователя»

Поле	Описание
Наименование	Создать пользователя
ID	20
Краткое описание	Администратор создает учетную запись
Действующие лица	Администратор
Предусловие	Администратор авторизован
Основной поток	1. Вводятся данные пользователя 2. Назначается роль 3. Сохраняется запись
Постусловие	Пользователь создан

Таблица А.21 – Спецификация варианта использования «Изменить роль»

Поле	Описание
Наименование	Изменить роль
ID	21
Краткое описание	Администратор изменяет роль пользователя
Действующие лица	Администратор
Предусловие	Пользователь существует
Основной поток	1. Выбирается пользователь 2. Назначается новая роль 3. Сохраняется
Постусловие	Роль изменена

Таблица А.22 – Спецификация варианта использования «Сбросить пароль»

Поле	Описание
Наименование	Сбросить пароль
ID	22
Краткое описание	Администратор сбрасывает пароль пользователя
Действующие лица	Администратор

Продолжение таблицы А.22

Поле	Описание
Предусловие	Пользователь существует
Основной поток	1. Выбирается пользователь 2. Выполняется сброс пароля
Постусловие	Пароль обновлен

Таблица А.23 – Спецификация варианта использования «Изменить статус пользователя»

Поле	Описание
Наименование	Изменить статус пользователя
ID	23
Краткое описание	Администратор меняет статус учетной записи
Действующие лица	Администратор
Предусловие	Пользователь существует
Основной поток	1. Выбирается пользователь 2. Меняется статус
Постусловие	Статус обновлен

Таблица 24 – Сценарий варианта использования «Управлять структурой организации»

Поле	Описание
Наименование	Управлять структурой организации
ID	24
Краткое описание	Администратор управляет организациями, подразделениями и должностями
Действующие лица	Администратор
Предусловие	Администратор авторизован
Основной поток	1. Выбирается сущность 2. Создание/редактирование 3. Сохранение
Постусловие	Структура обновлена

Приложение Б

(рекомендуемое)

Спецификация логической модели базы данных

Таблица Б.1 – Спецификация атрибутов отношения «Роли»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор роли	UUID	Да	Уникальный идентификатор роли	Да	Нет
Наименование роли	VARCHAR(100)	Да	Название роли	Нет	Нет
Описание	VARCHAR(255)	Нет	Описание роли	Нет	Нет

Таблица Б.2 – Спецификация атрибутов отношения «Пользователи»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор пользователя	UUID	Да	Уникальный идентификатор пользователя	Да	Нет
Электронная почта	VARCHAR(255)	Да	Email пользователя	Нет	Нет
Хэш пароля	VARCHAR(255)	Да	Хэш пароля	Нет	Нет
Имя	VARCHAR(100)	Да	Имя	Нет	Нет
Фамилия	VARCHAR(100)	Да	Фамилия	Нет	Нет
Отчество	VARCHAR(100)	Нет	Отчество	Нет	Нет
Телефон	VARCHAR(20)	Нет	Телефон	Нет	Нет
Должность профиля	VARCHAR(255)	Нет	Должность профиля	Нет	Нет
Идентификатор подразделения	UUID	Нет	Подразделение профиля	Нет	Нет
Код роли	UUID	Да	Роль пользователя	Нет	Да
Признак активности	BOOLEAN	Да	Активность	Нет	Нет
Признак удаления	BOOLEAN	Да	Признак удаления	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет

Таблица Б.3 – Спецификация атрибутов отношения «Токены обновления»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор токена	UUID	Да	Уникальный идентификатор	Да	Нет
Токен	VARCHAR(500)	Да	Значение токена	Нет	Нет
Дата истечения	TIMESTAMP	Да	Срок действия	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет
Признак отзыва	BOOLEAN	Да	Отозван ли токен	Нет	Нет
Идентификатор пользователя	UUID	Да	Пользователь	Нет	Да

Таблица Б.4 – Спецификация атрибутов отношения «История входов»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор записи	UUID	Да	Уникальный идентификатор	Да	Нет
IP-адрес	VARCHAR(45)	Нет	IP пользователя	Нет	Нет
User-Agent	VARCHAR(255)	Нет	Клиент пользователя	Нет	Нет
Дата и время входа	TIMESTAMP	Да	Время входа	Нет	Нет
Признак успешного входа	BOOLEAN	Да	Успешность входа	Нет	Нет
Идентификатор пользователя	UUID	Да	Пользователь	Нет	Да

Таблица Б.5 – Спецификация атрибутов отношения «Организации»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор организации	UUID	Да	Уникальный идентификатор	Да	Нет
Наименование	VARCHAR(255)	Да	Название организации	Нет	Нет
ИНН	VARCHAR(12)	Да	ИНН	Нет	Нет
Адрес	VARCHAR(512)	Да	Адрес	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет

Таблица Б.6 – Спецификация атрибутов отношения «Подразделения»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор подразделения	UUID	Да	Уникальный идентификатор	Да	Нет
Наименование	VARCHAR(255)	Да	Название	Нет	Нет
Идентификатор организации	UUID	Да	Организация	Нет	Да
Родительское подразделение	UUID	Нет	Иерархия	Нет	Да
Руководитель подразделения	UUID	Нет	Руководитель	Нет	Да
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет

Таблица Б.7 – Спецификация атрибутов отношения «Должности»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор должности	UUID	Да	Уникальный идентификатор	Да	Нет
Наименование	VARCHAR(100)	Да	Название должности	Нет	Нет
Описание	VARCHAR(255)	Нет	Описание	Нет	Нет

Таблица Б.8 – Спецификация атрибутов отношения «Сотрудники»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор сотрудника	UUID	Да	Уникальный идентификатор	Да	Нет
Идентификатор пользователя	UUID	Да	Пользователь	Нет	Да
Идентификатор организации	UUID	Да	Организация	Нет	Да
Идентификатор подразделения	UUID	Да	Подразделение	Нет	Да
Идентификатор должности	UUID	Да	Должность	Нет	Да
Дата найма	TIMESTAMP	Да	Дата приема	Нет	Нет
Признак активности	BOOLEAN	Да	Активность	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет
Дата изменения	TIMESTAMP	Да	Дата изменения	Нет	Нет

Таблица Б.9 – Спецификация атрибутов отношения «Отпускные политики»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор политики	UUID	Да	Уникальный идентификатор	Да	Нет
Идентификатор организации	UUID	Да	Организация	Нет	Да
Наименование политики	VARCHAR(255)	Да	Название	Нет	Нет
Тип отпуска	VARCHAR(64)	Да	Тип отпуска	Нет	Нет
Режим работы	VARCHAR(32)	Да	Режим	Нет	Нет
Количество дней	INTEGER	Да	Дни	Нет	Нет
Минимум дней	INTEGER	Да	Минимум	Нет	Нет
Максимум дней	INTEGER	Да	Максимум	Нет	Нет
Переносимые дни	INTEGER	Да	Перенос	Нет	Нет
Контроль замещения	BOOLEAN	Да	Контроль	Нет	Нет
Признак активности	BOOLEAN	Да	Активность	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет
Дата изменения	TIMESTAMP	Да	Дата изменения	Нет	Нет

Таблица Б.10 – Спецификация атрибутов отношения «Заявки на отпуск»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор заявки	UUID	Да	Уникальный идентификатор	Да	Нет
Тип отпуска	VARCHAR(64)	Да	Тип	Нет	Нет
Дата начала	DATE	Да	Начало	Нет	Нет
Дата окончания	DATE	Да	Окончание	Нет	Нет
Статус	VARCHAR(32)	Да	Статус	Нет	Нет
Комментарий	VARCHAR(1000)	Нет	Комментарий	Нет	Нет
Дата создания	TIMESTAMP	Да	Создание	Нет	Нет
Дата изменения	TIMESTAMP	Да	Изменение	Нет	Нет
Дата отправки	TIMESTAMP	Нет	Отправка	Нет	Нет
Дата решения	TIMESTAMP	Нет	Решение	Нет	Нет
Дата отмены	TIMESTAMP	Нет	Отмена	Нет	Нет
Сотрудник	UUID	Да	Инициатор	Нет	Да
Согласующий	UUID	Нет	Руководитель	Нет	Да
Организация	UUID	Да	Организация	Нет	Да
Подразделение	UUID	Да	Подразделение	Нет	Да
Должность	UUID	Да	Должность	Нет	Да

Таблица Б.11 – Спецификация атрибутов отношения «История заявок»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор записи	UUID	Да	Уникальный идентификатор	Да	Нет
Действие	VARCHAR(64)	Да	Действие	Нет	Нет
Старый статус	VARCHAR(32)	Нет	Старый статус	Нет	Нет
Новый статус	VARCHAR(32)	Нет	Новый статус	Нет	Нет
Комментарий	VARCHAR(1000)	Нет	Комментарий	Нет	Нет
Дата создания	TIMESTAMP	Да	Дата создания	Нет	Нет
Сотрудник	UUID	Нет	Исполнитель	Нет	Да
Заявка	UUID	Да	Заявка	Нет	Да

Таблица Б.12 – Спецификация атрибутов отношения «Уведомления»

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
Идентификатор уведомления	UUID	Да	Уникальный идентификатор	Да	Нет
Пользователь	UUID	Да	Получатель	Нет	Да
Заявка	UUID	Да	Связанная заявка	Нет	Да
Канал доставки	VARCHAR(16)	Да	Канал	Нет	Нет
Тип уведомления	VARCHAR(64)	Да	Тип	Нет	Нет
Заголовок	VARCHAR(256)	Да	Заголовок	Нет	Нет
Текст сообщения	VARCHAR(1000)	Да	Текст	Нет	Нет
Тип связанной сущности	VARCHAR(255)	Нет	Название события	Нет	Нет
Связанная сущность	UUID	Нет	Идентификатор события	Нет	Нет
Статус доставки	VARCHAR(16)	Да	Статус	Нет	Нет
Дата создания	TIMESTAMP	Да	Создание	Нет	Нет
Дата отправки	TIMESTAMP	Нет	Отправка	Нет	Нет
Дата прочтения	TIMESTAMP	Нет	Прочтение	Нет	Нет
Дата ошибки	TIMESTAMP	Нет	Ошибка	Нет	Нет
Текст ошибки	VARCHAR(1000)	Нет	Описание ошибки	Нет	Нет

Приложение В

(рекомендуемое)

Спецификация физической модели базы данных

Таблица В.1 – Спецификация таблицы Roles

Атрибут	Тип данных	Ограничения	Описание
RoleId	INTEGER	PK, NOT NULL	Идентификатор роли
Name	VARCHAR(100)	NOT NULL, UNIQUE	Наименование роли
Description	VARCHAR(255)	NULL	Описание

Таблица В.2 – Спецификация таблицы RefreshTokens

Атрибут	Тип данных	Ограничения	Описание
RefreshTokenId	UUID	PK, NOT NULL	Идентификатор токена
UserId	UUID	FK, NOT NULL	Пользователь
Token	VARCHAR(500)	NOT NULL, UNIQUE	Токен
ExpiresAt	TIMESTAMPT	NOT NULL	Дата истечения
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
Revoked	BOOLEAN	NOT NULL	Отозван

Таблица В.3 – Спецификация таблицы LoginHistory

Атрибут	Тип данных	Ограничения	Описание
LoginHistoryId	UUID	PK, NOT NULL	Идентификатор записи
UserId	UUID	FK, NOT NULL	Пользователь
IpAddress	VARCHAR(45)	NULL	IP-адрес
UserAgent	VARCHAR(255)	NULL	User-Agent
LoginTime	TIMESTAMPT	NOT NULL	Время входа
Success	BOOLEAN	NOT NULL	Успешность

Таблица В.4 – Спецификация таблицы Positions

Атрибут	Тип данных	Ограничения	Описание
PositionId	UUID	PK, NOT NULL	Идентификатор должности
Name	VARCHAR(255)	NOT NULL, UNIQUE	Наименование
Description	VARCHAR(64)	NULL	Описание

Таблица В.5 – Спецификация таблицы Users

Атрибут	Тип данных	Ограничения	Описание
UserId	UUID	PK, NOT NULL	Идентификатор пользователя
Email	VARCHAR(255)	NOT NULL, UNIQUE	Электронная почта
PasswordHash	VARCHAR(255)	NOT NULL	Хэш пароля
FirstName	VARCHAR(100)	NOT NULL	Имя
LastName	VARCHAR(100)	NOT NULL	Фамилия
MiddleName	VARCHAR(100)	NULL	Отчество
Phone	VARCHAR(20)	NULL	Телефон
Position	VARCHAR(255)	NULL	Должность
DepartmentId	UUID	NULL	Подразделение профиля
RoleId	INTEGER	FK, NOT NULL	Роль
IsActive	BOOLEAN	NOT NULL	Активен
IsDeleted	BOOLEAN	NOT NULL	Удалён
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
UpdatedAt	TIMESTAMPT	NOT NULL	Дата изменения

Таблица В.7 – Спецификация таблицы Departments

Атрибут	Тип данных	Ограничения	Описание
DepartmentId	UUID	PK, NOT NULL	Идентификатор подразделения
Name	VARCHAR(256)	NOT NULL	Наименование
OrganizationId	UUID	FK, NOT NULL	Организация
ParentDepartmentId	UUID	FK, NULL	Родительское подразделение
ManagerId	UUID	FK, NULL	Руководитель
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания

Таблица В.6 – Спецификация таблицы Organizations

Атрибут	Тип данных	Ограничения	Описание
OrganizationId	UUID	PK, NOT NULL	Идентификатор организации
Name	VARCHAR(256)	NOT NULL	Наименование
Inn	VARCHAR(32)	NOT NULL, UNIQUE	ИНН
Address	VARCHAR(512)	NOT NULL	Адрес
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания

Таблица В.8 – Спецификация таблицы VacationPolicies

Атрибут	Тип данных	Ограничения	Описание
VacationPolicyId	UUID	PK, NOT NULL	Идентификатор политики
OrganizationId	UUID	FK, NOT NULL	Организация
Name	VARCHAR(256)	NOT NULL	Наименование
VacationType	VARCHAR(64)	NOT NULL	Тип отпуска
WorkMode	VARCHAR(32)	NOT NULL	Режим работы
AnnualVacationDays	INTEGER	NOT NULL	Количество дней
MinVacationDaysPerRequest	INTEGER	NOT NULL	Минимум дней
MaxVacationDaysPerRequest	INTEGER	NOT NULL	Максимум дней
MaxCarryOverDays	INTEGER	NOT NULL	Перенос
EnforceDepartmentPositionOverlap	BOOLEAN	NOT NULL	Контроль замещения
IsActive	BOOLEAN	NOT NULL	Активна
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
UpdatedAt	TIMESTAMPT	NOT NULL	Дата изменения

Таблица В.9 – Спецификация таблицы Employees

Атрибут	Тип данных	Ограничения	Описание
EmployeeId	UUID	PK, NOT NULL	Идентификатор сотрудника
UserId	UUID	FK, NOT NULL, UNIQUE	Пользователь
OrganizationId	UUID	FK, NOT NULL	Организация

Продолжение таблицы 9

Атрибут	Тип данных	Ограничения	Описание
DepartmentId	UUID	FK, NOT NULL	Подразделение
PositionId	UUID	FK, NOT NULL	Должность
HireDate	TIMESTAMPT	NOT NULL	Дата найма
IsActive	BOOLEAN	NOT NULL	Активен
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
UpdatedAt	TIMESTAMPT	NOT NULL	Дата изменения

Таблица В.10 – Спецификация таблицы Notifications

Атрибут	Тип данных	Ограничения	Описание
NotificationId	UUID	PK, NOT NULL	Идентификатор уведомления
UserId	UUID	FK, NOT NULL	Пользователь
Channel	VARCHAR(16)	NOT NULL	Канал
Type	VARCHAR(64)	NOT NULL	Тип
Title	VARCHAR(256)	NOT NULL	Заголовок
Message	VARCHAR(2000)	NOT NULL	Текст
RelatedEntityType	VARCHAR(64)	NULL	Тип сущности
RelatedEntityId	UUID	NULL	Идентификатор сущности
DeliveryStatus	VARCHAR(16)	NOT NULL	Статус доставки
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
SentAt	TIMESTAMPT	NULL	Отправка
ReadAt	TIMESTAMPT	NULL	Прочтение
FailedAt	TIMESTAMPT	NULL	Ошибка
Error	VARCHAR(2000)	NULL	Текст ошибки

Таблица В.11 – Спецификация таблицы VacationRequests

Атрибут	Тип данных	Ограничения	Описание
VacationRequestId	UUID	PK, NOT NULL	Идентификатор заявки
EmployeeId	UUID	FK, NOT NULL	Сотрудник
OrganizationId	UUID	FK, NOT NULL	Организация
DepartmentId	UUID	FK, NOT NULL	Подразделение
PositionId	UUID	FK, NOT NULL	Должность
ApproverEmployeeId	UUID	FK, NULL	Согласующий
VacationType	VARCHAR(64)	NOT NULL	Тип отпуска
StartDate	TIMESTAMPT	NOT NULL	Дата начала
EndDate	TIMESTAMPT	NOT NULL	Дата окончания
Status	VARCHAR(32)	NOT NULL	Статус
Comment	VARCHAR(1000)	NULL	Комментарий
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания
UpdatedAt	TIMESTAMPT	NOT NULL	Дата изменения
SubmittedAt	TIMESTAMPT	NULL	Дата отправки
DecisionAt	TIMESTAMPT	NULL	Дата решения
CancelledAt	TIMESTAMPT	NULL	Дата отмены

Таблица В.12 – Спецификация таблицы VacationRequestHistory

Атрибут	Тип данных	Ограничения	Описание
VacationRequestHistoryId	UUID	PK, NOT NULL	Идентификатор записи
VacationRequestId	UUID	FK, NOT NULL	Заявка
ActorEmployeeId	UUID	FK, NULL	Исполнитель
Action	VARCHAR(64)	NOT NULL	Действие
OldStatus	VARCHAR(32)	NULL	Старый статус
NewStatus	VARCHAR(32)	NULL	Новый статус
Comment	VARCHAR(1000)	NULL	Комментарий
CreatedAt	TIMESTAMPT	NOT NULL	Дата создания

Приложение Г

(рекомендуемое)

Спецификация модулей системы контроля отпусков

Таблица Г.1 – Спецификация модуля Organization Module

Характеристика	Описание
Назначение	Управляет организационной структурой предприятия
Основные функции	Управление организациями, подразделениями, должностями и сотрудниками
Основные классы	EmployeeService, DepartmentService, PositionService
Таблицы БД	Organizations, Departments, Positions, Employees
Основные API	/api/v1/employees/, /api/v1/departments/
Входные данные	Данные сотрудника, подразделения, должности
Выходные данные	Списки сотрудников, подразделений
Связи с другими модулями	Используется всеми модулями для получения данных о сотрудниках

Таблица Г.2 – Спецификация модуля Vacation Policy Module

Характеристика	Описание
Назначение	Управляет правилами предоставления отпусков
Основные функции	Создание политик, проверка ограничений, расчет дней отпуска
Основные классы	VacationPolicyService, VacationPolicyRepository
Таблицы БД	VacationPolicies
Основные API	/api/v1/vacation-policies/*
Входные данные	Тип отпуска, период, параметры политики
Выходные данные	Результаты проверки политики
Связи с другими модулями	Используется модулем заявок и конфликтов

Таблица Г.3 – Спецификация модуля Vacation Request Module

Характеристика	Описание
Назначение	Обеспечивает создание и согласование заявок на отпуск
Основные функции	Создание заявки, отправка, согласование, отмена
Основные классы	VacationRequestService, VacationRequestRepository
Таблицы БД	VacationRequests, VacationRequestHistory
Основные API	/api/v1/vacation-requests/*

Продолжение таблицы Г.3

Характеристика	Описание
Входные данные	Данные заявки, сотрудник, даты
Выходные данные	Статус заявки, история
Связи с другими модулями	Использует модули политики, конфликтов и уведомлений

Таблица Г.4 – Спецификация модуля Conflict Detection Module

Характеристика	Описание
Назначение	Выполняет анализ конфликтов отпусков
Основные функции	Проверка пересечений, анализ нагрузки подразделений
Основные классы	ConflictDetectionService, ConflictDetectionRepository
Таблицы БД	Использует VacationRequests, Employees
Основные API	/api/v1/conflicts/*
Входные данные	Сотрудник, даты отпуска
Выходные данные	Результат анализа конфликтов
Связи с другими модулями	Использует данные сотрудников, заявок и политик

Таблица Г.5 – Спецификация модуля Schedule Module

Характеристика	Описание
Назначение	Формирует календарь отпусков и отчеты
Основные функции	Получение календаря, истории отпусков
Основные классы	ScheduleService, ScheduleRepository
Таблицы БД	Использует VacationRequests, Employees
Основные API	/api/v1/schedules/*
Входные данные	Период, сотрудник
Выходные данные	Календарь, история
Связи с другими модулями	Использует данные заявок и сотрудников

Таблица Г.6 – Спецификация модуля Notification Module

Характеристика	Описание
Назначение	Управляет уведомлениями пользователей
Основные функции	Создание уведомлений, получение, отметка прочтения
Основные классы	NotificationService, NotificationRepository
Таблицы БД	Notifications
Основные API	/api/v1/notifications/*
Входные данные	Пользователь, сообщение
Выходные данные	Список уведомлений
Связи с другими модулями	Используется всеми модулями для уведомлений

Приложение Д

(обязательное)

Код программ

Листинг Д.1 – AuthController.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using PlanTimeService.Modules.IdentityModule.Application;
using PlanTimeService.Modules.IdentityModule.Application.DTOs;

namespace PlanTimeService.Modules.IdentityModule.Api;

[ApiController]
[Route("api/v1/auth")]
public class AuthController : ControllerBase
{
    private readonly IAuthService _authService;

    public AuthController(IAuthService authService)
    {
        _authService = authService;
    }

    [HttpPost("login")]
    [AllowAnonymous]
    public async Task<IActionResult> Login([FromBody] LoginDto dto,
CancellationToken ct)
    {
        var ip = HttpContext.Connection.RemoteIpAddress?.ToString();
        var userAgent = HttpContext.Request.Headers.UserAgent.FirstOrDefault();
        var result = await _authService.LoginAsync(dto, ip, userAgent, ct);
        if (!result.IsSuccess)
            return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpPost("refresh")]
    [AllowAnonymous]
    public async Task<IActionResult> Refresh([FromBody] RefreshTokenDto dto,
CancellationToken ct)
    {
        var result = await _authService.RefreshAsync(dto.RefreshToken, ct);
        if (!result.IsSuccess)
            return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpPost("logout")]
    [Authorize]
    public async Task<IActionResult> Logout(CancellationToken ct)
    {
        var refreshToken = HttpContext.Request.Headers["X-Refresh-
Token"].FirstOrDefault();
        var userId = GetUserId();
        await _authService.LogoutAsync(refreshToken, userId, ct);
        return NoContent();
    }

    [HttpGet("me")]
    [Authorize]
    public async Task<IActionResult> Me(CancellationToken ct)
    {
        var userId = GetUserId();
        if (userId == null)
            return Unauthorized();
```

```
        var result = await _authService.GetCurrentUserAsync(userId.Value, ct);
        if (!result.IsSuccess)
            return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    private Guid? GetUserId()
    {
        var sub = User.FindFirstValue(ClaimTypes.NameIdentifier);
        return Guid.TryParse(sub, out var id) ? id : null;
    }
}
```

Листинг Д.2 – RolesController.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using PlanTimeService.Modules.IdentityModule.Application;
using PlanTimeService.Modules.IdentityModule.Application.DTOs;

namespace PlanTimeService.Modules.IdentityModule.Api;

[ApiController]
[Route("api/v1/roles")]
[Authorize]
public class RolesController : ControllerBase
{
    private readonly IRoleService _roleService;

    public RolesController(IRoleService roleService)
    {
        _roleService = roleService;
    }

    [HttpGet]
    public async Task<IActionResult> GetAll(CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _roleService.GetAllAsync(userId.Value, roleId!.Value,
ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpPost]
    public async Task<IActionResult> Create([FromBody] CreateRoleDto dto,
CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _roleService.CreateAsync(dto, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return CreatedAtAction(nameof(GetAll), result.Value);
    }

    [HttpDelete("{id:int}")]
    public async Task<IActionResult> Delete(int id, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
```

```
        var result = await _roleService.DeleteAsync(id, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    private (Guid? userId, int? roleId) GetCurrentUser()
    {
        var sub = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var roleIdStr = User.FindFirstValue("role_id");
        var userId = Guid.TryParse(sub, out var uid) ? uid : (Guid?)null;
        var roleId = int.TryParse(roleIdStr, out var rid) ? rid : (int?)null;
        return (userId, roleId);
    }
}
```

Листинг Д.3 – UsersController.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using PlanTimeService.Modules.IdentityModule.Application;
using PlanTimeService.Modules.IdentityModule.Application.DTOs;

namespace PlanTimeService.Modules.IdentityModule.Api;

[ApiController]
[Route("api/v1/users")]
[Authorize]
public class UsersController : ControllerBase
{
    private readonly IUserService _userService;

    public UsersController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpPost]
    public async Task<IActionResult> Create([FromBody] CreateUserDto dto,
CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.CreateAsync(dto, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return CreatedAtAction(nameof(GetById), new { id = result.Value!.Id },
result.Value);
    }

    [HttpGet("{id:guid}")]
    public async Task<IActionResult> GetById(Guid id, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.GetByIdAsync(id, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return NotFound(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpGet]
```

```
    public async Task<IActionResult> GetList(
        [FromQuery] Guid? departmentId,
        [FromQuery] int? role,
        [FromQuery] bool? active,
        [FromQuery] string? search,
        CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.GetListAsync(departmentId, role, active,
search, userId.Value, roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpPut("{id:guid}")]
    public async Task<IActionResult> Update(Guid id, [FromBody] UpdateUserDto
dto, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.UpdateAsync(id, dto, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    [HttpDelete("{id:guid}")]
    public async Task<IActionResult> Delete(Guid id, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.DeleteAsync(id, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    [HttpPatch("{id:guid}/deactivate")]
    public async Task<IActionResult> Deactivate(Guid id, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.DeactivateAsync(id, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    [HttpPatch("{id:guid}/activate")]
    public async Task<IActionResult> Activate(Guid id, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.ActivateAsync(id, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    [HttpPatch("{id:guid}/password")]
    public async Task<IActionResult> ChangePassword(Guid id, [FromBody]
ChangePasswordDto dto, CancellationToken ct)
```

```
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.ChangePasswordAsync(id, dto,
userId.Value, roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    [HttpPatch("{id:guid}/role")]
    public async Task<IActionResult> ChangeRole(Guid id, [FromBody]
ChangeRoleDto dto, CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.ChangeRoleAsync(id, dto, userId.Value,
roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return NoContent();
    }

    [HttpGet("department/{departmentId:guid}")]
    public async Task<IActionResult> GetByDepartment(Guid departmentId,
CancellationToken ct)
    {
        var (userId, roleId) = GetCurrentUser();
        if (userId == null) return Unauthorized();
        var result = await _userService.GetByDepartmentAsync(departmentId,
userId.Value, roleId!.Value, ct);
        if (!result.IsSuccess) return BadRequest(new { error = result.Error });
        return Ok(result.Value);
    }

    private (Guid? userId, int? roleId) GetCurrentUser()
    {
        var sub = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var roleIdStr = User.FindFirstValue("role_id");
        var userId = Guid.TryParse(sub, out var uid) ? uid : (Guid?)null;
        var roleId = int.TryParse(roleIdStr, out var rid) ? rid : (int?)null;
        return (userId, roleId);
    }
}
```

Листинг Д.4 – RoleRepository.cs

```
using Dapper;
using PlanTimeService.Modules.IdentityModule.Domain;

namespace PlanTimeService.Modules.IdentityModule.Infrastructure.Repositories;

public class RoleRepository : IRoleRepository
{
    private readonly DapperContext _context;

    public RoleRepository(DapperContext context)
    {
        _context = context;
    }

    public async Task<IReadOnlyList<Role>> GetAllAsync(CancellationToken ct =
default)
    {
        using var conn = _context.CreateConnection();
```

```
        var list = await conn.QueryAsync<Role>(new CommandDefinition(
            "SELECT \"Id\", \"Name\", \"Description\" FROM \"Roles\" ORDER BY
\"Id\"",
            cancellationToken: ct));
        return list.ToList();
    }

    public async Task<Role?> GetByIdAsync(int id, CancellationToken ct =
default)
    {
        using var conn = _context.CreateConnection();
        return await conn.QuerySingleOrDefaultAsync<Role>(new CommandDefinition(
            "SELECT \"Id\", \"Name\", \"Description\" FROM \"Roles\" WHERE
\"Id\" = @Id",
            new { Id = id },
            cancellationToken: ct));
    }

    public async Task<Role?> GetByNameAsync(string name, CancellationToken ct =
default)
    {
        using var conn = _context.CreateConnection();
        return await conn.QuerySingleOrDefaultAsync<Role>(new CommandDefinition(
            "SELECT \"Id\", \"Name\", \"Description\" FROM \"Roles\" WHERE
\"Name\" = @Name",
            new { Name = name },
            cancellationToken: ct));
    }

    public async Task<int> CreateAsync(Role role, CancellationToken ct =
default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "INSERT INTO \"Roles\" (\"Id\", \"Name\", \"Description\") VALUES
(@Id, @Name, @Description)",
            role,
            cancellationToken: ct));
    }

    public async Task<int> DeleteAsync(int id, CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "DELETE FROM \"Roles\" WHERE \"Id\" = @Id",
            new { Id = id },
            cancellationToken: ct));
    }

    public async Task<int> GetNextIdAsync(CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        var max = await conn.ExecuteScalarAsync<int?>(new CommandDefinition(
            "SELECT COALESCE(MAX(\"Id\"), 0) FROM \"Roles\"",
            cancellationToken: ct));
        return max.GetValueOrDefault(0) + 1;
    }

    public async Task<bool> HasUsersWithRoleAsync(int roleId, CancellationToken
ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteScalarAsync<bool>(new CommandDefinition(
```

```
            "SELECT EXISTS(SELECT 1 FROM \"Users\" WHERE \"RoleId\" = @RoleId
AND \"IsDeleted\" = false)",
            new { RoleId = roleId },
            cancellationToken: ct));
    }
}
```

Листинг Д.5 - UserRepository.cs

```
using Dapper;
using PlanTimeService.Modules.IdentityModule.Domain;

namespace PlanTimeService.Modules.IdentityModule.Infrastructure.Repositories;

public class UserRepository : IUserRepository
{
    private readonly DapperContext _context;

    public UserRepository(DapperContext context)
    {
        _context = context;
    }

    public async Task<User?> GetByEmailAsync(string email, bool includeDeleted =
false, CancellationToken ct = default)
    {
        var sql = """
            SELECT u."Id", u."Email", u."PasswordHash", u."FirstName",
u."LastName", u."MiddleName", u."Phone",
                   u."Position", u."DepartmentId", u."RoleId", u."IsActive",
u."IsDeleted", u."CreatedAt", u."UpdatedAt",
                   r."Name" AS "RoleName"
            FROM "Users" u
            INNER JOIN "Roles" r ON r."Id" = u."RoleId"
            WHERE u."Email" = @Email
            """ + (includeDeleted ? "" : " AND u.\"IsDeleted\" = false");
        using var conn = _context.CreateConnection();
        var row = await conn.QuerySingleOrDefaultAsync<UserWithRole>(new
CommandDefinition(sql, new { Email = email }, cancellationToken: ct));
        return row == null ? null : ToUser(row);
    }

    public async Task<UserWithRole?> GetByIdWithRoleAsync(Guid id, bool
includeDeleted = false, CancellationToken ct = default)
    {
        var sql = """
            SELECT u."Id", u."Email", u."PasswordHash", u."FirstName",
u."LastName", u."MiddleName", u."Phone",
                   u."Position", u."DepartmentId", u."RoleId", u."IsActive",
u."IsDeleted", u."CreatedAt", u."UpdatedAt",
                   r."Name" AS "RoleName"
            FROM "Users" u
            INNER JOIN "Roles" r ON r."Id" = u."RoleId"
            WHERE u."Id" = @Id
            """ + (includeDeleted ? "" : " AND u.\"IsDeleted\" = false");
        using var conn = _context.CreateConnection();
        return await conn.QuerySingleOrDefaultAsync<UserWithRole>(new
CommandDefinition(sql, new { Id = id }, cancellationToken: ct));
    }

    public async Task<bool> ExistsByEmailAsync(string email, CancellationToken
ct = default)
    {
```

```
        using var conn = _context.CreateConnection();
        return await conn.ExecuteScalarAsync<bool>(new CommandDefinition(
            "SELECT EXISTS(SELECT 1 FROM \"Users\" WHERE \"Email\" = @Email AND
\"IsDeleted\" = false)",
            new { Email = email },
            cancellationToken: ct));
    }

    public async Task<int> CreateAsync(User user, CancellationToken ct =
default)
    {
        var sql = """
            INSERT INTO "Users" ("Id", "Email", "PasswordHash", "FirstName",
"LastName", "MiddleName", "Phone", "Position",
                "DepartmentId", "RoleId", "IsActive", "IsDeleted", "CreatedAt",
"UpdatedAt")
            VALUES (@Id, @Email, @PasswordHash, @FirstName, @LastName,
@MiddleName, @Phone, @Position,
                @DepartmentId, @RoleId, @IsActive, @IsDeleted, @CreatedAt,
@UpdatedAt)
            """;
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(sql, user,
cancellationToken: ct));
    }

    public async Task<int> UpdateAsync(User user, CancellationToken ct =
default)
    {
        var sql = """
            UPDATE "Users" SET "FirstName" = @FirstName, "LastName" = @LastName,
"MiddleName" = @MiddleName,
                "Phone" = @Phone, "Position" = @Position, "DepartmentId" =
@DepartmentId, "UpdatedAt" = @UpdatedAt
            WHERE "Id" = @Id
            """;
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(sql, user,
cancellationToken: ct));
    }

    public async Task<int> SetDeletedAsync(Guid id, bool isDeleted,
CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "UPDATE \"Users\" SET \"IsDeleted\" = @IsDeleted, \"UpdatedAt\" =
@UpdatedAt WHERE \"Id\" = @Id",
            new { Id = id, IsDeleted = isDeleted, UpdatedAt = DateTime.UtcNow },
            cancellationToken: ct));
    }

    public async Task<int> SetActiveAsync(Guid id, bool isActive,
CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "UPDATE \"Users\" SET \"IsActive\" = @IsActive, \"UpdatedAt\" =
@UpdatedAt WHERE \"Id\" = @Id",
            new { Id = id, IsActive = isActive, UpdatedAt = DateTime.UtcNow },
            cancellationToken: ct));
    }
```

```
    public async Task<int> UpdatePasswordAsync(Guid id, string passwordHash,
CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "UPDATE \"Users\" SET \"PasswordHash\" = @PasswordHash,
\"UpdatedAt\" = @UpdatedAt WHERE \"Id\" = @Id",
            new { Id = id, PasswordHash = passwordHash, UpdatedAt =
DateTime.UtcNow },
            cancellationToken: ct));
    }

    public async Task<int> UpdateRoleAsync(Guid id, int roleId,
CancellationToken ct = default)
    {
        using var conn = _context.CreateConnection();
        return await conn.ExecuteAsync(new CommandDefinition(
            "UPDATE \"Users\" SET \"RoleId\" = @RoleId, \"UpdatedAt\" =
@UpdatedAt WHERE \"Id\" = @Id",
            new { Id = id, RoleId = roleId, UpdatedAt = DateTime.UtcNow },
            cancellationToken: ct));
    }

    public async Task<IReadOnlyList<UserWithRole>> GetListAsync(Guid?
departmentId, int? roleId, bool? isActive, string? search, CancellationToken ct
= default)
    {
        var sql = """
            SELECT u."Id", u."Email", u."PasswordHash", u."FirstName",
u."LastName", u."MiddleName", u."Phone",
                   u."Position", u."DepartmentId", u."RoleId", u."IsActive",
u."IsDeleted", u."CreatedAt", u."UpdatedAt",
                   r."Name" AS "RoleName"
            FROM "Users" u
            INNER JOIN "Roles" r ON r."Id" = u."RoleId"
            WHERE u."IsDeleted" = false
            """;
        var dyn = new DynamicParameters();
        if (departmentId.HasValue) { sql += " AND u.\"DepartmentId\" =
@DepartmentId"; dyn.Add("DepartmentId", departmentId.Value); }
        if (roleId.HasValue) { sql += " AND u.\"RoleId\" = @RoleId";
dyn.Add("RoleId", roleId.Value); }
        if (isActive.HasValue) { sql += " AND u.\"IsActive\" = @IsActive";
dyn.Add("IsActive", isActive.Value); }
        if (!string.IsNullOrWhiteSpace(search))
        {
            var s = $"%{search.Trim()}%";
            sql += " AND (LOWER(u.\"Email\") LIKE @Search OR
LOWER(u.\"FirstName\" || ' ' || u.\"LastName\") LIKE @Search OR
LOWER(u.\"LastName\" || ' ' || u.\"FirstName\") LIKE @Search)";
            dyn.Add("Search", s);
        }
        sql += " ORDER BY u.\"LastName\", u.\"FirstName\"";
        using var conn = _context.CreateConnection();
        var list = await conn.QueryAsync<UserWithRole>(new
CommandDefinition(sql, dyn, cancellationToken: ct));
        return list.ToList();
    }

    public async Task<IReadOnlyList<UserWithRole>> GetByDepartmentAsync(Guid
departmentId, CancellationToken ct = default)
    {
        var sql = """
```

```
            SELECT u."Id", u."Email", u."PasswordHash", u."FirstName",
u."LastName", u."MiddleName", u."Phone",
                   u."Position", u."DepartmentId", u."RoleId", u."IsActive",
u."IsDeleted", u."CreatedAt", u."UpdatedAt",
                   r."Name" AS "RoleName"
            FROM "Users" u
            INNER JOIN "Roles" r ON r."Id" = u."RoleId"
            WHERE u."IsDeleted" = false AND u."DepartmentId" = @DepartmentId
            ORDER BY u."LastName", u."FirstName"
            """;
        using var conn = _context.CreateConnection();
        var list = await conn.QueryAsync<UserWithRole>(new
CommandDefinition(sql, new { DepartmentId = departmentId }, cancellationToken:
ct));
        return list.ToList();
    }

    private static User ToUser(UserWithRole r) => new()
    {
        Id = r.Id,
        Email = r.Email,
        PasswordHash = r.PasswordHash,
        FirstName = r.FirstName,
        LastName = r.LastName,
        MiddleName = r.MiddleName,
        Phone = r.Phone,
        Position = r.Position,
        DepartmentId = r.DepartmentId,
        RoleId = r.RoleId,
        IsActive = r.IsActive,
        IsDeleted = r.IsDeleted,
        CreatedAt = r.CreatedAt,
        UpdatedAt = r.UpdatedAt,
        Role = new Role { Id = r.RoleId, Name = r.RoleName }
    };
}
```

Приложение Е

(рекомендуемое)

Результаты тестирования черным ящиком

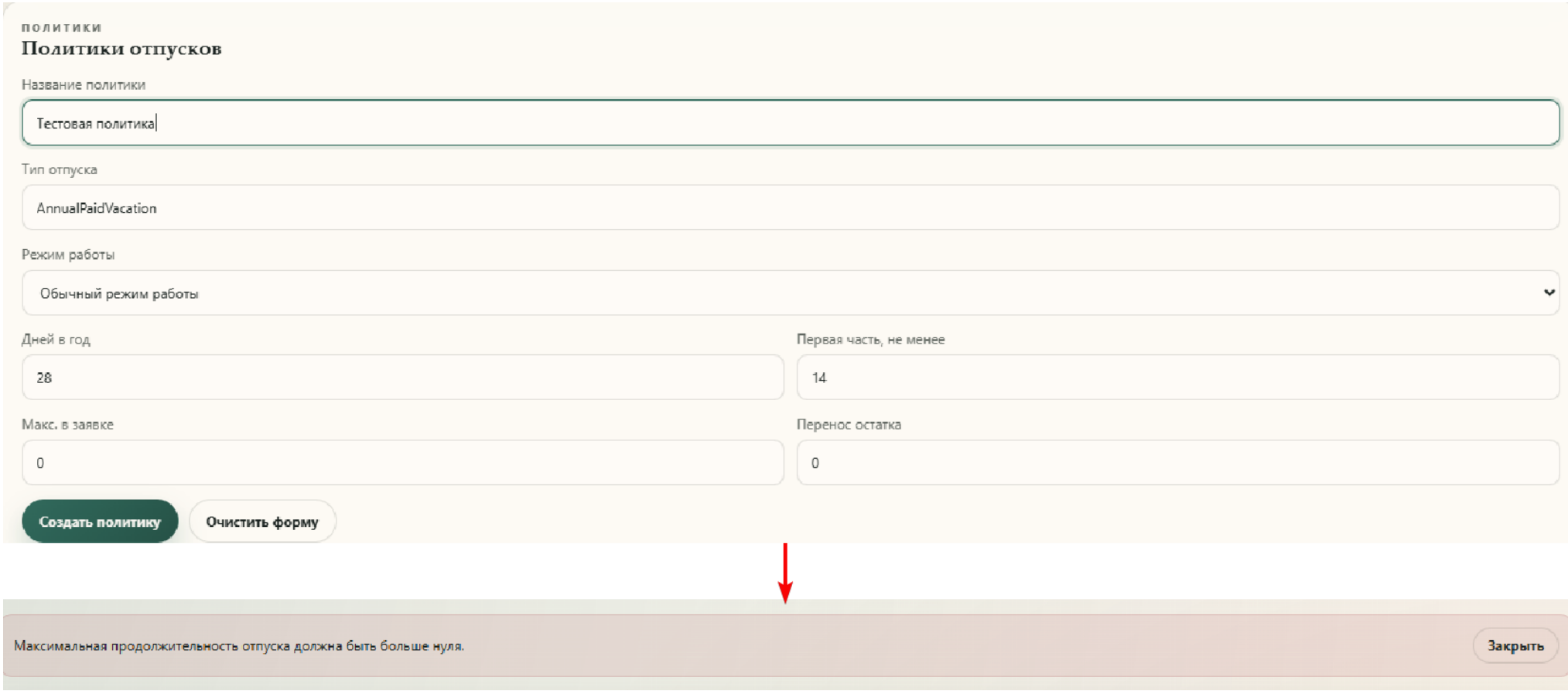

Рисунок Е.1 – Ошибка при вводе нулевой максимальной продолжительности отпуска

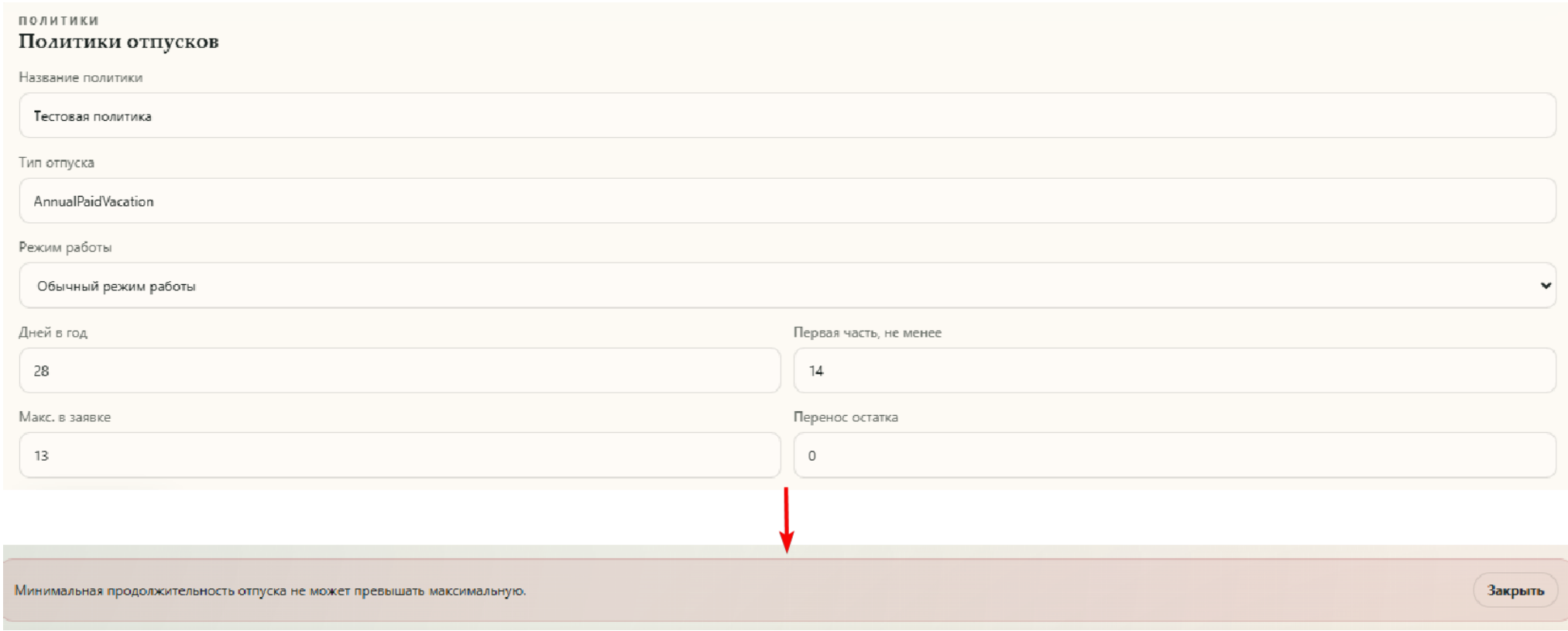

Рисунок Е.2 – Ошибка при вводе значения меньше минимальной продолжительности отпуск

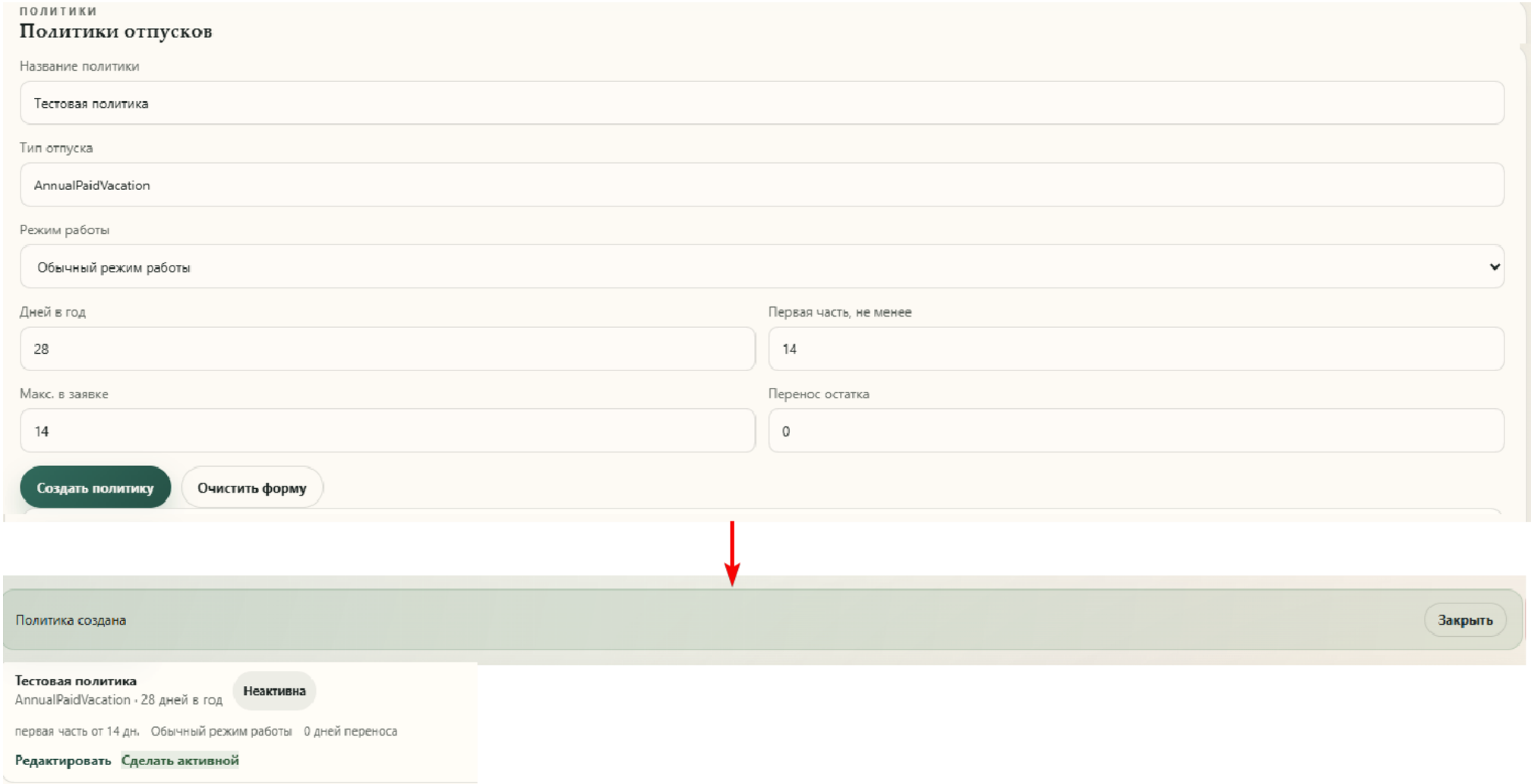

Рисунок Е.3 – Успешное создание политики при нижней допустимой границе

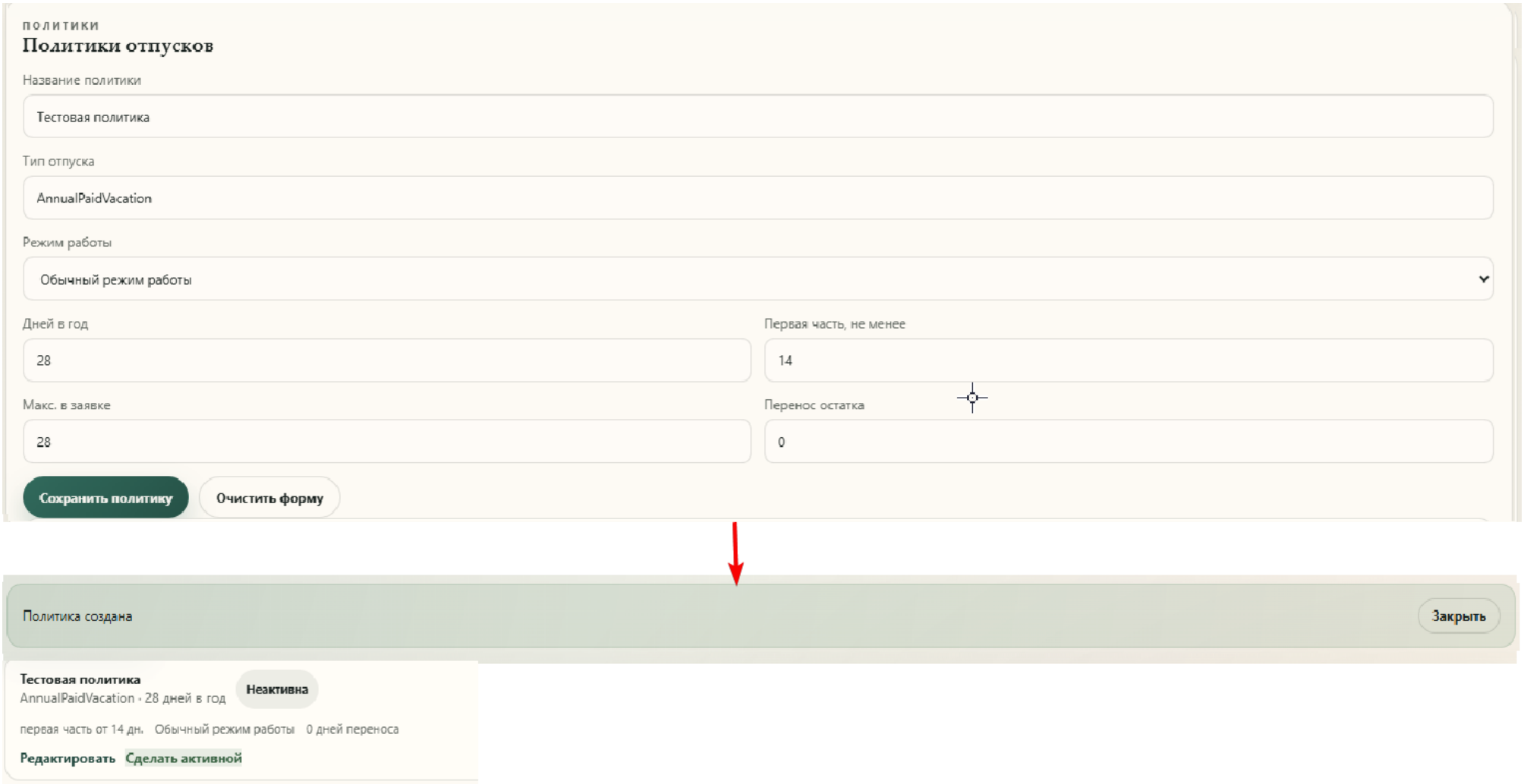

Рисунок Е.4 – Успешное создание политики при верхней допустимой границе

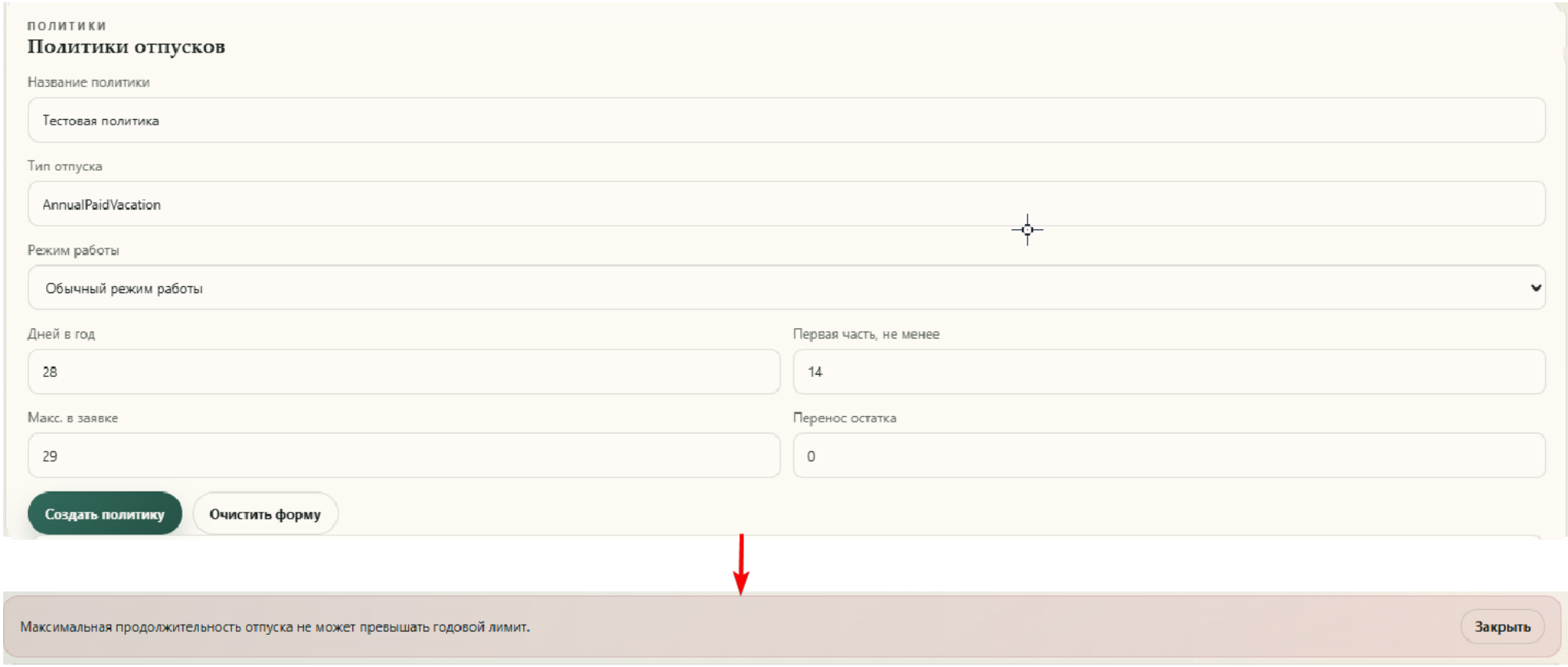

Рисунок Е.5 – Ошибка при превышении годового лимита отпуска

Приложение Ж

(обязательное)

Код юнит-тестов

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Moq;
using PlanTimeService.Modules.ConflictDetectionModule.Api;
using PlanTimeService.Modules.ConflictDetectionModule.Application;
using PlanTimeService.Modules.ConflictDetectionModule.Application.DTOs;
using PlanTimeService.Modules.ConflictDetectionModule.Domain;
using PlanTimeService.Modules.IdentityModule.Api;
using PlanTimeService.Modules.IdentityModule.Application;
using PlanTimeService.Modules.IdentityModule.Application.DTOs;
using PlanTimeService.Modules.NotificationModule.Api;
using PlanTimeService.Modules.NotificationModule.Application;
using PlanTimeService.Modules.NotificationModule.Application.DTOs;
using PlanTimeService.Modules.VacationPolicyModule.Api;
using PlanTimeService.Modules.VacationPolicyModule.Application;
using PlanTimeService.Modules.VacationPolicyModule.Application.DTOs;
using PlanTimeService.Modules.VacationPolicyModule.Domain;
using PlanTimeService.Modules.VacationRequestModule.Api;
using PlanTimeService.Modules.VacationRequestModule.Application;
using PlanTimeService.Modules.VacationRequestModule.Application.DTOs;
using PlanTimeService.Modules.VacationRequestModule.Domain;
using Xunit;

namespace PlanTimeService.Tests.Controllers;

public class VacationRequestsControllerTests
{
    private readonly Mock<IVacationRequestService> _serviceMock = new();
    private readonly VacationRequestsController _controller;

    public VacationRequestsControllerTests()
    {
        _controller = new VacationRequestsController(_serviceMock.Object);
    }

    [Fact]
    public async Task Create_ValidRequest_ReturnsCreatedAtAction()
    {
        // Проверяет, что при успешном создании заявки контроллер возвращает
CreatedAtAction с созданной сущностью.
        var dto = new CreateVacationRequestDto(
            Guid.NewGuid(),
            VacationTypes.AnnualPaidVacation,
            new DateTime(2026, 7, 1),
            new DateTime(2026, 7, 10),
            "Vacation");
        var request = CreateVacationRequest(dto.EmployeeId);

        _serviceMock
            .Setup(service => service.CreateAsync(dto,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<VacationRequest>.Ok(request));

        var result = await _controller.Create(dto, CancellationToken.None);

        var createdResult = Assert.IsType<CreatedAtActionResult>(result);
        var returnRequest = Assert.IsType<VacationRequest>(createdResult.Value);
        Assert.Equal(request.Id, returnRequest.Id);
    }

    [Fact]
    public async Task GetById_RequestNotFound_ReturnsNotFound()
    {
```

```
        // Проверяет, что при отсутствии заявки по идентификатору контроллер
возвращает 404 NotFound.
        var id = Guid.NewGuid();

        _serviceMock
            .Setup(service => service.GetByIdAsync(id,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<VacationRequest>.Fail("Request not found"));

        var result = await _controller.GetById(id, CancellationToken.None);

        Assert.IsType<NotFoundObjectResult>(result);
    }

    private static VacationRequest CreateVacationRequest(Guid employeeId) =>
new()
    {
        Id = Guid.NewGuid(),
        EmployeeId = employeeId,
        OrganizationId = Guid.NewGuid(),
        DepartmentId = Guid.NewGuid(),
        PositionId = Guid.NewGuid(),
        VacationType = VacationTypes.AnnualPaidVacation,
        StartDate = new DateTime(2026, 7, 1),
        EndDate = new DateTime(2026, 7, 10),
        Status = VacationRequestStatuses.Draft,
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };
}

public class VacationPoliciesControllerTests
{
    private readonly Mock<IVacationPolicyService> _serviceMock = new();
    private readonly VacationPoliciesController _controller;

    public VacationPoliciesControllerTests()
    {
        _controller = new VacationPoliciesController(_serviceMock.Object);
    }

    [Fact]
    public async Task Create_ValidPolicy_ReturnsCreatedAtAction()
    {
        // Проверяет, что при успешном создании политики контроллер возвращает
CreatedAtAction с созданной политикой.
        var dto = new CreateVacationPolicyDto(
            Guid.NewGuid(),
            "Annual vacation",
            VacationTypes.AnnualPaidVacation,
            VacationWorkModes.Normal,
            28,
            14,
            28,
            0);
        var policy = new VacationPolicy
        {
            Id = Guid.NewGuid(),
            OrganizationId = dto.OrganizationId,
            Name = dto.Name,
            VacationType = dto.VacationType,
            WorkMode = dto.WorkMode,
            AnnualVacationDays = dto.AnnualVacationDays,
```

```
            MinVacationDaysPerRequest = dto.MinVacationDaysPerRequest,
            MaxVacationDaysPerRequest = dto.MaxVacationDaysPerRequest,
            MaxCarryOverDays = dto.MaxCarryOverDays
        };

        _serviceMock
            .Setup(service => service.CreateAsync(dto,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<VacationPolicy>.Ok(policy));

        var result = await _controller.Create(dto, CancellationToken.None);

        var createdResult = Assert.IsType<CreatedAtActionResult>(result);
        var returnPolicy = Assert.IsType<VacationPolicy>(createdResult.Value);
        Assert.Equal("Annual vacation", returnPolicy.Name);
    }

    [Fact]
    public async Task CalculateAvailableDays_ReturnsOk_WithAvailableDays()
    {
        // Проверяет, что контроллер корректно возвращает рассчитанное
количество доступных дней отпуска.
        var employeeId = Guid.NewGuid();

        _serviceMock
            .Setup(service => service.CalculateAvailableDaysAsync(
                It.Is<CalculateAvailableVacationDaysDto>(dto => dto.EmployeeId
== employeeId && dto.Year == 2026),
                It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<int>.Ok(18));

        var result = await _controller.CalculateAvailableDays(
            employeeId,
            2026,
            VacationTypes.AnnualPaidVacation,
            CancellationToken.None);

        var okResult = Assert.IsType<OkObjectResult>(result);
        var availableDays =
okResult.Value!.GetType().GetProperty("availableDays")!.GetValue(okResult.Value)
;
        Assert.Equal(18, availableDays);
    }
}

public class ConflictDetectionControllerTests
{
    private readonly Mock<IConflictDetectionService> _serviceMock = new();
    private readonly ConflictDetectionController _controller;

    public ConflictDetectionControllerTests()
    {
        _controller = new ConflictDetectionController(_serviceMock.Object);
    }

    [Fact]
    public async Task Check_ReturnsOk_WithConflictAnalysis()
    {
        // Проверяет, что контроллер возвращает 200 OK и сам результат анализа
конфликтов при успешной проверке.
        var dto = new CheckVacationConflictDto(
            Guid.NewGuid(),
            new DateTime(2026, 7, 1),
```

```
            new DateTime(2026, 7, 10),
            VacationTypes.AnnualPaidVacation,
            null);
        var analysis = new VacationConflictAnalysis
        {
            EmployeeId = dto.EmployeeId,
            DepartmentId = Guid.NewGuid(),
            PositionId = Guid.NewGuid(),
            StartDate = dto.StartDate,
            EndDate = dto.EndDate,
            HasBlockingConflicts = false
        };
        _serviceMock
            .Setup(service => service.CheckConflictsAsync(dto,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<VacationConflictAnalysis>.Ok(analysis));
        var result = await _controller.Check(dto, CancellationToken.None);
        var okResult = Assert.IsType<OkObjectResult>(result);
        Assert.Equal(analysis, okResult.Value);
    }
    [Fact]
    public async Task AnalyzeRequest_RequestNotFound_ReturnsNotFound()
    {
        var requestId = Guid.NewGuid();
        _serviceMock
            .Setup(service => service.AnalyzeRequestAsync(requestId,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<VacationConflictAnalysis>.Fail("Request not
found"));
        var result = await _controller.AnalyzeRequest(requestId,
CancellationToken.None);

        Assert.IsType<NotFoundObjectResult>(result);
    }
}
public class AuthControllerTests
{
    private readonly Mock<IAuthService> _authServiceMock = new();
    private readonly AuthController _controller;
    public AuthControllerTests()
    {
        _controller = new AuthController(_authServiceMock.Object)
        {
            ControllerContext = new ControllerContext
            {
                HttpContext = new DefaultHttpContext()
            }
        };
    }
    [Fact]
    public async Task Login_ValidCredentials_ReturnsOk()
    {
        // Проверяет, что при валидных учетных данных контроллер возвращает 200
OK с токенами.
        var dto = new LoginDto("user@example.com", "password");
        var token = new TokenResponseDto("access-token", "refresh-token",
DateTime.UtcNow.AddMinutes(15));

        _authServiceMock
            .Setup(service => service.LoginAsync(dto, It.IsAny<string?>(),
It.IsAny<string?>(), It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<TokenResponseDto>.Ok(token));
        var result = await _controller.Login(dto, CancellationToken.None);
```

```
        var okResult = Assert.IsType<OkObjectResult>(result);
        var returnToken = Assert.IsType<TokenResponseDto>(okResult.Value);
        Assert.Equal("access-token", returnToken.AccessToken);
    }
    [Fact]
    public async Task Refresh_InvalidToken_ReturnsBadRequest()
    {
       BadRequest.
        var dto = new RefreshTokenDto("bad-refresh-token");
        _authServiceMock
            .Setup(service => service.RefreshAsync(dto.RefreshToken,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<TokenResponseDto>.Fail("Invalid refresh
token"));
        var result = await _controller.Refresh(dto, CancellationToken.None);
        Assert.IsType<BadRequestObjectResult>(result);
    }
}
public class NotificationsControllerTests
{
    private readonly Mock<INotificationService> _serviceMock = new();
    private readonly NotificationsController _controller;
    public NotificationsControllerTests()
    {
        _controller = new NotificationsController(_serviceMock.Object);
    }
    [Fact]
    public async Task GetUnreadCount_ReturnsOk_WithUnreadCount()
    {
        // Проверяет, что контроллер возвращает 200 OK и корректное число
непрочитанных уведомлений.
        var userId = Guid.NewGuid();

        _serviceMock
            .Setup(service => service.GetUnreadCountAsync(userId,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result<int>.Ok(3));

        var result = await _controller.GetUnreadCount(userId,
CancellationToken.None);

        var okResult = Assert.IsType<OkObjectResult>(result);
        var unreadCount =
okResult.Value!.GetType().GetProperty("unreadCount")!.GetValue(okResult.Value);
        Assert.Equal(3, unreadCount);
    }
    [Fact]
    public async Task MarkAsRead_Success_ReturnsNoContent()
    {
            var userId = Guid.NewGuid();
        var notificationId = Guid.NewGuid();
        var dto = new MarkNotificationReadDto(userId);
        _serviceMock
            .Setup(service => service.MarkAsReadAsync(userId, notificationId,
It.IsAny<CancellationToken>()))
            .ReturnsAsync(Result.Ok());
        var result = await _controller.MarkAsRead(notificationId, dto,
CancellationToken.None);
        Assert.IsType<NoContentResult>(result);
    }
}
```