

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

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

Студент _____
(подпись, дата) Васильев Дмитрий Александрович
(ФИО полностью)

Руководитель _____
(подпись, дата) Гурьянов Л.В.
(фамилия, инициалы)

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

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

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

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

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

Пенза, 2026

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

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

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
В ФОРМЕ СТАРТАП-ПРОЕКТА**

1. Студент Васильев Дмитрий Александрович гр. 22ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Бизнес-проект «Информационная система планирования отпусков.

Клиентская часть информационной системы контроля отпусков»

Тема утверждена приказом ПГУ № _____ от “___” _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Гурьянов Л.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 система Windows

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

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

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

Основные функции приложения:

а) регистрация и авторизация работников;

б) управление пользователями и организационной структурой предприятия;

в) создание и обработка заявок на отпуск;

г) согласование и управление отпусками;

д) настройка отпускной политики;

е) расчет отпускных дней и проверка ограничений;

<i>ж) выявление конфликтов при планировании;</i>
<i>и) просмотр календаря отпусков;</i>
<i>к) массовый импорт сотрудников из Excel;</i>
<i>л) формирование отчетов по отпускам в компании.</i>

5. Перечень подлежащих разработке вопросов

<i>1) Анализ предметной области и постановка задачи</i>
<i>2) Проектирование программных средств</i>
<i>3) Реализация программных средств</i>
<i>4) Контроль качества программного обеспечения</i>
<i>5) Оформление пояснительной записки</i>

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 154 листа, 35 рисунков, 66 таблиц, 31 использованных источников, 4 приложения.

КЛИЕНТСКАЯ ЧАСТЬ, СИСТЕМА КОНТРОЛЯ ОТПУСКОВ, ВЕБ-ПРИЛОЖЕНИЕ, REACT, TYPESCRIPT, REDUX TOOLKIT, MATERIAL UI, AXIOS, JWT, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, SPA-ПРИЛОЖЕНИЕ, UML, C4, DOCKER, API-ВЗАИМОДЕЙСТВИЕ

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

Цель работы – разработка клиентской части системы контроля отпусков, обеспечивающей взаимодействие пользователей с приложением, обработку данных, управление интерфейсом и обмен данными с серверной частью посредством API.

Клиентская часть реализована на языке TypeScript с использованием библиотеки React. Для управления состоянием приложения используется Redux Toolkit, а для взаимодействия с серверной частью применяется библиотека Axios. Построение пользовательского интерфейса осуществляется с использованием Material UI.

Программное обеспечение разработано по принципу клиент-серверной архитектуры. Клиентская часть представляет собой одностраничное веб-приложение, обеспечивающее динамическое обновление интерфейса без полной перезагрузки страницы. Приложение предназначено для использования через современные веб-браузеры, включая Yandex Browser, Google Chrome и Edge.

					ПГУ 09.03.04 – 08КР221.04			
Изм.	Лист	№ докум.	Подп.	Дата	Бизнес-проект «Информационная система планирования отпусков. Клиентская часть информационной системы контроля отпусков». Пояснительная записка	Лит.	Лист	Листов
Разраб.		Васильев Д.А.					4	154
Пров.		Гурьянов Л.В.						
Н. контр.		Такташкин Д.В.				Группа 22ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	9
1 Стартап-часть.....	11
1.1 Описание существующей проблемы на рынке	11
1.2 Описание решения проблемы и продуктового решения	13
1.3 Объем, структура и развитие рынка	18
1.4 Анализ конкурентов.....	20
1.5 Бизнес-модель проекта	22
1.6 Команда стартап-проекта	24
1.7 Описание финансовой модели и необходимости привлечения инвестиций.....	25
1.8 Описание развития бизнес-проекта	27
1.9 Описание рисков	28
2 Анализ предметной области и постановка задачи на разработку средств автоматизации контроля отпусков	30
2.1 Анализ предметной области	30
2.2 Анализ программных решений аналогичного назначения.....	31
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.....	42
2.5 Планирование разработки и оценка бюджета.....	43

3 Проектирование клиентской части приложения системы	
контроля отпусков.....	47
3.1 Проектирование базы данных	47
3.1.1 Концептуальное проектирование базы данных.....	47
3.1.2 Логическое проектирование базы данных	50
3.2 Выбор технологий разработки.....	52
3.3 Проектирование общей архитектуры программного обеспечения	55
3.4 Проектирование компонентной структуры.....	58
3.5 Проектирование маршрутизации и навигации	60
3.6 Проектирование взаимодействия с серверной частью	62
3.7 Проектирование управления состоянием приложения.....	64
4 Программная реализация.....	68
4.1 Реализация базы данных	68
4.2 Структура клиентской части.....	71
4.3 Реализация пользовательского интерфейса	74
4.3.1 Страница авторизации.....	74
4.3.2 Личный кабинет сотрудника	75
4.3.3 Форма создания заявки на отпуск.....	78
4.3.4 Календарь отпусков команды.....	79
4.3.5 Интерфейс согласования заявок.....	81
4.3.6 Интерфейс HR-специалиста.....	83
4.3.7 Интерфейс администратора	86
4.4 Реализация взаимодействия с API.....	88
4.4.1 Конфигурация HTTP-клиента	88
4.4.2 Реализация сервисного слоя	90

4.4.3 Обработка ошибок и индикация загрузки	91
4.5 Реализация маршрутизации и защиты маршрутов.....	92
4.6 Реализация управления состоянием.....	95
5 Тестирование системы контроля отпусков	98
5.1 Цели и задачи тестирования	98
5.2 Объект тестирования	98
5.3 Функциональное тестирование	99
5.3.1 Таблица решений	100
5.3.2 Таблица истинности	101
5.3.3 Парное тестирование.....	102
5.4 Тестирование пользовательского интерфейса	106
5.4.1 Цель и методология UI/UX тестирования.....	106
5.4.2 Результаты проверки по критериям.....	106
5.4.3 Выводы по UI/UX тестированию	108
5.5 Оценка качества и метрики клиентской части приложения.....	109
5.5.1 Подход к оценке качества	109
5.5.2 Производительность и время загрузки	109
5.5.3 Стабильность и корректность маршрутизации	110
5.5.4 Поддерживаемость и архитектурные метрики	110
5.5.5 Итоговые метрики качества.....	111
Заключение	113
Список использованных источников	115
Приложение А (рекомендуемое) Спецификация диаграммы вариантов использования.....	119
Приложение Б (рекомендуемое) Спецификация логической модели базы данных	129

Приложение В (рекомендуемое) Спецификация физической модели базы данных	136
Приложение Г (обязательное) Код программ	142

Введение

Разработка информационной системы контроля отпусков обусловлена необходимостью автоматизации управления отпусками в организациях с большим числом сотрудников. В таких предприятиях ручное планирование нередко приводит к ошибкам при расчёте отпускных дней и затрудняет процесс согласования заявок. В условиях цифровизации кадровых процессов возрастает потребность в программном решении, которое обеспечивает не только надёжное хранение и обработку данных на сервере, но и удобный, понятный для пользователя интерфейс.

Клиентская часть информационной системы выступает основным средством взаимодействия пользователя с программным продуктом. Именно через неё сотрудники создают заявки на отпуск, руководители рассматривают и согласуют их, а HR-специалисты управляют отпускными политиками и организационной структурой. От качества реализации клиентской части зависят удобство работы с системой, скорость выполнения повседневных операций и общее восприятие программного продукта.

Данный бизнес-проект направлен на разработку клиентской части веб-приложения, обеспечивающей наглядное представление данных, получаемых от серверной части системы, и предоставляющей пользователям функциональные возможности для планирования, согласования и контроля отпусков. Разрабатываемая система ориентирована на применение в рамках действующего трудового законодательства Российской Федерации [1], что позволяет обеспечить корректность учёта отпускных дней и соблюдение нормативных требований.

Цель работы – разработка клиентской части веб-ориентированной системы, предназначенной для автоматизации планирования и согласования отпусков сотрудников в организациях различного профиля, обеспечивающей удобный доступ к функциям системы через браузер и адаптированной под нужды конечных пользователей.

Для достижения поставленной цели решаются задачи, связанные с анализом предметной области и требований к пользовательскому интерфейсу, проектированием архитектуры клиентского приложения, структуры компонентов, системы маршрутизации и механизмов управления состоянием, разработкой интерфейсов для основных ролей системы, реализацией взаимодействия с серверной частью посредством REST API, а также обеспечением корректной обработки данных на стороне клиента, включая валидацию входных значений, обработку ошибок и отображение состояния выполнения операций.

Разрабатываемая система представляет собой веб-приложение, построенное по принципу Single Page Application. Такой подход предоставляет нам возможность осуществлять переходы между разделами без перезагрузки страницы, благодаря этому повышается комфорт использования.

Клиентская часть обменивается серверной частью посредством REST API, передавая запросы на получение, изменение и сохранение данных. Такая градация системы, на клиентскую и серверную части, дает возможность упростить сопровождение программного продукта и обеспечить независимую разработку отдельных компонентов системы [2].

Данное веб-приложение предназначено для автоматизации процессов планирования и согласования отпусков сотрудников. Разработанное решение может быть использовано в организациях разного масштаба, предоставляя возможность дальнейшего расширения функциональных возможностей.

1 Стартап-часть

1.1 Описание существующей проблемы на рынке

Наиболее актуальной проблемой в сфере управления персоналом является организация процесса учёта и планирования отпусков сотрудников. Во многих компаниях для этих целей продолжают использоваться электронные таблицы или бумажные документы. Подобный подход требует значительных временных затрат со стороны сотрудников кадровых подразделений и руководителей, а также увеличивает вероятность возникновения ошибок при расчёте доступных дней отпуска, согласовании заявок и формировании графиков отпусков.

Следующая проблема имеет особое значение в организациях с большим числом сотрудников, поскольку необходимо брать в расчет загрузку подразделений. 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. Недостаточный контроль замещения сотрудников приводит к понижению продуктивности, нарушению сроков сдачи работ .

Таким образом, существующие способы управления отпусками не всегда позволяют обеспечить удобный контроль заявок и корректное планирование отпускных периодов. Для решения данной проблемы целесообразно использовать информационную систему, которая автоматизирует процессы подачи, согласования и учёта отпусков. Внедрение такого решения позволит сократить количество ошибок, связанных с человеческим фактором, а также повысить эффективность работы сотрудников и кадровых служб.

1.2 Описание решения проблемы и продуктового решения

Для устранения выявленных проблем разрабатывается информационная система контроля отпусков. Система автоматизирует подачу, согласование и учет отпускных заявок, что позволяет сократить время на обработку заявок и снизить количество ошибок при планировании.

Предлагаемое решение закрывает основные недостатки ручного управления – снижает нагрузку на HR-специалистов и руководителей и делает процесс согласования более прозрачным.

Клиентская часть реализована в виде одностраничного веб-приложения. Оно работает в браузере и отображает данные, получаемые от сервера. Через интерфейс пользователи могут создавать и согласовывать заявки на отпуск, управлять отпускными политиками, просматривать календарь и формировать отчеты.

Система предусматривает разграничение доступа в зависимости от роли пользователя. Сотрудники могут создавать и редактировать заявки на отпуск, отслеживать их статус и просматривать остаток доступных отпускных дней. Руководители подразделений получают доступ к инструментам согласования заявок, анализа загрузки сотрудников и просмотра календаря отпусков команды. HR-специалисты осуществляют управление кадровой структурой организации, настройку отпускных политик и импорт данных сотрудников. Администраторы отвечают за управление пользователями, ролями и общей конфигурацией системы.

Одной из особенностей клиентской части является поддержка различных режимов работы организации, включая круглосуточный режим. В данном случае интерфейс создания заявки предоставляет возможность указания замещающего сотрудника, что позволяет учитывать особенности непрерывных рабочих процессов.

Дополнительно система предоставляет инструменты аналитики и отчетности. Пользователи с соответствующими правами могут анализировать

статистику отпусков, просматривать загрузку подразделений и экспортировать данные в формате Excel.

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.

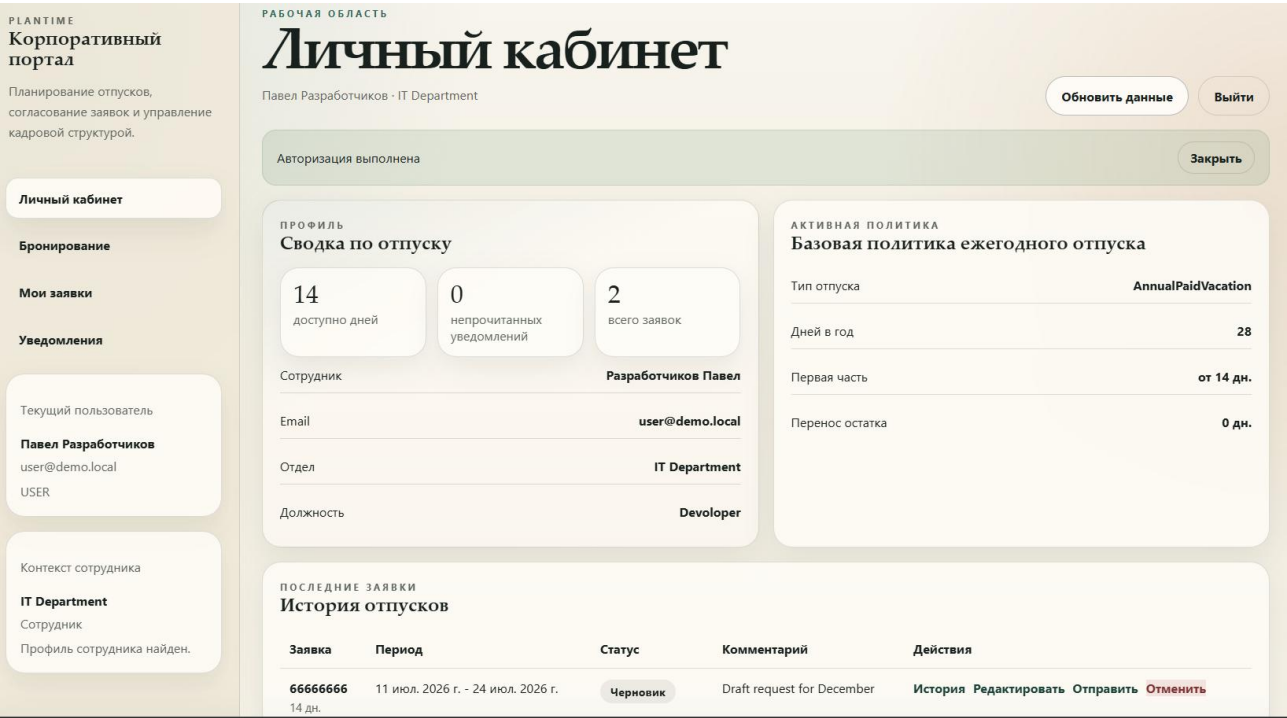


Рисунок 2 – Интерфейс личного кабинета сотрудника

Руководитель подразделения осуществляет согласование заявок сотрудников, анализирует загрузку подразделения и выявляет конфликты отпусков. Также руководителю доступен календарь отпусков команды. Интерфейс функции

Был разработан интерфейс функции «Календарь отпусков команды» в личном кабинете руководителя. Следующее визуальное решение представлено на рисунке 3.

Интерфейс функций формирования отчетов по отпускам и согласования заявок в личном кабинете руководителя приведен на рисунке 4.

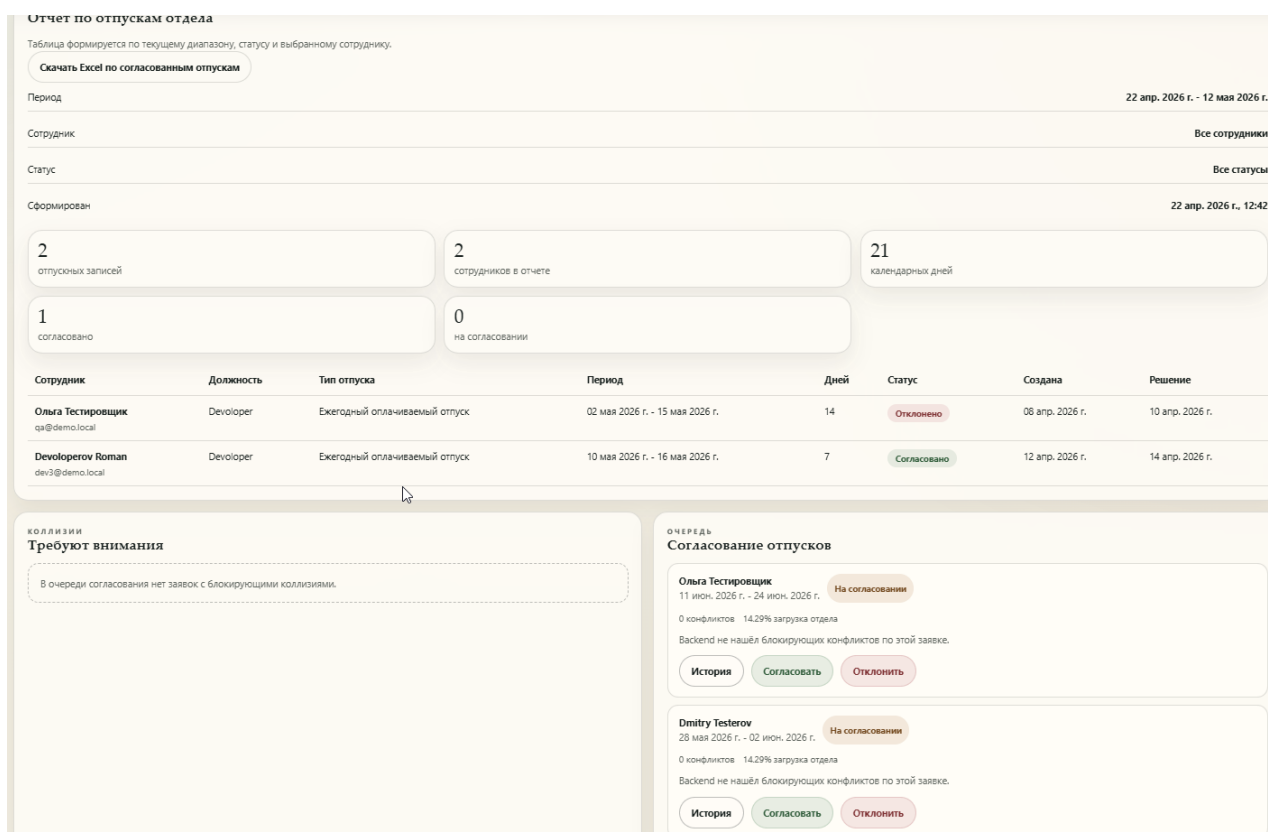


Рисунок 4 – Интерфейс функций формирования отчетов по отпускам и согласования заявок в личном кабинете руководителя

На рисунке 5 представлен интерфейс функции управления отпусками.

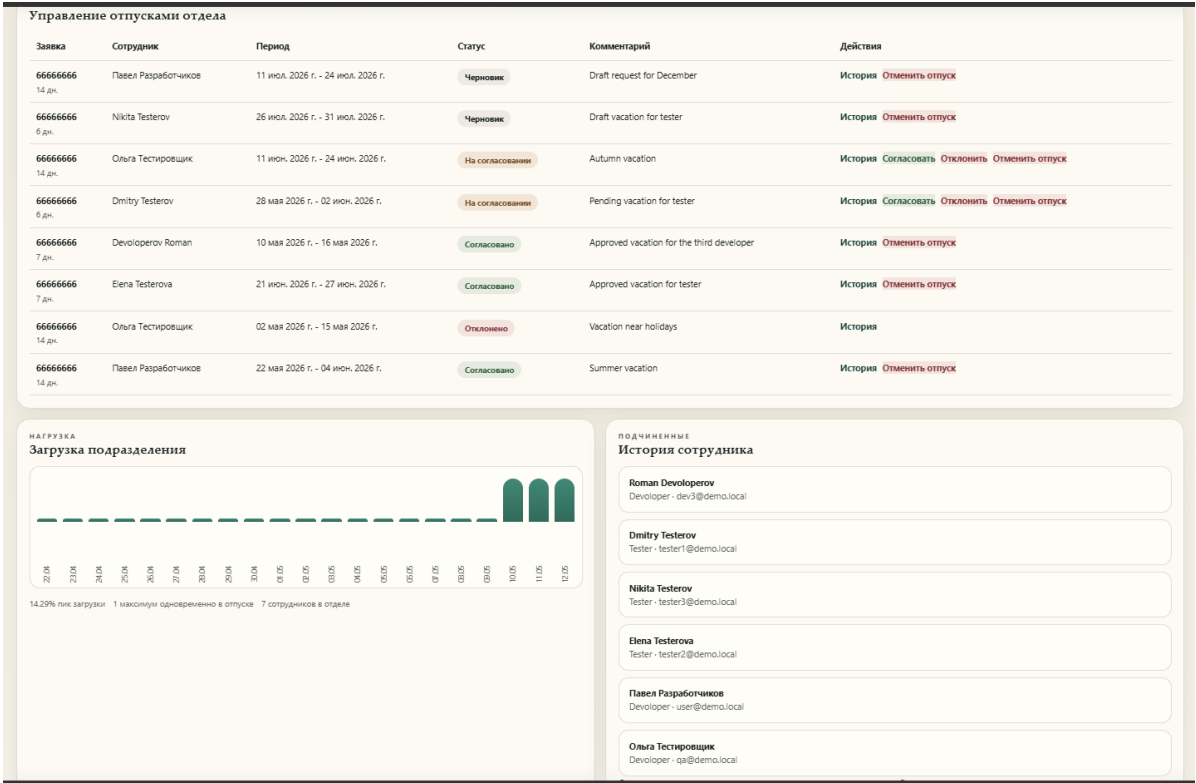


Рисунок 5 – Интерфейс функции управления отпусками и историями сотрудников в личном кабинете руководителя

Система обеспечивает автоматизацию процесса подачи и обработки заявок на отпуск. Пример интерфейса создания заявки представлен на рисунке 6.

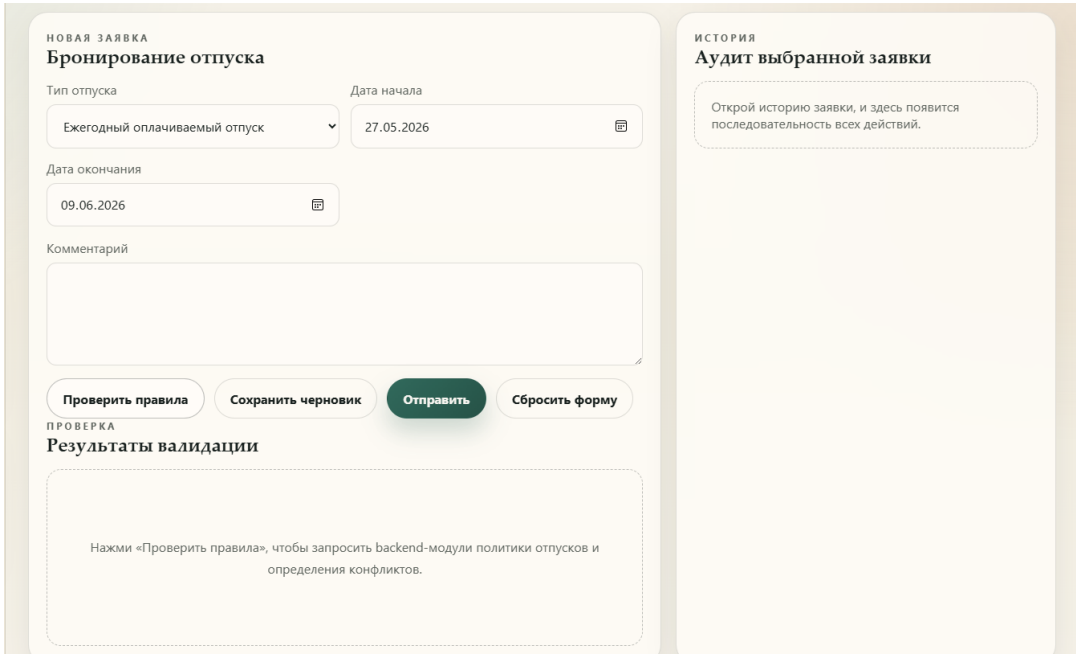


Рисунок 6 – Интерфейс создания заявки на отпуск

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.

политики
Политика отпусков

Название политики

Тип отпуска
AnnualPaidVacation

Режим работы
Обычный режим работы

Дней в год
28

Первая часть, не менее
14

Макс. в заявке
28

Перенос остатка
0

Создать политику Очистить форму

Базовая политика ежегодного отпуска
AnnualPaidVacation - 28 дней в год Активна

первая часть от 14 дн. Обычный режим работы 0 дней переноса

Редактировать Деактивировать

Рисунок 7 – Интерфейс управления отпускными политиками

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

Также решение 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.

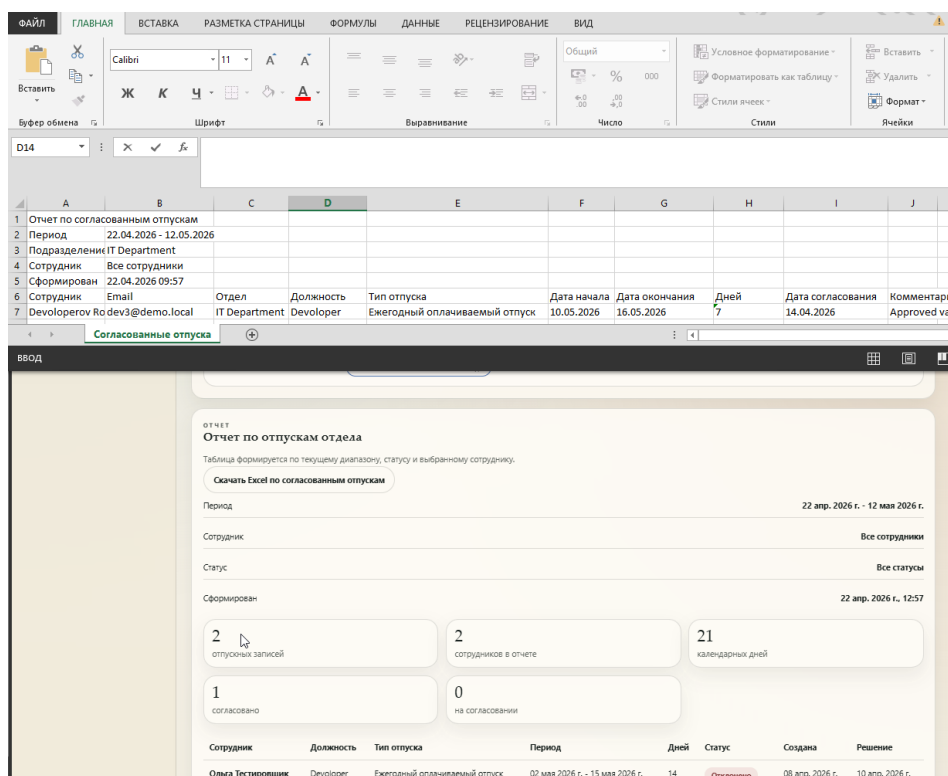


Рисунок 8 – Пример формирования отчета по отпускам

Предлагаемая информационная система представляет собой клиентское веб-приложение, ориентированное на обеспечение удобного и эффективного взаимодействия пользователей с системой управления отпусками. Внедрение данного решения позволит сократить временные затраты на обработку заявок, уменьшить количество ошибок за счёт автоматизированной обработки данных и повысить удобство работы сотрудников с кадровыми процессами организации.

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 нескольких 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.

В таблице 1 один представлена сводка сравнения с конкурентами.

Таблица 1 – Сравнение с конкурентами

Критерий	VambooHR	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

Проведённый анализ показал, что большинство существующих программных решений предназначено для комплексной автоматизации процессов управления персоналом. Такой подход позволяет решать широкий круг задач, однако зачастую сопровождается избыточной функциональностью, сложностью внедрения и необходимостью дополнительной настройки при использовании только для управления отпусками сотрудников.

Разрабатываемая информационная система ориентирована на автоматизацию процессов планирования, согласования и учёта отпусков. Особенностью предлагаемого решения является возможность гибкой настройки отпускных политик организации, учёт различных режимов работы сотрудников, а также контроль пересечений отпускных периодов и замещения персонала. Кроме того, система отличается более простой структурой, что позволяет сократить время внедрения и упростить процесс её освоения пользователями.

Крупные HR-платформы могут со временем добавить похожий функционал, однако разрабатываемая система изначально ориентирована именно на управление отпусками. За счет этого целевые бизнес-процессы реализованы проще и понятнее, а стоимость внедрения и сопровождения ниже, чем у универсальных решений.

В качестве стратегии развития предусмотрена доработка системы под потребности конкретных организаций. При необходимости можно добавить новый функционал или изменить логику работы под требования заказчика.

Система не требует сложной настройки при внедрении. Она подходит для организаций разного профиля, в том числе для предприятий с большим штатом и круглосуточным режимом работы – производственных и военных.

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.

Дополнительным источником дохода может являться разработка и внедрение функциональных возможностей по индивидуальным требованиям заказчиков. При необходимости система может быть адаптирована под особенности конкретной организации, включая настройку бизнес-процессов, расширение существующего функционала и интеграцию с используемыми информационными системами предприятия.

Продвижение программного продукта планируется осуществлять посредством прямого взаимодействия с организациями, заинтересованными в автоматизации процессов управления отпусками сотрудников. Дополнительно возможно размещение информации о системе на специализированных интернет-площадках и ресурсах, посвящённых программному обеспечению для бизнеса. Для повышения узнаваемости продукта могут использоваться инструменты интернет-маркетинга, а также демонстрация возможностей системы потенциальным заказчикам в рамках презентаций и пилотных внедрений. Такой подход позволит получить обратную связь от пользователей и оценить эффективность решения в реальных условиях эксплуатации.

Целевой аудиторией разрабатываемой системы являются организации, в которых процесс согласования и учёта отпусков требует координации большого количества сотрудников. В первую очередь к таким организациям относятся средние и крупные предприятия, коммерческие компании, государственные учреждения и образовательные организации. Для подобных организаций особенно важно обеспечить своевременное планирование отпускных периодов, контроль доступных отпускных дней и предотвращение пересечения отпусков сотрудников, выполняющих взаимосвязанные функции.

Использование специализированной информационной системы позволяет упростить взаимодействие между сотрудниками, руководителями и кадровыми

специалистами, а также повысить прозрачность процессов управления отпусками внутри организации.

Система масштабируется под организации разного размера – от небольших компаний до крупных предприятий с распределенной структурой. Подключение не требует длительной настройки, что сокращает время на внедрение.

Основной поток дохода формируется за счет подписки, дополнительный – за счет доработки функционала под конкретного заказчика. Такое сочетание делает модель устойчивой и не зависящей от разовых продаж.

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.

Расходы проекта делятся на несколько статей: разработка и сопровождение ПО, серверная инфраструктура и продвижение. На начальном этапе разработка ведется силами команды, что снижает стартовые затраты. Ежемесячные расходы на серверную инфраструктуру составят около 30 000 рублей, на маркетинг – около 50 000 рублей.

В первые полгода выручка будет небольшой – этот период отводится на тестирование продукта и формирование клиентской базы. В течение последующих двух лет планируется рост числа клиентов и постепенное увеличение выручки. После закрепления на рынке проект должен 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;
- 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.

Одним из ключевых преимуществ системы является архитектура модульного монолита. Каждый модуль отвечает за свою область – пользователи, заявки, политики – и не зависит от остальных. Это упрощает добавление нового функционала и сопровождение кода. При росте нагрузки система может быть переведена на микросервисную архитектуру без существенной переработки.

Основные целевые показатели проекта – рост числа пользователей и подключенных организаций. На начальном этапе планируется привлечение первых клиентов и тестирование продукта в реальных условиях. Далее предполагается постепенное расширение клиентской базы.

В первые месяцы основной акцент делается на доработке продукта и привлечении пользователей. По мере появления отзывов и расширения функционала ожидается рост числа клиентов. В долгосрочной перспективе возможен выход на новые сегменты рынка и адаптация системы под разные типы организаций.

Проект имеет готовую серверную и клиентскую часть, гибкую архитектуру и возможность масштабирования. Это создает основу для успешного выхода на рынок и дальнейшего развития продукта.

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, сбой, рост нагрузки	Средняя	Высокое	Использование проверенных технологий, тестирование, масштабирование

Продолжение таблицы 2

Вид риска	Описание риска	Вероятность	Влияние	Меры минимизации
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

Проведенный анализ позволил сделать вывод, что наиболее значимыми являются рыночные и технологические риски, так как они напрямую влияют на успешность внедрения продукта и его стабильную работу.

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 разработкой клиентских веб-приложений для автоматизации управления отпусками сотрудников организации. В рамках исследования основное внимание уделяется проектированию и реализации пользовательского интерфейса, обеспечивающего эффективное взаимодействие пользователей с системой контроля отпусков.

Клиентская часть современного веб-приложения представляет собой программный комплекс, функционирующий в среде веб-браузера и отвечающий за отображение данных, обработку пользовательского ввода, валидацию информации и взаимодействие с серверной частью через программный интерфейс (API). От качества реализации клиентского приложения зависят удобство использования системы, скорость выполнения операций, отзывчивость интерфейса и общее восприятие программного продукта пользователями.

Клиентское приложение отображает данные о сотрудниках, организационной структуре, отпускных политиках и заявках. Через интерфейс можно создавать, редактировать и согласовывать заявки, управлять кадровой структурой и политиками, просматривать календари отпусков и аналитику, а также получать уведомления об изменениях статусов.

Система разрабатывается для организаций, работающих в соответствии с трудовым законодательством Российской Федерации. При проектировании учитываются разные режимы работы предприятий – стандартный рабочий график, сменный и круглосуточный. Клиентская часть обеспечивает отображение данных об отпусках и взаимодействие с серверной частью системы.

При разработке клиентской части необходимо обеспечить удобство работы пользователей. Поскольку системой будут пользоваться сотрудники с разным уровнем владения компьютером, интерфейс должен быть понятным и

простым в использовании. Пользователь должен иметь возможность быстро получить доступ к необходимым функциям без выполнения лишних действий.

Кроме того, приложение должно стабильно работать при одновременном использовании несколькими пользователями и корректно отображать актуальные данные, получаемые от серверной части системы. Это позволяет своевременно получать информацию о заявках, отпусках и изменениях в расписании сотрудников.

2.2 Анализ программных решений аналогичного назначения

В рамках разработки клиентской части информационной системы контроля отпусков был проведён анализ существующих программных решений аналогичного назначения с акцентом на особенности реализации пользовательских интерфейсов.

2.2.1 Популярные программные решения управления отпусками

VambooHR – американская система управления персоналом, предоставляющая функционал для ведения кадрового учёта и управления отпусками. Клиентская часть системы реализована в виде веб-приложения с современным пользовательским интерфейсом, представлен на рисунке 10 [8]. Интерфейс личного кабинета сотрудника позволяет создавать заявки на отпуск через наглядную форму с выбором дат из календаря, отслеживать остаток отпускных дней и просматривать историю заявок. Руководители получают доступ к интерфейсу согласования с удобной таблицей заявок и функциями фильтрации. К недостаткам интерфейса относится отсутствие полной локализации на русский язык и ограниченная адаптация под требования российского законодательства.

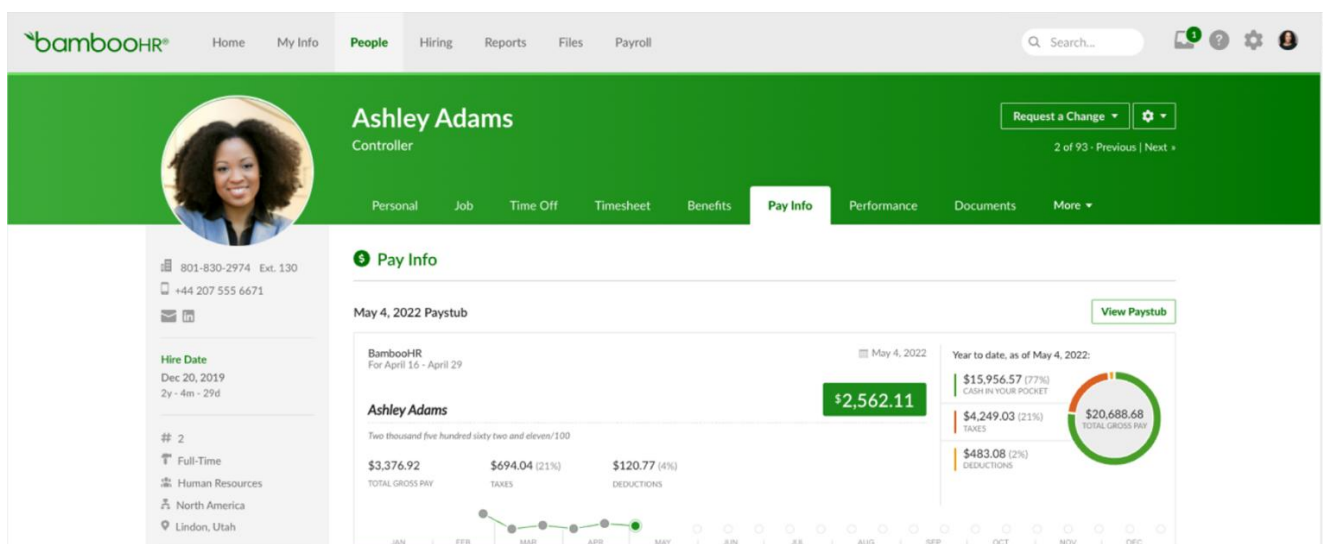


Рисунок 10 – Интерфейс программного обеспечения «BambooHR»

ТопФактор – российская информационная система управления персоналом, предназначенная для автоматизации кадровых процессов. Интерфейс системы обеспечивает централизованное управление информацией о сотрудниках и предоставляет доступ к графикам отпусков. Ключевой особенностью является интеграция кадровых процессов в единую цифровую среду. Однако интерфейс системы, представлен на рисунке 11 [9], имеет средний уровень удобства использования, а настройка отпускных политик требует дополнительных усилий со стороны пользователя.

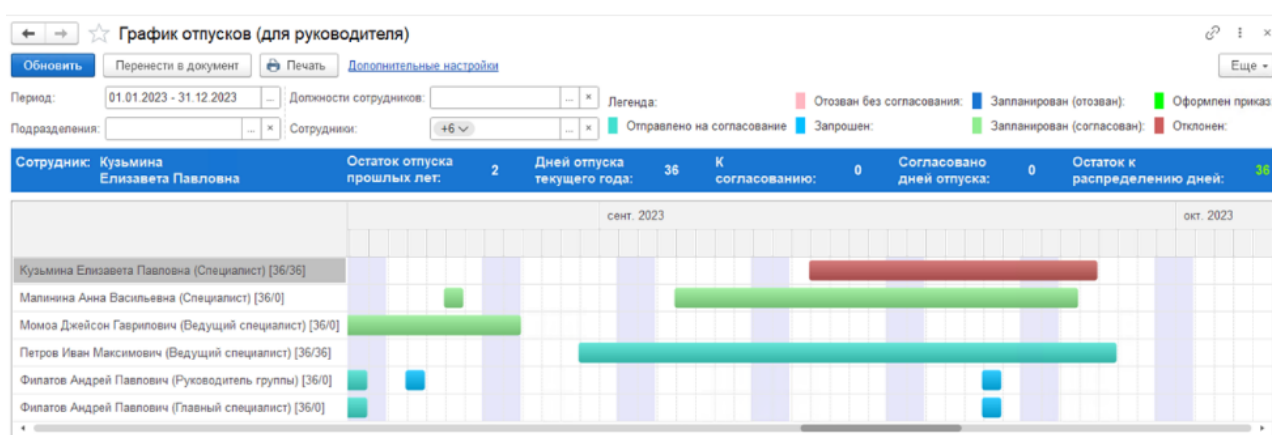


Рисунок 11 – Интерфейс программного обеспечения «ТопФактор»

1С:Зарплата и управление персоналом – комплексная система кадрового учёта и расчёта заработной платы. Интерфейс системы ориентирован на

опытных пользователей и обладает высокой функциональной плотностью, что создаёт сложности для новых пользователей. Управление отпусками реализовано как часть общего кадрового учёта, что требует от пользователя понимания общей структуры системы. К недостаткам клиентской части относятся устаревший визуальный дизайн, сложная навигация и недостаточная гибкость настройки под специфические требования организации. Интерфейс системы представлен на рисунке 12 [10].

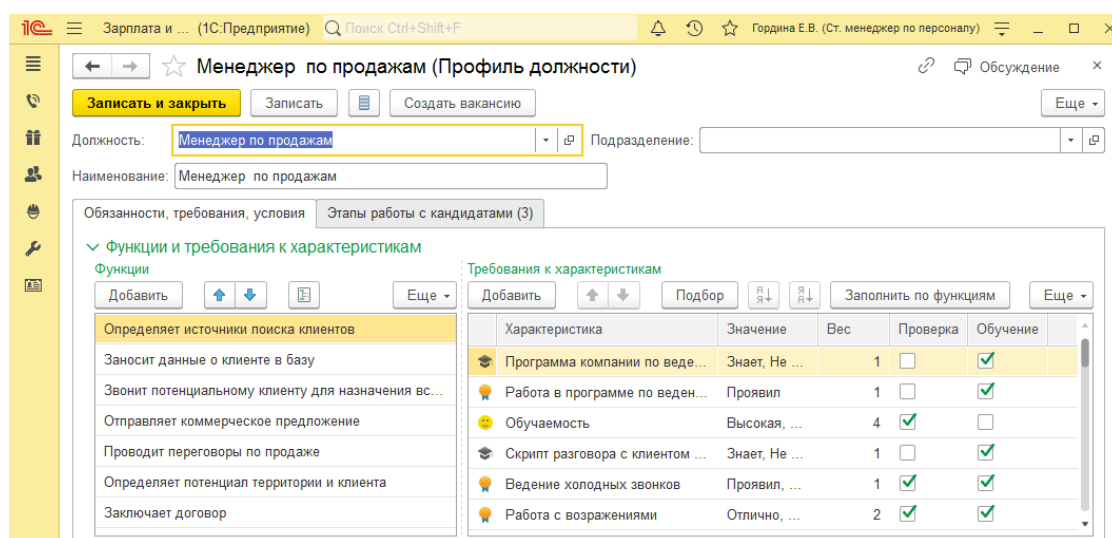


Рисунок 12 – Интерфейс программного обеспечения «1С:Зарплата и управление персоналом»

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 дизайна интерфейса 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 интерфейса, затратам на внедрение, ориентированность на российский рынок,

скорость работы интерфейса, адаптивность под мобильные устройства, поддержка сложной логики отпусков в интерфейсе.

Результаты сравнительного анализа представлены в таблице 3.

Таблица 3 – Сравнительный анализ программных решений для управления отпусками

Критерий	VambooHR	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 интерфейса	Высокое	Среднее	Низкое	Высокое
Скорость работы интерфейса	Высокая	Средняя	Низкая	Высокая
Адаптивность под мобильные устройства	Есть	Ограниченная	Нет	Планируется
Поддержка сложной логики отпусков в интерфейсе	Ограниченная	Ограниченная	Частично	Расширенная
Стоимость внедрения	Средняя	Средняя	Высокая	Низкая
Избыточность функционала	Высокая	Высокая	Очень высокая	Низкая
Поддержка российского законодательства	Ограниченная	Полная	Полная	Полная
Круглосуточный режим работы	Нет	Нет	Частично	Полная
Контроль замещения сотрудников	Нет	Нет	Есть	Есть
Масштабируемость	Высокая	Высокая	Высокая	Высокая

Сравнение показывает, что существующие решения в большинстве случаев ориентированы на комплексную автоматизацию HR-процессов, что отражается на сложности их пользовательских интерфейсов. В свою очередь, разрабатываемая система фокусируется на узкой задаче управления отпусками, что позволяет создать более простой и удобный интерфейс, адаптированный под потребности конечных пользователей.

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 клиентского приложения является гибкая система управления отпускными политиками с поддержкой различных режимов работы организации и наглядным отображением всех этапов согласования заявок.

Таким образом, разрабатываемая система ориентирована на решение конкретной задачи управления отпусками, сочетая удобство использования, гибкость интерфейса 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 клиентской части информационной системы контроля отпусков определены следующие функциональные требования:

- реализация формы авторизации с валидацией учётных данных и отображением сообщений об ошибках;
- предоставление пользователю информации о профиле и остатке отпускных дней в личном кабинете;
- реализация интерфейса создания, редактирования и отправки заявок на отпуск с выбором дат, типа отпуска и указанием комментария;
- отображение статусов заявок (черновик, на согласовании, согласовано, отклонено, отменено) с визуальной индикацией;
- реализация интерфейса согласования заявок для руководителей с возможностью одобрения, отклонения и добавления комментария;
- отображение календаря отпусков команды в наглядном формате (таблица, timeline);
- реализация интерфейсов управления кадровой структурой для HR-специалиста и администратора;
- реализация форм настройки отпускных политик с валидацией вводимых значений;
- отображение уведомлений о событиях системы в реальном времени;
- реализация интерфейса формирования и экспорта отчётов в формате Excel;
- обеспечение системы навигации между разделами приложения;

- реализация загрузки и отображения списков сотрудников с возможностью поиска и фильтрации;
- обеспечение импорта данных сотрудников из Excel-файлов через интерфейс загрузки.

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 выполнения определенных задач [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 системы, а определяется положением сотрудника внутри организации.

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

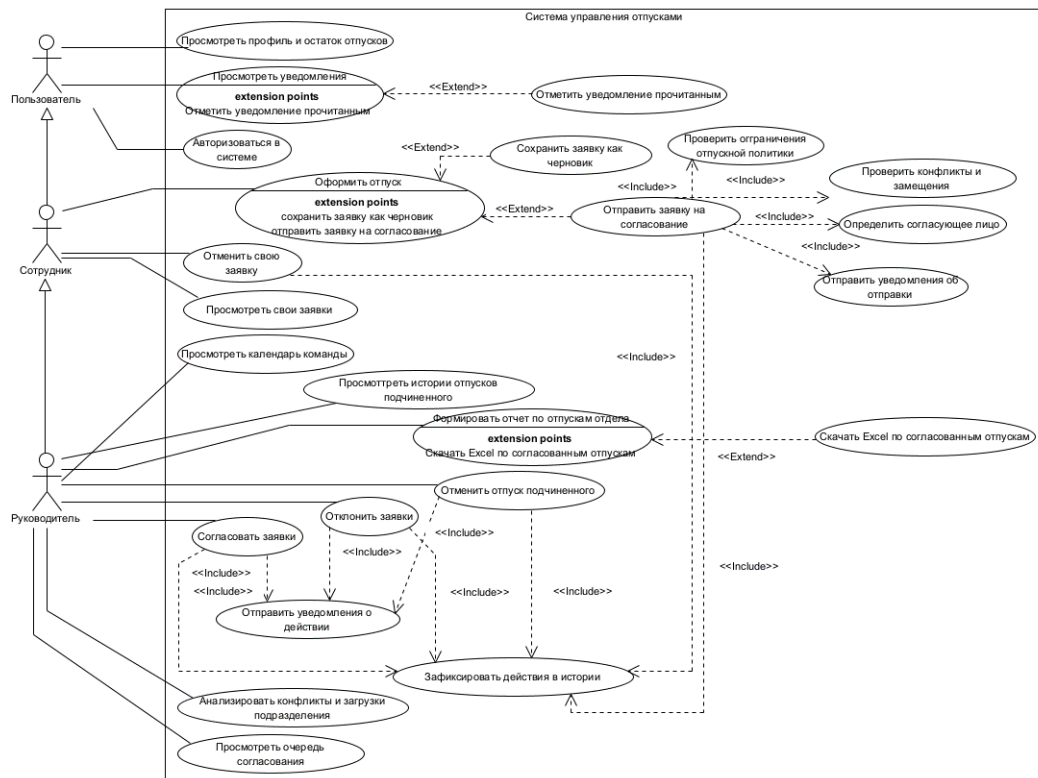


Рисунок 13 – Диаграмма вариантов использования, отражающая возможности ролей пользователя, сотрудника и руководителя

Таким образом, диаграмма вариантов использования полностью отражает основные возможности пользователя, сотрудника и руководителя.

Также была разработана отдельная UML диаграмма вариантов использования для роли HR-специалиста и администратора. Следующая схема позволяет оценить полный спектр возможностей предоставляемый разработанным решением.

Диаграмма вариантов использования для роли HR-специалиста и администратора представлена на рисунке 14.

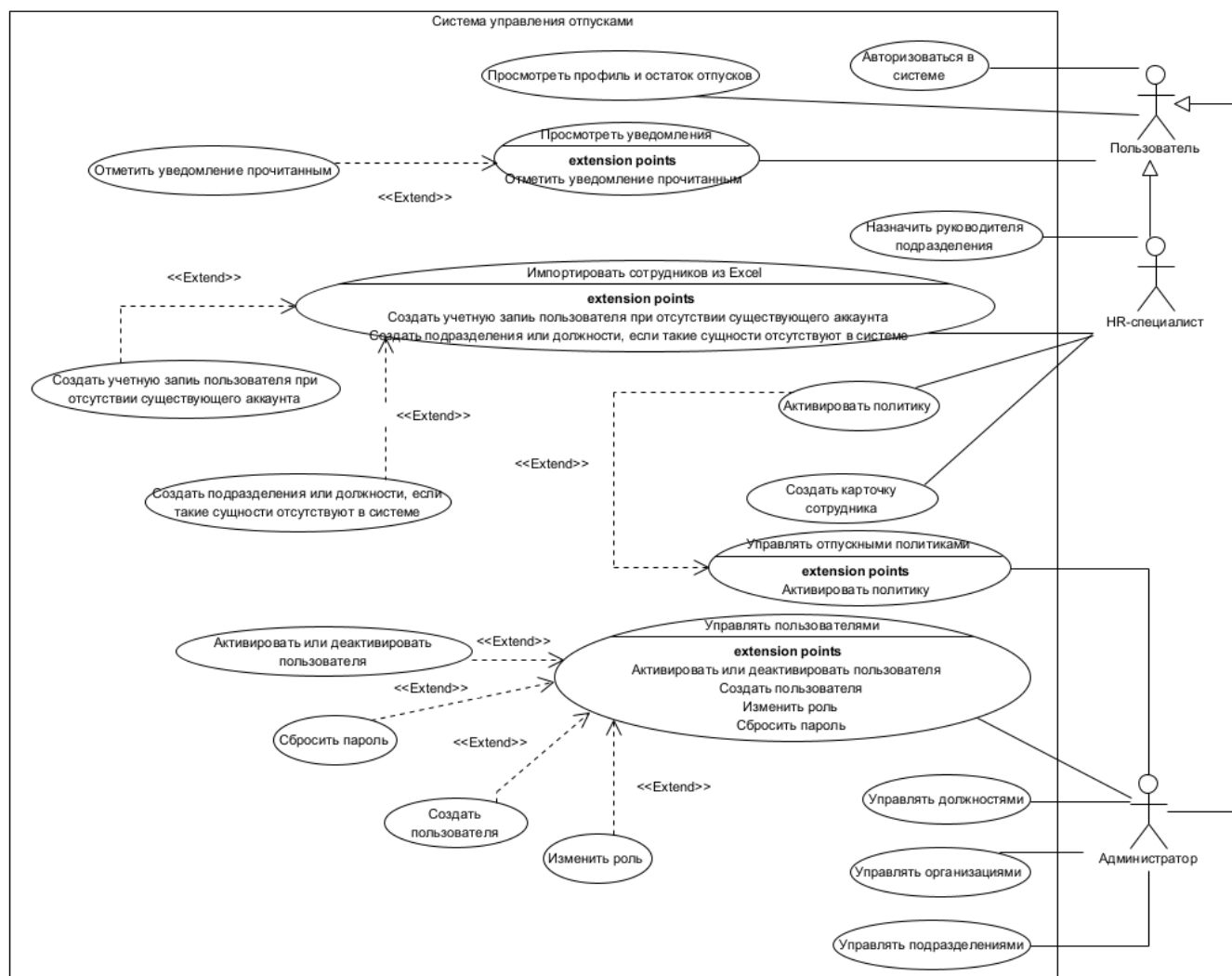


Рисунок 14 – Диаграмма вариантов использования, отражающая возможности ролей HR-специалиста и администратора

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, системы отражают реализуемый в клиентской части функционал и взаимодействие пользователей с интерфейсом. Результаты полученные, в ходе анализа функциональных требований послужат основой для дальнейшего проектирования системы и реализации клиентской части продукта в рамках бизнес-проекта.

2.4.2 Анализ нефункциональных требований

На данном этапе были определены нефункциональные требования к разрабатываемой клиентской части системы. Клиентское приложение реализуется с использованием языка программирования TypeScript и библиотеки React. Разработка ведётся в среде Visual Studio Code. Данный стек технологий обеспечивает создание современных одностраничных веб-приложений с высокой скоростью работы интерфейса и поддержкой компонентного подхода к разработке.

Взаимодействие клиентской части с сервером осуществляется через REST API с использованием протокола HTTP и формата данных JSON. Для выполнения HTTP-запросов применяется библиотека Axios, обеспечивающая удобную обработку запросов, ответов сервера и ошибок взаимодействия. Управление состоянием приложения реализовано с использованием Redux Toolkit, что позволяет централизованно хранить данные и обеспечивать предсказуемость потока данных внутри приложения.

Клиентская часть реализована как SPA (Single Page Application) – страница не перезагружается полностью при переходах, содержимое обновляется динамически. Архитектура построена на принципе разделения ответственности: каждый компонент отвечает за свою часть интерфейса и не зависит от остальных.

К производительности интерфейса предъявляется требование минимального времени отклика. Переходы между разделами и обновление данных должны выполняться без заметных задержек для пользователя.

Интерфейс должен иметь единый стиль оформления и последовательную структуру расположения элементов. Это снижает время на освоение системы и уменьшает вероятность ошибок при работе с ней.

Приложение должно корректно работать на устройствах с разными размерами экрана – как на настольных компьютерах, так и на мобильных устройствах. Для этого предусмотрен адаптивный интерфейс.

Система должна поддерживать работу в современных веб-браузерах, включая Google Chrome, Mozilla Firefox, Microsoft Edge, Safari и Яндекс Браузер. Также необходимо обеспечить корректную обработку сетевых и серверных ошибок с выводом понятных пользователю сообщений.

Для защиты данных пользователей применяется механизм аутентификации и авторизации на основе JWT-токенов. Клиентская часть отвечает за хранение данных текущей сессии и обновление токенов при взаимодействии с серверной частью системы.

Архитектура приложения должна обеспечивать возможность дальнейшего развития системы. Добавление новых страниц, компонентов и функциональных модулей не должно требовать существенной переработки уже реализованного кода.

2.5 Планирование разработки и оценка бюджета

После определения требований к клиентской части был сформирован план разработки приложения. В него вошли работы по проектированию пользовательского интерфейса, реализации компонентной структуры, настройке маршрутизации, организации управления состоянием, интеграции с серверной частью и проведению тестирования.

Планирование разработки позволяет определить последовательность выполнения работ, оценить трудоёмкость отдельных этапов и рассчитать предполагаемые затраты на реализацию проекта. Полученные оценки используются для формирования календарного плана разработки и расчёта бюджета проекта.

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

	Задача	Исполнитель	Статус	+
1	Анализ предметной области и требований	 Фронтенд Дмитрий	● Открыт	...
2	Проектирование архитектуры клиентской части	 Фронтенд Дмитрий	● Открыт	...
3	Проектирование структуры интерфейса и навиг...	 Фронтенд Дмитрий	● Открыт	...
4	Разработка дизайн-макетов пользовательского ...	 Фронтенд Дмитрий	● Открыт	...
5	Настройка структуры проекта и окружения разр...	 Фронтенд Дмитрий	● Открыт	...
6	Реализация маршрутизации приложения	 Фронтенд Дмитрий	● Открыт	...
7	Реализация системы авторизации и аутентифик...	 Фронтенд Дмитрий	● Открыт	...
8	Реализация управления состоянием приложения	 Фронтенд Дмитрий	● Открыт	...
9	Разработка интерфейса личного кабинета сотру...	 Фронтенд Дмитрий	● Открыт	...
10	Разработка интерфейса руководителя	 Фронтенд Дмитрий	● Открыт	...
11	Разработка интерфейса HR-специалиста	 Фронтенд Дмитрий	● Открыт	...
12	Разработка интерфейса администратора	 Фронтенд Дмитрий	● Открыт	...
13	Реализация форм и клиентской валидации данн...	 Фронтенд Дмитрий	● Открыт	...
14	Интеграция клиентской части с REST API	 Фронтенд Дмитрий	● Открыт	...
15	Реализация календарей и отображения графиков	 Фронтенд Дмитрий	● Открыт	...
16	Реализация системы уведомлений	 Фронтенд Дмитрий	● Открыт	...
17	Обработка ошибок и состояний загрузки	 Фронтенд Дмитрий	● Открыт	...
18	Реализация адаптивного интерфейса	 Фронтенд Дмитрий	● Открыт	...
19	Тестирование клиентской части	 Фронтенд Дмитрий	● Открыт	...
20	Оптимизация производительности интерфейса	 Фронтенд Дмитрий	● Открыт	...
21	Подготовка документации и диаграмм	 Фронтенд Дмитрий	● Открыт	...

Рисунок 15 – Список задач на разработку клиентской 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 клиентской части информационной системы контроля отпусков с указанием временных интервалов выполнения задач.

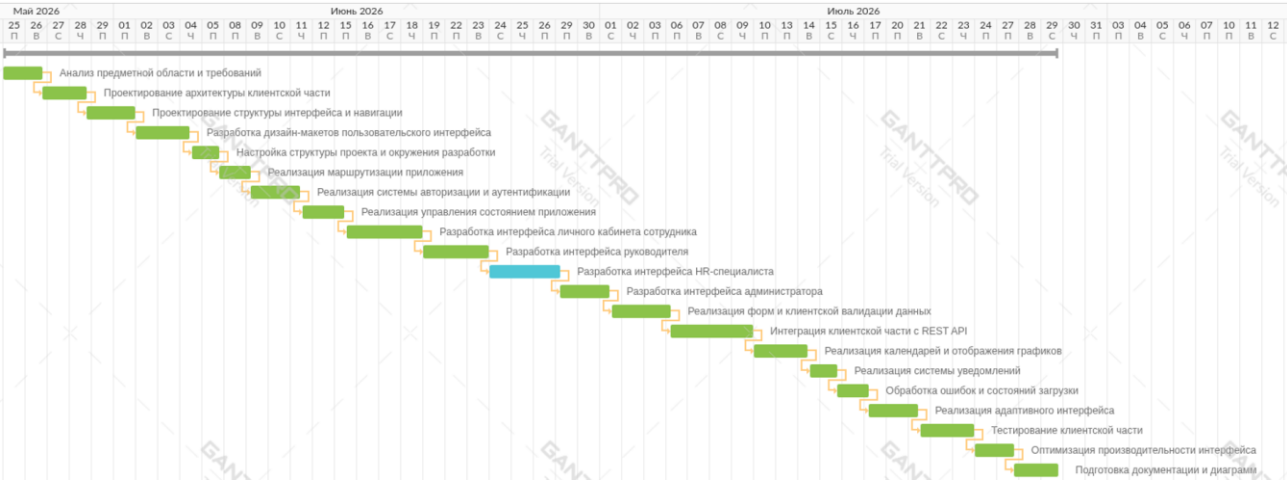


Рисунок 16 – Диаграмма Ганта системы контроля отпусков

На начальных этапах работы выполняются анализ и проектирование, после чего осуществляется разработка основных модулей системы. Наиболее трудоемкими этапами являются разработка интерфейса личного кабинета, интеграция клиентской части с REST API.

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

Загрузка ресурсов													5/22/2026
Ресурс	5/17/2026	5/24/2026	5/31/2026	6/7/2026	6/14/2026	6/21/2026	6/28/2026	7/5/2026	7/12/2026	7/19/2026	7/26/2026	8/2/2026	
Фронтенд Дмитрий	0	40	40	40	40	40	40	40	40	40	22	0	

Рисунок 17 – Количество затраченных часов на разработку

Приблизительный расчет разработки клиентской 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.)
Frontend-разработчик	382	800	1100	305,600
Итого	382	—	—	305,600

Таким образом, разработка клиентской 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 клиентской части приложения системы контроля отпусков

3.1 Проектирование базы данных

3.1.1 Концептуальное проектирование базы данных

Одним из ключевых этапов проектирования системы контроля отпусков является разработка концептуальной схемы базы данных [14]. 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 18 представлена диаграмма концептуальной модели базы данных системы контроля отпусков.

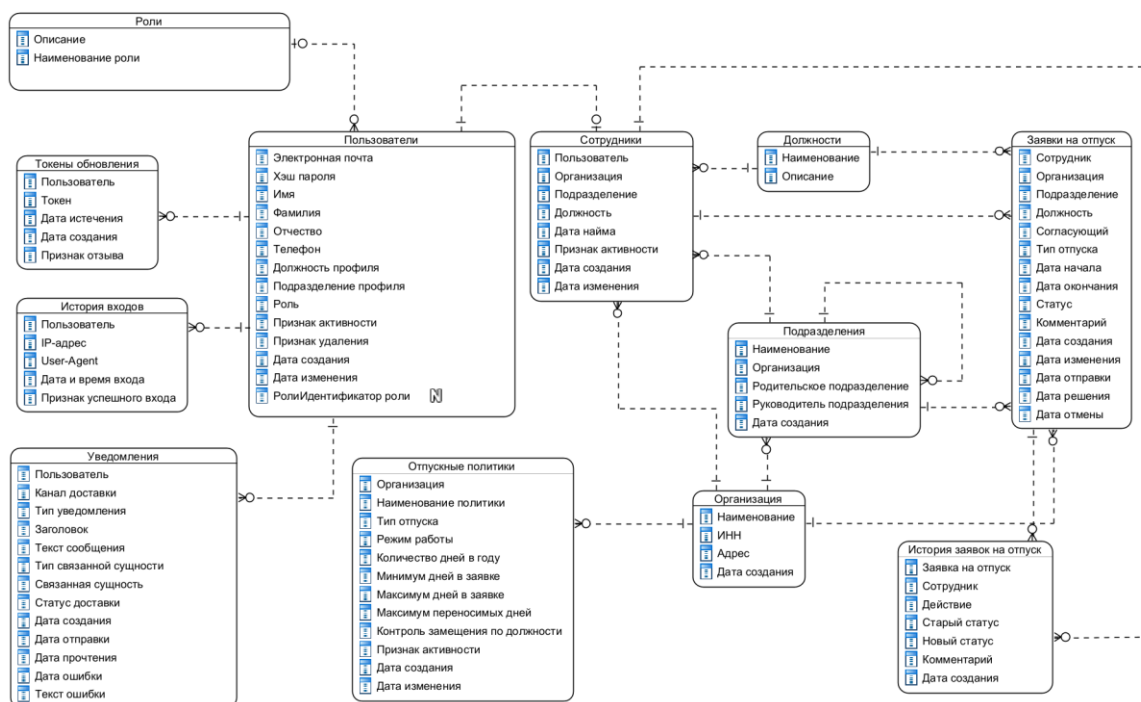


Рисунок 18 – Диаграмма концептуальной модели (схемы) базы данных системы контроля отпусков

В таблице 5 приведено краткое содержание сущностей концептуальной схемы базы данных, позволяющее получить полноценное представление о структуре и назначении основных объектов системы. Каждая сущность отражает отдельную предметную область, связанную с управлением отпусками сотрудников, хранением информации о пользователях, организационной структуре и заявках на отпуск. Представленная концептуальная схема служит основой для дальнейшего логического и физического проектирования базы данных, а также обеспечивает целостность и взаимосвязанность данных в рамках разрабатываемой системы.

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

Продолжение таблицы 6

№ п/п	Сущность 1	Сущность 2	Тип связи	Описание
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
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 логической схемы базы данных.

3.1.2 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 [15].

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 рисунке 19 представлена диаграмма логической модели базы данных системы контроля отпусков [15].

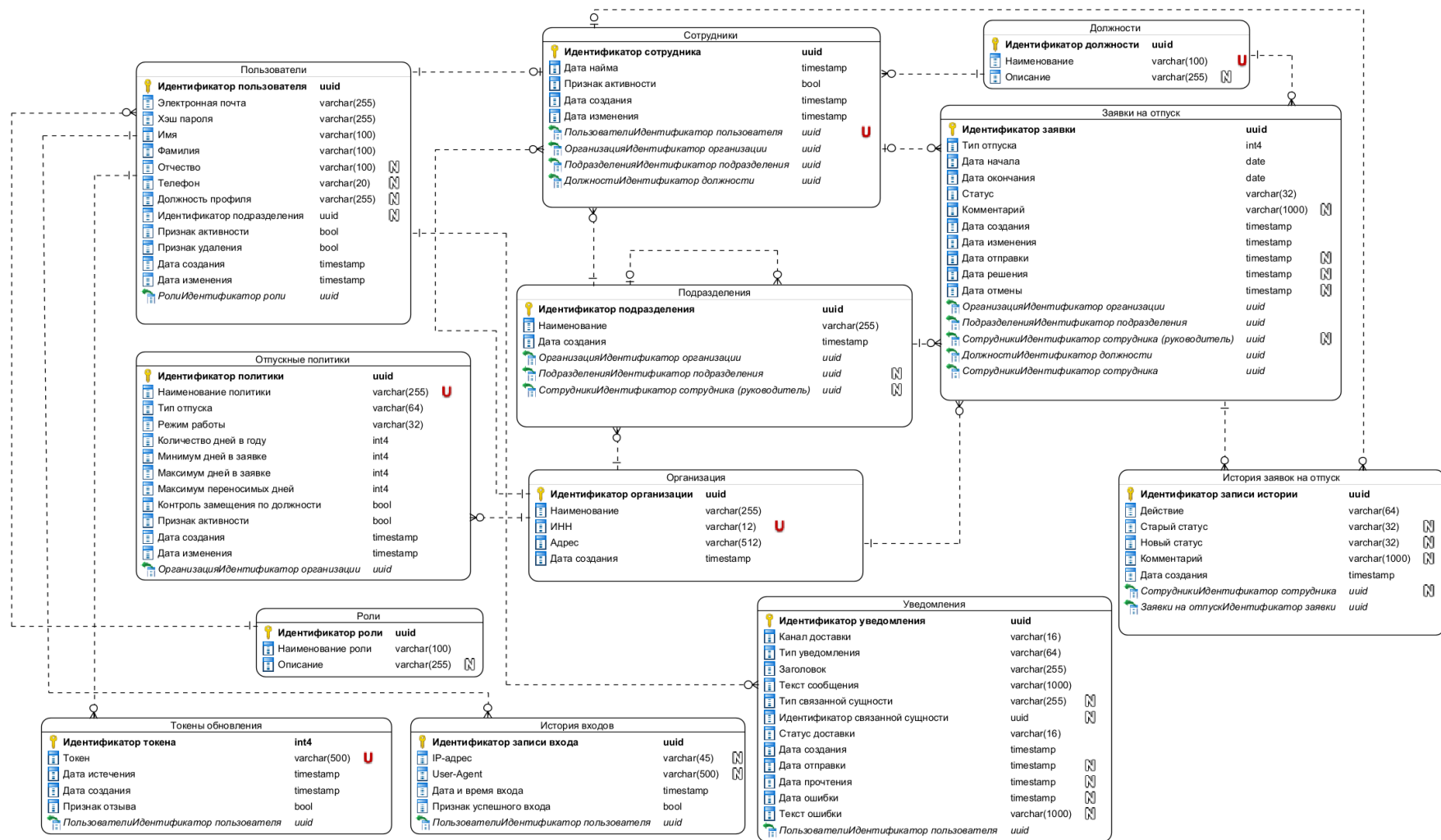


Рисунок 19 – Диаграмма логической модели базы данных системы контроля отпусков

Подробное описание атрибутов и ограничений сущностей представлено в приложении Б.

Таким образом, логическая модель базы данных детализирует структуру хранения данных системы контроля отпусков, включая используемые типы данных и ограничения целостности. Это обеспечивает корректное и непротиворечивое хранение информации, а также формирует основу для дальнейшей реализации физической модели базы данных.

3.2 Выбор технологий разработки

В процессе проектирования клиентской части приложения был проведён анализ современных подходов и технологий разработки веб-приложений, на основе которого был выбран оптимальный технологический стек, соответствующий требованиям проекта [16]. В качестве основного языка программирования используется TypeScript – строго типизированное надмножество JavaScript, разработанное компанией Microsoft. Его применение позволяет обнаруживать ошибки ещё на этапе компиляции, повышает стабильность кода и упрощает его последующее сопровождение и рефакторинг [17].

Использование TypeScript особенно актуально для приложений со сложной структурой состояния и множеством взаимосвязанных компонентов, поскольку типизация интерфейсов данных, получаемых от сервера, гарантирует корректность их обработки на стороне клиента.

Библиотека для построения пользовательского интерфейса. Для разработки клиентской части выбрана библиотека React, разработанная компанией Meta. React реализует компонентный подход к построению интерфейсов, при котором каждый элемент представляет собой независимый, переиспользуемый компонент [18]. Ключевыми преимуществами React являются:

- виртуальный DOM (Document Object Model), обеспечивающий эффективное обновление интерфейса при изменении данных;
- декларативный подход к описанию UI, упрощающий понимание и отладку кода;
- обширная экосистема библиотек и инструментов;
- высокая производительность при работе с динамическими данными;
- поддержка TypeScript из коробки.

Сборщик проекта. В качестве инструмента сборки выбран Vite – современный сборщик, обеспечивающий быстрый старт разработки, мгновенную горячую перезагрузку модулей (HMR) и оптимизированную сборку production-версии. Vite использует нативные ES-модули в режиме разработки и Rollup для production-сборки, что обеспечивает высокую скорость работы [19].

Стилизация компонентов. Для стилизации пользовательского интерфейса выбрана утилитарная CSS-библиотека Tailwind CSS. Данный подход позволяет создавать адаптивные интерфейсы непосредственно в разметке компонентов, обеспечивает единообразие дизайна за счёт predefined системы значений (цветов, отступов, типографики) и снижает объём пользовательских CSS-файлов [20].

Управление состоянием. Для управления глобальным состоянием приложения выбрана библиотека Zustand. Zustand предоставляет минималистичный API для создания хранилищ состояния с поддержкой TypeScript, не требует обёрток провайдеров и обеспечивает высокую производительность благодаря селекторной подписке на изменения [21].

Маршрутизация. Для организации клиентской маршрутизации выбрана библиотека React Router – стандартное решение для React-приложений, обеспечивающее декларативное определение маршрутов, вложенную маршрутизацию, защиту маршрутов и поддержку параметров URL [22].

HTTP-клиент. Для взаимодействия с серверным API выбрана библиотека Axios. Axios предоставляет удобный API для выполнения HTTP-запросов, поддерживает перехватчики запросов и ответов, автоматическую трансформацию данных JSON и удобную обработку ошибок [23].

Валидация форм. Для валидации данных в формах выбрана библиотека Zod, обеспечивающая декларативное описание схем данных с полной поддержкой TypeScript. Zod позволяет определять правила валидации на стороне клиента, соответствующие ограничениям серверной части [24].

Таблица 7 содержит сводку выбранных технологий разработки клиентской части.

Таблица 7 – Технологический стек клиентской части

Категория	Технология	Назначение
Язык программирования	TypeScript 5.x	Типизированная разработка
UI-библиотека	React 18.x	Построение пользовательского интерфейса
Сборщик	Vite 5.x	Сборка и запуск проекта
Стилизация	Tailwind CSS 3.x	Утилитарные CSS-стили
Состояние	Zustand 4.x	Управление глобальным состоянием
Маршрутизация	React Router 6.x	Клиентская навигация
HTTP-клиент	Axios 1.x	Взаимодействие с REST API
Валидация	Zod 3.x	Валидация форм и данных

Продолжение таблицы 7

Категория	Технология	Назначение
Даты	date-fns 3.x	Работа с датами
Компоненты	shadcn/ui	Базовые UI-компоненты

Работа с датами. Для обработки и отображения дат выбрана библиотека date-fns, предоставляющая модульный набор функций для парсинга, форматирования и манипуляции датами.

Таким образом, в ходе проектирования клиентской части приложения был сформирован современный и технологически обоснованный стек. Выбранные технологии обеспечивают высокую производительность, масштабируемость и надёжность разработки, а также удобство сопровождения и расширения приложения.

3.3 Проектирование общей архитектуры программного обеспечения

В качестве основы построения клиентской части приложения была выбрана слоистая архитектура с использованием компонентного подхода. Архитектура клиентской части предполагает разделение приложения на независимые уровни, каждый из которых отвечает за выполнение определённого набора задач.

Преимущества выбранного подхода являются:

- разделение ответственности между слоями приложения;
- возможность повторного использования компонентов интерфейса;
- централизованное управление состоянием приложения;
- упрощение поддержки и масштабирования системы;
- независимость пользовательского интерфейса от реализации

серверного API.

Архитектура клиентской части представлена на рисунке 20.

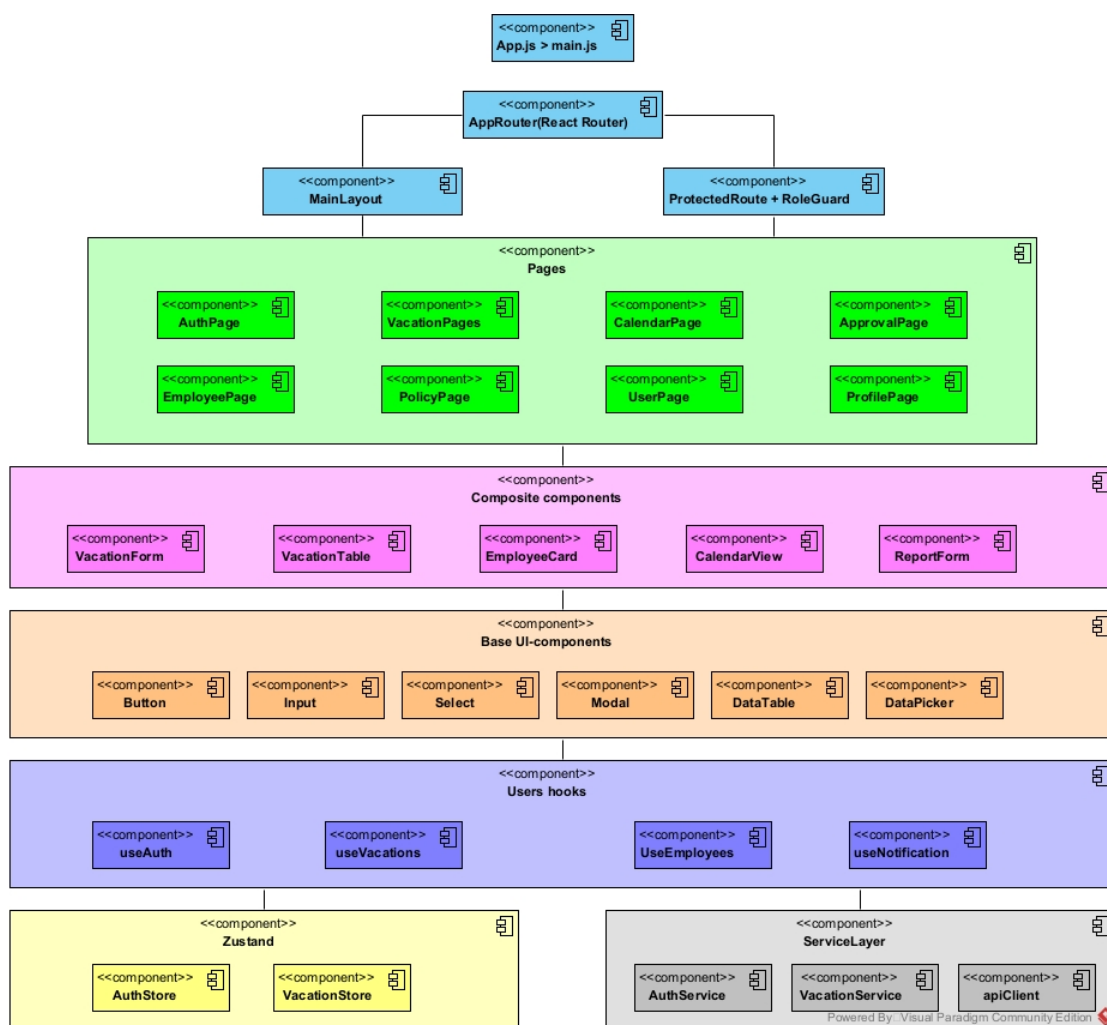


Рисунок 20 – Архитектура клиентской части

Клиентское приложение включает несколько основных уровней. Уровень представления отвечает за отображение пользовательского интерфейса и обработку действий пользователя. На данном уровне располагаются React-компоненты, реализующие страницы приложения, формы, таблицы, элементы навигации и другие интерфейсные элементы. Компоненты данного уровня не содержат бизнес-логики и взаимодействуют с другими уровнями системы через пользовательские hook и механизмы управления состоянием.

Для организации навигации между страницами приложения используется библиотека React Router. Маршрутизация обеспечивает переход между разделами системы без полной перезагрузки страницы. Для ограничения доступа к защищённым разделам реализованы компоненты ProtectedRoute и RoleGuard, обеспечивающие проверку аутентификации пользователя и его роли в системе.

Дополнительно в структуре интерфейса используются layout-компоненты, которые отвечают за формирование общей разметки приложения, включая боковое меню, верхнюю панель и основную область отображения контента страниц.

Уровень состояния реализован с использованием библиотеки Zustand. Он включает в себя набор хранилищ, методы изменения состояния, а также получение производных данных. Состояние разделено по основным функциональным модулям системы, таким как авторизация, управление сотрудниками, заявки на отпуск, уведомления и другие части приложения.

Для разделения бизнес-логики и компонентов интерфейса в проекте применяются пользовательские React-hooks. Они позволяют инкапсулировать работу с хранилищами состояния и сервисами, за счёт чего достигается повторное использование логики и уменьшается связность компонентов.

Для взаимодействия с серверной частью выделен отдельный слой доступа к данным. Он реализован через сервисы на базе библиотеки Axios, которые отвечают за выполнение HTTP-запросов и обработку ответов от сервера. Каждый сервис связан с конкретным модулем backend-системы и скрывает детали работы с REST API, обеспечивая более удобное и структурированное взаимодействие с данными.

В архитектуре приложения также выделен инфраструктурный уровень. В него входят конфигурация HTTP-клиента, обработчики ошибок, утилиты, константы и общие TypeScript-типы. Это позволяет не дублировать одни и те же механизмы в разных модулях приложения.

Взаимодействие между уровнями работает следующим образом. Пользователь выполняет действие в интерфейсе – компонент обращается к хранилищу или сервису, сервис формирует HTTP-запрос к серверу, ответ сохраняется в состоянии и отображается в интерфейсе.

Слоистая архитектура в сочетании с компонентным подходом делает код более предсказуемым: каждый уровень отвечает за свою задачу и не зависит от

деталей реализации других. Это упрощает внесение изменений и добавление новой функциональности.

3.4 Проектирование компонентной структуры

Компонентная структура клиентского приложения построена с использованием компонентного подхода, обеспечивающего разделение интерфейса на независимые переиспользуемые элементы. Такой подход упрощает сопровождение проекта, позволяет повторно использовать элементы интерфейса и облегчает дальнейшее расширение функциональности приложения.

На рисунке 21 представлена диаграмма компонентов клиентской части приложения.

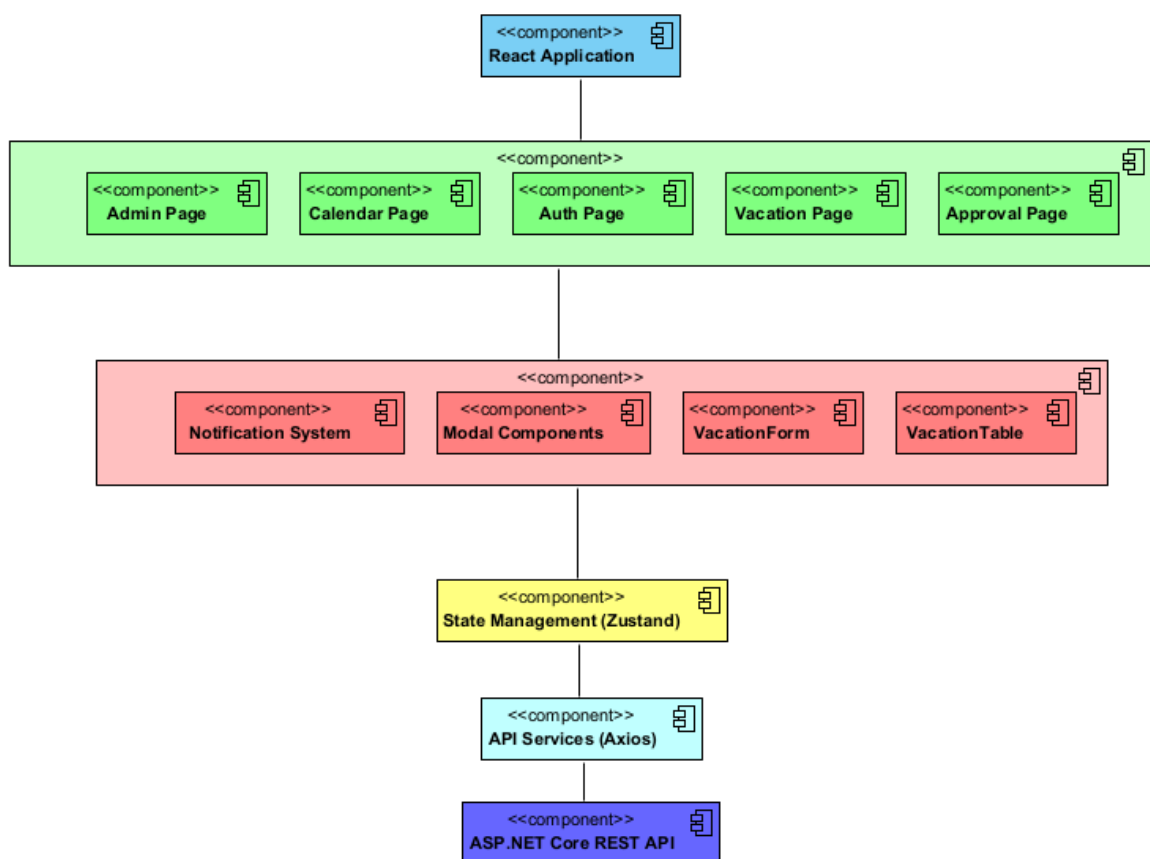


Рисунок 21 – Диаграмма компонентов

В структуре клиентской части можно выделить несколько уровней компонентов.

Базовые UI-компоненты представляют собой универсальные элементы интерфейса, не зависящие от предметной области системы. К ним относятся кнопки, поля ввода, таблицы, модальные окна, выпадающие списки, элементы выбора даты, уведомления и другие элементы интерфейса. Реализация базовых компонентов выполнена с использованием библиотеки `shadcn/ui` с дополнительной адаптацией под стилистику разрабатываемой информационной системы.

На следующем уровне располагаются композитные компоненты, объединяющие несколько базовых элементов в законченные функциональные блоки интерфейса. К таким компонентам относятся формы создания заявок, карточки сотрудников, таблицы заявок, элементы календаря и панели фильтрации данных.

Таблица 8 содержит описание основных страниц клиентского приложения и их назначение в системе.

Таблица 8 – Технологический стек клиентской части

Страница	Назначение	Роли
Авторизация	Форма входа в систему	Все
Профиль	Просмотр и редактирование профиля	Все
Мои заявки	Список заявок сотрудника, создание новой	Сотрудник
Создание заявки	Форма создания/редактирования заявки	Сотрудник
Календарь команды	Визуальный календарь отпусков подразделения	Руководитель
Очередь согласования	Список заявок на согласование	Руководитель
Отчёты	Формирование и выгрузка отчётов	Руководитель, HR
Сотрудники	Управление списком сотрудников	HR, Администратор

Продолжение таблицы 8

Страница	Назначение	Роли
Политики отпусков	Настройка правил предоставления отпусков	HR
Пользователи	Управление учётными записями	Администратор
Структура организации	Управление подразделениями и должностями	Администратор
Уведомления	Список системных уведомлений	Все

Страничные компоненты соответствуют отдельным разделам веб-приложения и формируют пользовательские сценарии работы с системой. Каждая страница объединяет набор композитных и базовых компонентов, необходимых для реализации определённого функционала системы.

Дополнительно в приложении используются layout-компоненты, отвечающие за общую структуру интерфейса. Они обеспечивают единое расположение навигационной панели, верхнего меню, рабочей области и вспомогательных элементов интерфейса для всех страниц приложения.

Таким образом, компонентная структура клиентской части приложения обеспечивает разделение интерфейса на независимые функциональные элементы, что упрощает разработку, сопровождение и дальнейшее развитие системы. Использование многоуровневой организации компонентов позволяет переиспользовать элементы интерфейса в различных разделах приложения и поддерживать единый стиль оформления.

3.5 Проектирование маршрутизации и навигации

Система маршрутизации клиентского приложения разработана с учётом разграничения доступа пользователей в зависимости от их роли и обеспечения удобной навигации между разделами системы.

Для реализации маршрутизации используется библиотека React Router версии 6.x. Структура маршрутов построена по иерархическому принципу с использованием вложенных маршрутов, позволяющих применять общую

структуру интерфейса для группы страниц. Такой подход обеспечивает единообразие отображения элементов навигации и упрощает сопровождение клиентской части приложения.

На рисунке 22 представлена схема маршрутизации клиентского приложения.

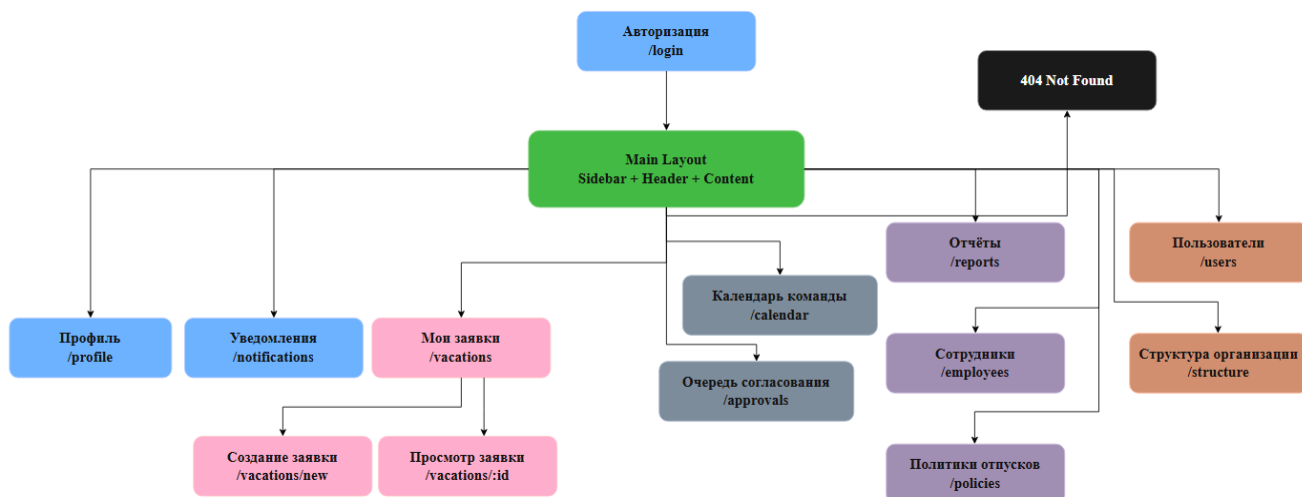


Рисунок 22 – Диаграмма навигации

При проектировании системы навигации были реализованы следующие принципы:

- разграничение доступа к маршрутам в зависимости от роли пользователя;
- использование защищённых маршрутов для страниц, требующих авторизации;
- применение вложенной маршрутизации для повторного использования layout-компонентов;
- поддержка ленивой загрузки страниц для уменьшения времени первоначальной загрузки приложения;
- обработка неизвестных маршрутов с отображением страницы ошибки 404.

После прохождения аутентификации пользователь получает доступ только к тем разделам системы, которые соответствуют его роли. Проверка прав доступа выполняется на стороне клиента перед отображением страницы. При отсутствии авторизации пользователь автоматически перенаправляется на страницу входа в систему.

Таблица 9 содержит описание основных маршрутов клиентского приложения.

Таблица 9 – Основные маршруты клиентского приложения

Маршрут	Страница	Требуется авторизация	Доступные роли
/login	Авторизация	Нет	–
/	Главная страница (редирект)	Да	Все
/profile	Профиль пользователя	Да	Все
/vacations	Мои заявки	Да	Сотрудник
/vacations/new	Создание заявки	Да	Сотрудник
/vacations/:id	Просмотр заявки	Да	Сотрудник, Руководитель
/calendar	Календарь команды	Да	Руководитель
/approvals	Очередь согласования	Да	Руководитель
/reports	Отчёты	Да	Руководитель, HR
/employees	Сотрудники	Да	HR, Администратор
/policies	Политики отпусков	Да	HR
/users	Пользователи	Да	Администратор
/structure	Структура организации	Да	Администратор
/notifications	Уведомления	Да	Все
*	Страница 404	Нет	–

Таким образом навигационная панель приложения формируется динамически. Пункты меню, недоступные текущему пользователю, не отображаются в интерфейсе, что позволяет уменьшить количество лишних элементов навигации и упростить взаимодействие с системой.

3.6 Проектирование взаимодействия с серверной частью

Взаимодействие клиентской части с серверной частью системы осуществляется через REST API с использованием протокола HTTP и формата обмена данными JSON. Такой подход обеспечивает независимость клиентского и серверного приложений, а также упрощает дальнейшее развитие системы.

Для выполнения HTTP-запросов в клиентском приложении используется библиотека Axios. На её основе настроен единый экземпляр HTTP-клиента, содержащий общую конфигурацию запросов и обработчиков ответов.

На рисунке 23 представлена диаграмма последовательности.

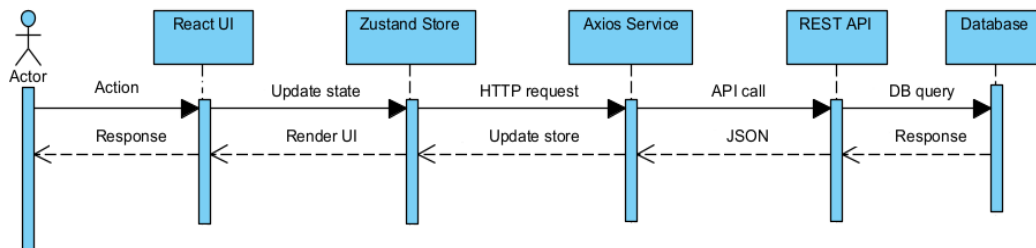


Рисунок 23 – Диаграмма последовательности

В процессе работы приложения используются перехватчики запросов и ответов. Перед отправкой запроса клиент автоматически добавляет заголовок Authorization с JWT-токеном текущего пользователя. После получения ответа выполняется централизованная обработка данных и возможных ошибок, вызванных взаимодействием.

Аутентификация в системе построена на JWT-токенах. При входе сервер выдает два токена: access-токен для запросов к API и refresh-токен для его обновления. Когда срок действия access-токена истекает, приложение автоматически запрашивает новый с помощью refresh-токена и повторяет исходный запрос. Пользователю при этом ничего делать не нужно – сессия продолжается без повторного ввода логина и пароля.

Обработка ошибок в приложении разделена по типам. При отсутствии соединения с сервером выводится уведомление о проблемах с сетью. Ошибки валидации отображаются рядом с полями формы, к которым они относятся – это позволяет сразу понять, что именно нужно исправить. Если ошибка произошла на стороне сервера, пользователь видит сообщение с предложением повторить попытку позже.

За взаимодействие с REST API отвечает отдельный сервисный слой. Каждый сервис отвечает за взаимодействие с определённым функциональным модулем серверной части: авторизацией, сотрудниками, заявками на отпуск,

отчётами и уведомлениями. Использование сервисного слоя позволяет централизовать логику работы с API и уменьшить связность компонентов приложения.

Таблица 10 содержит описание основных сервисных модулей клиентской части.

Таблица 10 – Сервисные модули клиентской части

Сервис	Методы	Назначение
AuthService	login, logout, refresh	Аутентификация и управление сессией
UserService	getCurrentUser, updateProfile	Управление профилем пользователя
EmployeeService	getList, getById, create, update, import	Управление сотрудниками
VacationService	getList, create, update, submit, cancel	Управление заявками на отпуск
ApprovalService	getQueue, approve, reject	Согласование заявок
PolicyService	getList, create, update, delete	Управление отпускными политиками
CalendarService	getTeamCalendar	Получение данных календаря
ReportService	generate, exportExcel	Формирование и экспорт отчётов
NotificationService	getList, markAsRead	Управление уведомлениями

Все ответы серверной части преобразуются в типизированные объекты TypeScript, что позволяет повысить надёжность обработки данных и снизить вероятность ошибок при разработке клиентского приложения.

3.7 Проектирование управления состоянием приложения

Управление состоянием клиентского приложения реализовано с использованием библиотеки Zustand. Архитектура управления состоянием построена по принципу разделения хранилищ в соответствии с предметными областями приложения. Каждое хранилище отвечает за отдельный функциональный модуль системы и содержит данные состояния, методы их изменения, а также функции получения производной информации.

На рисунке 24 представлена схема взаимодействия компонентов клиентского приложения с хранилищами состояния.

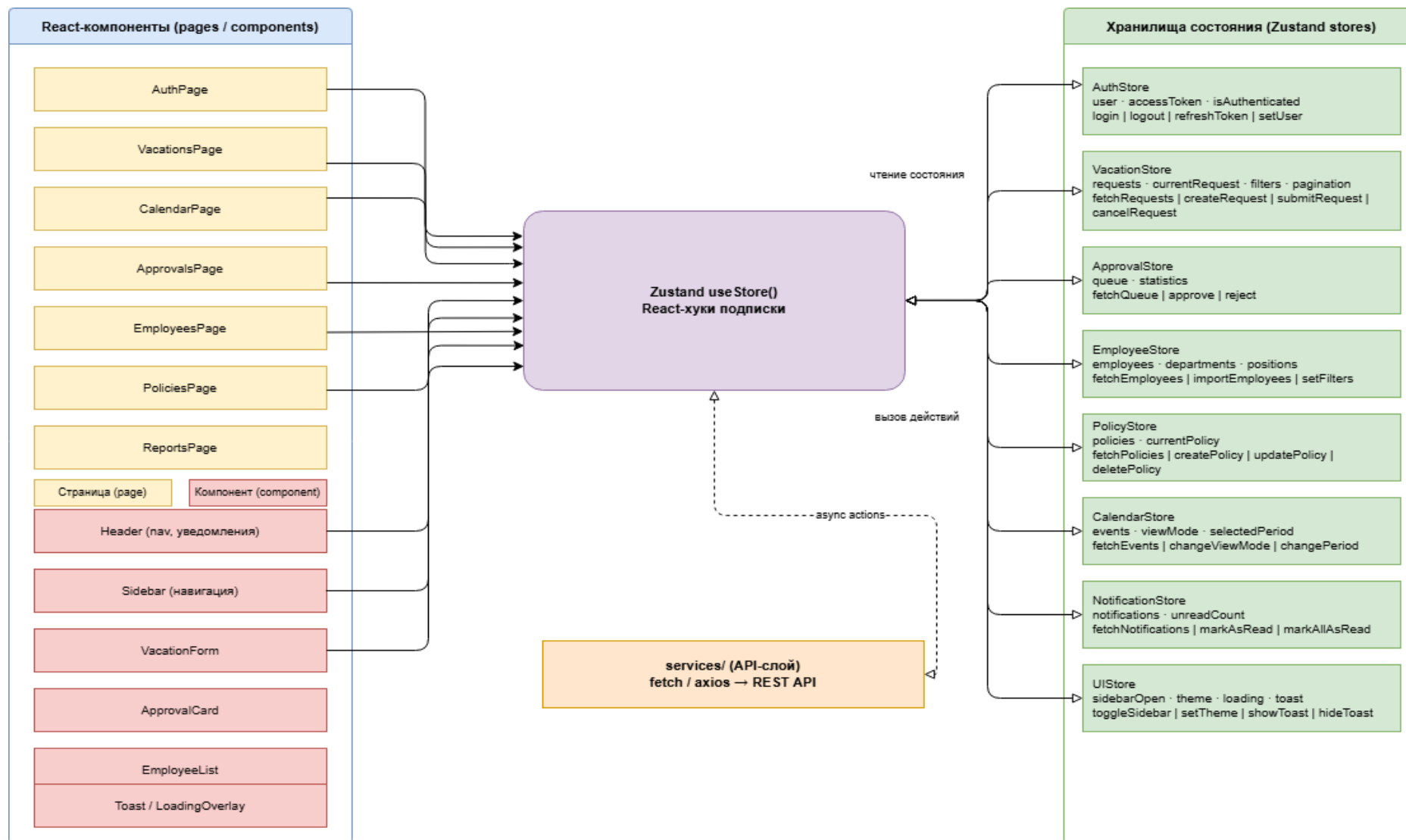


Рисунок 24 – Диаграмма компонентов состояния

В приложении реализованы отдельные хранилища для работы с авторизацией пользователей, заявками на отпуск, сотрудниками, уведомлениями, календарём отпусков и параметрами пользовательского интерфейса. Такой подход позволяет уменьшить связность между модулями и упростить сопровождение клиентской части приложения.

Таблица 11 содержит описание основных хранилищ состояния клиентского приложения.

Таблица 11 – Основные хранилища состояния

Хранилище	Состояние	Действия
AuthStore	user, accessToken, isAuthenticated	login, logout, refreshToken, setUser
VacationStore	requests, currentRequest, filters, pagination	fetchRequests, createRequest, submitRequest, cancelRequest, setFilters
ApprovalStore	queue, statistics	fetchQueue, approve, reject
EmployeeStore	employees, departments, positions	fetchEmployees, importEmployees, setFilters
PolicyStore	policies, currentPolicy	fetchPolicies, createPolicy, updatePolicy, deletePolicy
CalendarStore	events, viewMode, selectedPeriod	fetchEvents, changeViewMode, changePeriod
NotificationStore	notifications, unreadCount	fetchNotifications, markAsRead, markAllAsRead
UIStore	sidebarOpen, theme, loading, toast	toggleSidebar, setTheme, showToast, hideToast

Для выполнения асинхронных операций в хранилищах предусмотрены состояния загрузки и обработки ошибок. Это позволяет интерфейсу отображать индикаторы выполнения операций, уведомления об ошибках и актуальное состояние данных без необходимости дополнительной обработки в компонентах.

Локальное состояние отдельных элементов интерфейса, не требующее глобального хранения, реализовано с использованием встроенных механизмов

React – hook `useState` и `useReducer`. К таким данным относятся состояния форм, параметры фильтрации, открытие модальных окон и временные пользовательские данные. Использование локального состояния позволяет снизить нагрузку на глобальные хранилища и упростить структуру приложения.

Синхронизация клиентского состояния с серверной частью выполняется посредством загрузки данных при открытии страниц и выполнении пользовательских действий. Для операций изменения данных применяется механизм оптимистичного обновления интерфейса. В данном случае изменения сначала отображаются пользователю на стороне клиента, а при получении ошибки от серверной части выполняется откат состояния и вывод соответствующего уведомления.

Использование централизованного управления состоянием позволяет обеспечить предсказуемость работы интерфейса, уменьшить дублирование логики между компонентами и упростить масштабирование клиентского приложения при расширении функциональности системы.

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 [25]. В отличие от логической модели, физическая модель учитывает особенности используемой СУБД, включая типы данных, ограничения целостности и механизмы хранения данных [26].

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 12 приведено соответствие наименований сущностей обеих схем.

Таблица 12 – 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 25 представлена физическая модель базы данных.

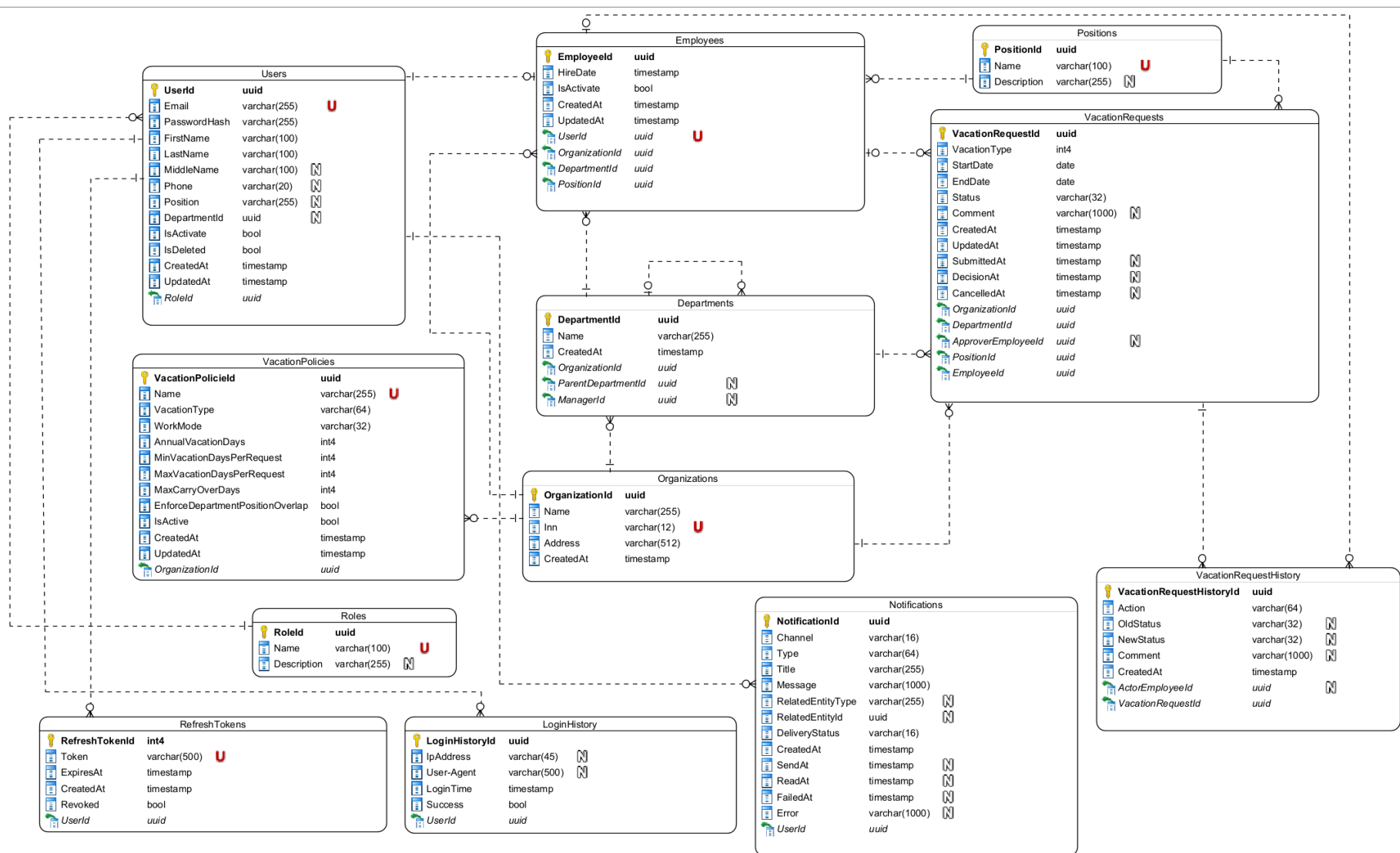


Рисунок 25 – Диаграмма физической модели базы данных системы контроля отпусков

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 репозитория. 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 [27].

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 },
        cancellationTokens: 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, cancellationTokens: ct));
}
}

```

Аналогично реализованы репозитории для остальных сущностей системы. Стоит отметить, что каждый репозиторий содержит методы, соответствующие своей предметной области. Использование слоя репозитория в сочетании с Дарпер 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 репозитория, обеспечивающий изоляцию SQL-запросов и возможность удобного расширения функционала системы.

4.2 Структура клиентской части

Клиентская часть системы реализована в виде одностраничного веб-приложения (Single Page Application) с использованием библиотеки React и языка программирования TypeScript. Структура проекта организована по принципу разделения ответственности между функциональными модулями

приложения, что позволяет упростить сопровождение и дальнейшее развитие программного обеспечения.

На рисунке 26 представлена иерархическая структура проекта.



Рисунок 26 – Структура проекта

Основная логика приложения размещена в директории `src`, содержащей ключевые модули клиентской части. Для взаимодействия с серверной частью используется директория `api`, в которой расположена конфигурация HTTP-клиента и базовые механизмы выполнения запросов. Статические ресурсы

приложения, включая изображения, иконки и шрифты, размещены в директории `assets`.

Компоненты пользовательского интерфейса сгруппированы в директории `components`. Внутри неё выделены базовые UI-компоненты (`ui`), компоненты разметки (`layout`), формы ввода данных (`forms`), компоненты отображения данных (`data-display`), а также элементы обратной связи (`feedback`), отвечающие за отображение уведомлений, сообщений об ошибках и индикаторов загрузки.

Для повторного использования логики в приложении применяются пользовательские `React-hooks`, размещённые в директории `hooks`. Вспомогательные функции, конфигурации и утилиты располагаются в директории `lib`. Страничные компоненты системы находятся в директории `pages` и разделены по функциональным областям: авторизация, управление отпусками, календарь, согласование заявок, сотрудники, политики отпусков, пользователи и отчёты.

Конфигурация маршрутизации приложения размещена в директории `routes`. Для взаимодействия с REST API используется сервисный слой (`services`), инкапсулирующий логику выполнения HTTP-запросов. Управление состоянием приложения реализовано с использованием библиотеки `Zustand`, а соответствующие хранилища размещены в директории `stores`.

Для обеспечения типобезопасности приложения используются интерфейсы и типы `TypeScript`, расположенные в директории `types`. Валидация пользовательских данных реализована с использованием библиотеки `Zod`, схемы которой размещены в директории `schemas`.

Корневым компонентом приложения является файл `App.tsx`, содержащий основную структуру клиентской части и конфигурацию маршрутизации. Точка входа приложения располагается в файле `main.tsx`, отвечающем за инициализацию `React`-приложения и подключение глобальных зависимостей.

Использование подобной структуры проекта обеспечивает модульность клиентской части, упрощает навигацию по исходному коду и позволяет

расширять функциональность системы без существенной переработки существующей архитектуры.

4.3 Реализация пользовательского интерфейса

Пользовательский интерфейс клиентской части реализован в виде набора страниц, каждая из которых предназначена для выполнения определённых задач пользователями с соответствующей ролью. Интерфейс спроектирован с учётом принципов единообразия, наглядности и минимизации когнитивной нагрузки.

4.3.1 Страница авторизации

Страница авторизации является точкой входа в систему. Реализована в виде формы с полями для ввода электронной почты и пароля. Форма содержит клиентскую валидацию: проверка формата email, минимальная длина пароля. При отправке формы выполняется запрос к API аутентификации, при успешном ответе токены сохраняются, а пользователь перенаправляется на главную страницу.

Форма авторизации представлена на рисунке 27.



Рисунок 27 – Страница авторизации

Текстовое поле 1 используется для указания базового адреса программного интерфейса серверной части. Значение по умолчанию – /api/v1. Поле предназначено для окружений с нестандартным размещением бэкенда и не требует изменений при работе в типовой конфигурации. Заполненное значение передаётся в HTTP-клиент модуля api/ при инициализации сессии;

Текстовое поле 2 реализует поле для ввода корпоративного адреса электронной почты. Значение проверяется на соответствие формату RFC 5322 средствами библиотеки Zod до отправки запроса на сервер; при несоответствии отображается встроенное сообщение об ошибке без перезагрузки страницы

Текстовое поле 3 отвечает поле ввода с маскированием символов. Обеспечивает защиту учётных данных от визуального перехвата. Содержимое поля не кэшируется браузером благодаря атрибуту autocomplete=«current-password»

Основная кнопка 4. При нажатии инициирует вызов действия login хранилища AuthStore, которое выполняет HTTP-запрос к эндпоинту POST /api/v1/auth/login. В случае успешного ответа сервера токен доступа сохраняется в хранилище, флаг isAuthenticated принимает значение true, и пользователь перенаправляется на главную страницу приложения. Над кнопкой отображается информационная метка «Сессия завершена», сигнализирующая о причине появления формы

Таким образом, форма авторизации обеспечивает как функциональные требования к идентификации пользователя, так и технические требования к конфигурируемости клиентского приложения.

4.3.2 Личный кабинет сотрудника

Личный кабинет сотрудника предназначен для взаимодействия пользователя с системой контроля отпусков и предоставляет доступ к основным функциям управления заявками. На данной странице сотрудник может просматривать информацию о доступных отпускных днях, отслеживать статус созданных заявок, создавать новые запросы на отпуск, а также получать

уведомления о результатах согласования. Интерфейс личного кабинета разработан с учётом принципов удобства использования и обеспечивает быстрый доступ к наиболее востребованным функциям системы.

Интерфейс личного кабинета сотрудника представлен на рисунке 28.

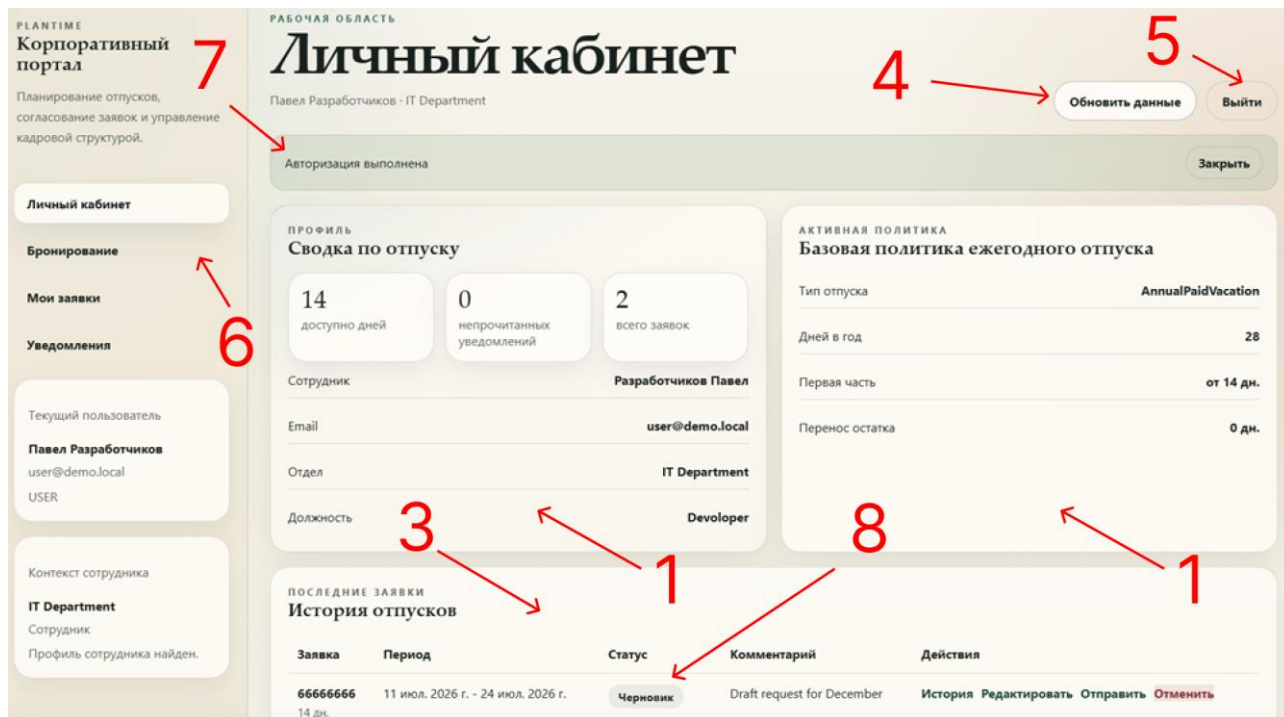


Рисунок 28 – Интерфейс личного кабинета

Блок под цифрой 1 – «Активная политика» – показывает параметры отпускной политики, которая назначена текущему сотруднику. В нём отображается тип отпуска, количество дней в год, минимальная продолжительность первой части и перенос остатка. Данные загружаются при открытии страницы и недоступны для редактирования в этом разделе. Блок нужен для того, чтобы сотрудник мог сразу видеть условия своего отпуска, не обращаясь к кадровой службе.

Элемент под цифрой 2 – столбец «Статус» в таблице заявок. Для каждой записи отображается текущее состояние в виде цветного тега. На скриншоте заявка имеет статус «Черновик» (Draft). Цветовое выделение позволяет быстро ориентироваться в списке заявок – не нужно открывать каждую запись отдельно, чтобы понять её состояние.

Элемент под цифрой 3 – карточка профиля в блоке «Сводка по отпуску». В ней выводятся имя сотрудника, почта, отдел и должность. Данные берутся из состояния приложения, которое обновляется при авторизации. Карточка удобна тем, что сотрудник видит свои реквизиты прямо на главной странице, без перехода в отдельный раздел профиля.

Кнопка под цифрой 4 – «Обновить данные» – делает повторный запрос к серверу и обновляет информацию профиля без перезагрузки страницы. Это нужно в ситуации, когда администратор изменил данные сотрудника в ходе текущей рабочей сессии – кнопка позволяет получить актуальные сведения сразу.

Кнопка под цифрой 5 – «Выйти» – завершает сеанс работы. При нажатии выполняется выход из учётной записи: токен доступа удаляется, сессия сбрасывается, после чего приложение перенаправляет пользователя на страницу авторизации. Корректный выход из системы важен с точки зрения безопасности данных.

Элемент под цифрой 6 – боковое меню навигации. В нём перечислены основные разделы приложения: «Личный кабинет», «Бронирование», «Мои заявки», «Уведомления». Ниже пунктов меню отображается информация о текущем пользователе и его принадлежности к подразделению. Это позволяет всегда видеть, под какой учётной записью выполнен вход, не переходя в профиль.

Элемент под цифрой 7 – баннер «Авторизация выполнена», который появляется сразу после успешного входа в систему. Он автоматически скрывается через несколько секунд или по нажатию кнопки «Закрыть». Такой тип уведомления выбран потому, что он не блокирует интерфейс – пользователь может сразу начать работу, не закрывая никаких диалогов.

Элемент под цифрой 8 – три счётчика в блоке «Профиль»: доступные дни отпуска, непрочитанные уведомления и общее количество заявок. Значения обновляются при изменении данных в приложении. Вынос этих показателей на

главный экран позволяет сотруднику оценить текущую ситуацию сразу при открытии личного кабинета.

Таким образом, страница «Личный кабинет» объединяет данные из нескольких хранилищ состояния и представляет собой центральную точку взаимодействия сотрудника с системой. Реализованная компоновка интерфейса обеспечивает доступ к ключевым функциям и персональным данным в рамках единого экрана, что соответствует принципу минимизации навигационных переходов при выполнении типовых сценариев использования.

4.3.3 Форма создания заявки на отпуск

На рисунке 29 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.

НОВАЯ ЗАЯВКА
Бронирование отпуска

Тип отпуска: Ежегодный оплачиваемый отпуск

Дата начала: 03.07.2026

Дата окончания: 16.07.2026

Комментарий

1 2 3

Проверить правила Сохранить черновик **Отправить** Сбросить форму

ПРОВЕРКА
Результаты валидации

4

Нажми «Проверить правила», чтобы запросить backend-модули политики отпусков и определения конфликтов.

Рисунок 29 – Форма создания заявки на отпуск

Кнопка под цифрой 1 запускает предварительную проверку заявки. Сервер проверяет соответствие отпускной политике и наличие конфликтов с другими заявками. Пользователь видит результат проверки, при этом сама заявка никуда не отправляется.

Кнопка под цифрой 2 сохраняет заявку в статусе «Черновик». Никаких проверок при этом не выполняется – заявка просто фиксируется в системе, чтобы к ней можно было вернуться позже.

Кнопка под цифрой 3 отправляет заявку на согласование. Перед этим выполняется полная цепочка проверок: данные заявки, правила политики, пересечения с другими периодами и определение согласующего. Если все проверки пройдены – заявка переходит в статус ожидания решения.

Блок под цифрой 4 показывает итог выполненной операции. В зависимости от результата там отображается либо список ошибок, либо подтверждение того, что заявка обработана успешно.

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

4.3.4 Календарь отпусков команды

Страница «Календарь отпусков команды» предназначена для централизованного контроля отпускных периодов сотрудников подразделения и поддержки процессов управленческого планирования. На данной странице руководитель может анализировать распределение отпусков в рамках выбранного периода, отслеживать количество заявок, ожидающих согласования, оценивать уровень загрузки отдела, а также просматривать временное распределение отсутствий сотрудников на интерактивной диаграмме Ганта. Интерфейс страницы разработан с учётом требований к наглядности и оперативности восприятия информации, что обеспечивает быстрый доступ к ключевым показателям и упрощает принятие решений при согласовании новых отпускных заявок.

Интерфейс страницы представлен на рисунке 30.

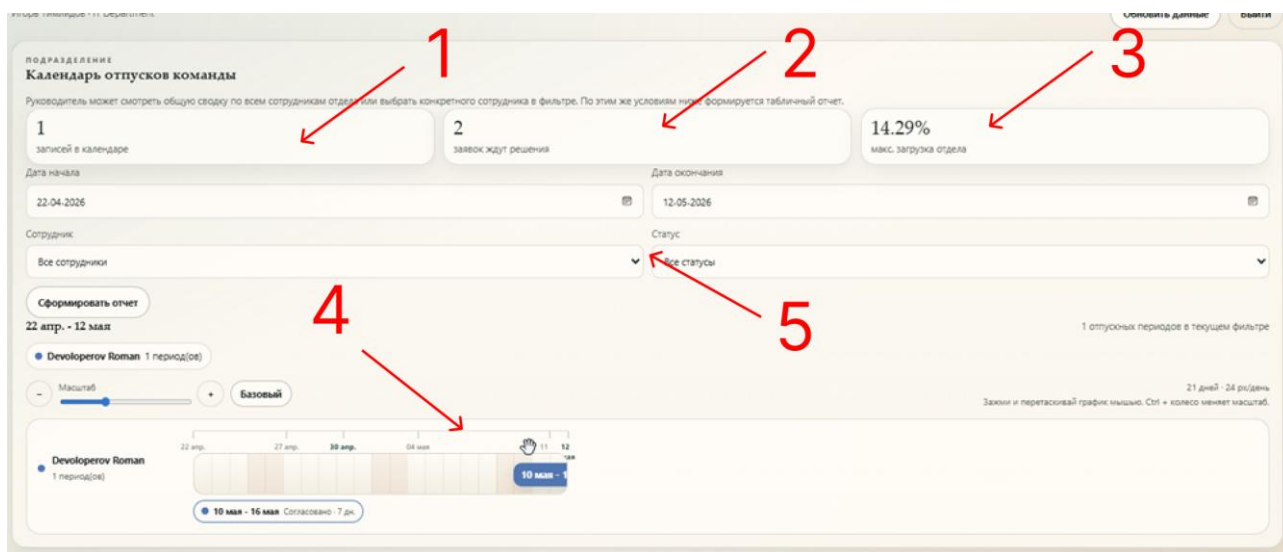


Рисунок 30 – Календарь отпусков команды

Элемент под цифрой 1 – счётчик «Записей в календаре» – отображает общее количество отпускных периодов, попадающих в заданный диапазон дат с учётом активных фильтров по сотруднику и статусу. Значение формируется на основе коллекции events хранилища CalendarStore и пересчитывается при каждом изменении параметров фильтрации без повторного обращения к серверной части. Наличие данного счётчика позволяет руководителю мгновенно оценить загруженность периода ещё до визуального анализа диаграммы.

Элемент под цифрой 2 – счётчик «Заявок ждут решения» – отображает количество заявок подразделения, находящихся в статусе ожидания согласования. Значение синхронизируется с очередью хранилища ApprovalStore и служит индикатором управленческой нагрузки на руководителя.

Элемент под цифрой 3 – показатель «Макс. загрузка отдела» – показывает максимальный процент сотрудников подразделения, которые одновременно отсутствуют в выбранный период. Значение считается на основе пересечений отпускных периодов относительно общей численности отдела. Метрика помогает руководителю заранее увидеть, в какие даты команда будет наименее укомплектована, и учесть это при рассмотрении новых заявок.

Элемент под цифрой 4 – диаграмма Ганта – отображает отпуска сотрудников подразделения на временной шкале. Каждая строка соответствует отдельному сотруднику, цветные отрезки показывают периоды отсутствия с датами и количеством дней. Масштаб диаграммы регулируется ползунком, шкалу можно перетаскивать мышью. Диаграмма Ганта выбрана потому, что это привычный формат для планирования – большинство пользователей понимают её без дополнительных пояснений.

Элемент под цифрой 5 – выпадающий список «Статус» – фильтрует записи на диаграмме по статусу заявки. Доступные значения: черновик, на рассмотрении, согласовано, отклонено. При выборе значения фильтр применяется сразу ко всем элементам страницы – счетчикам, диаграмме и таблице – без перезагрузки. Это позволяет руководителю быстро отобрать, например, только согласованные заявки или только те, что ещё ждут решения.

Таким образом, страница «Календарь отпусков команды» предоставляет руководителю возможность взаимодействия с отпусками сотрудников. Совокупность сводных метрик, гибких фильтров и интерактивной временной диаграммы обеспечивает целостное представление о состоянии отпусков команды в рамках единого экрана, что соответствует требованиям к эргономике управленческих интерфейсов корпоративных информационных систем.

4.3.5 Интерфейс согласования заявок

Страница «Отчёт по отпускам отдела» предназначена для формирования аналитической сводки по отпускным заявкам подразделения в разрезе выбранного периода, статуса и конкретного сотрудника. На данной странице руководитель может получить агрегированную статистику по заявкам, проанализировать детализированную таблицу отпусков с указанием дат решений, выявить коллизии, требующие оперативного вмешательства, а также перейти непосредственно к согласованию входящих заявок. Совмещение отчётной и управленческой функциональности в рамках единого экрана

обеспечивает минимизацию навигационных переходов и повышает оперативность принятия кадровых решений.

Интерфейс окна представлен на рисунке 31.

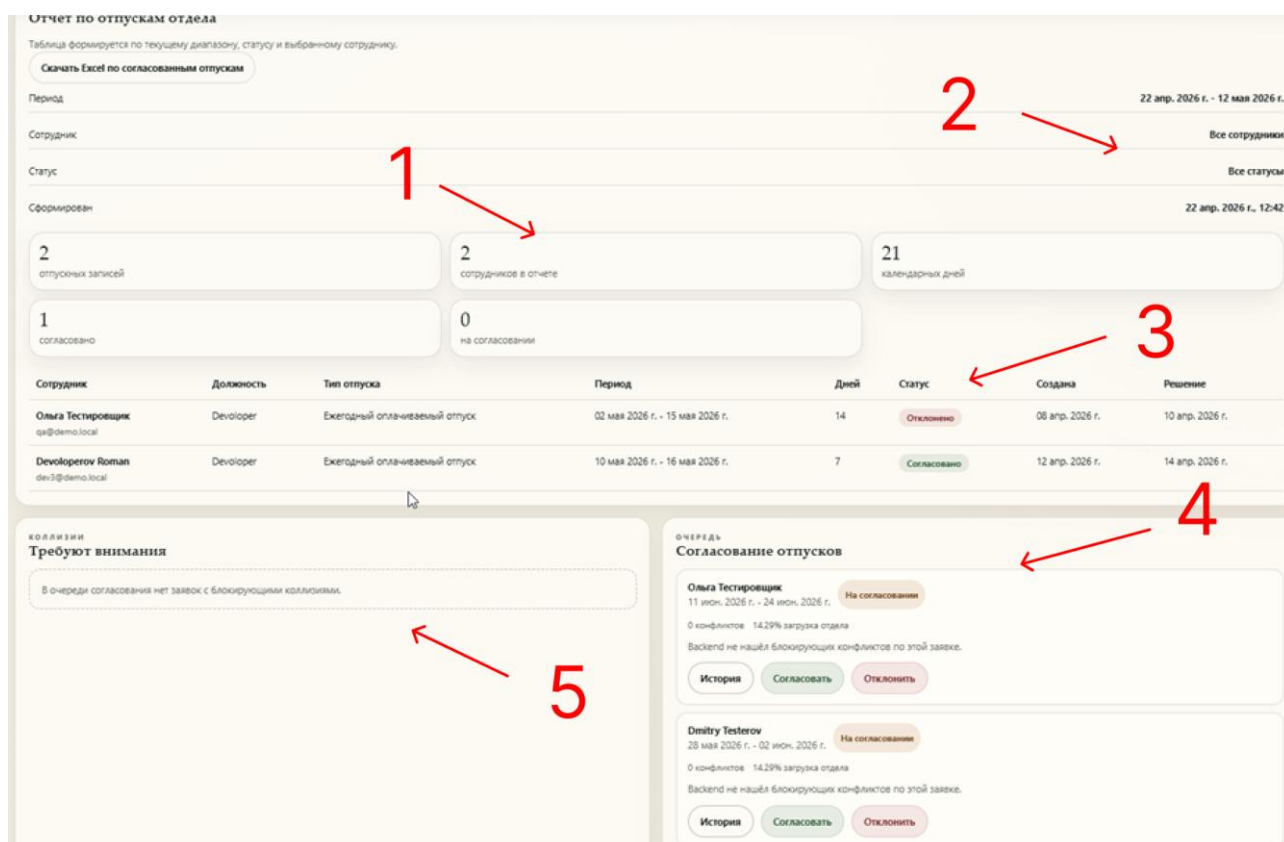


Рисунок 31 – Интерфейс согласования заявок

Элемент под цифрой 1 – блок сводных показателей – выводит пять цифр по текущей выборке: количество отпускных записей, число сотрудников в отчете, суммарные календарные дни, а также сколько заявок согласовано и сколько ещё на рассмотрении. Все значения берутся из уже загруженных данных, без дополнительных запросов к серверу. За счет этого руководитель сразу видит общую картину по периоду, не считая строки в таблице вручную.

Элемент под цифрой 2 – панель параметров отчета – показывает, по каким фильтрам сформирован отчет: период, сотрудник, статус заявок, а также когда именно отчет был сформирован. Поля доступны только для просмотра. Дата и время формирования нужны для того, чтобы понять, насколько актуальны данные – если прошло много времени, можно обновить отчет.

Элемент под цифрой 3 – таблица заявок – содержит подробные сведения по каждой записи: имя и должность сотрудника, тип отпуска, период, количество дней, статус, дату создания и дату решения. Статус выделен цветным тегом, поэтому не нужно читать каждую строку – достаточно одного взгляда, чтобы понять состояние заявки. Табличный формат удобен и для выгрузки данных во внешние системы.

Элемент под цифрой 4 – блок «Согласование отпусков» – это очередь заявок, которые ждут решения. В каждой карточке видно: кто запрашивает отпуск, на какой период, сколько конфликтов выявлено, какой процент загрузки отдела. Там же три кнопки – «История», «Согласовать» и «Отклонить». То, что очередь согласования находится прямо на странице отчета, удобно – не нужно переходить в другой раздел, чтобы принять решение.

Элемент под цифрой 5 – блок «Коллизии / Требуют внимания» – показывает заявки, по которым сервер нашел проблемы: пересечения периодов, нарушения политики или слишком высокий процент одновременного отсутствия. Если проблемных заявок нет, блок показывает соответствующее сообщение. Отдельный блок для конфликтов нужен для того, чтобы руководитель мог быстро найти проблемные случаи, не листая всю таблицу.

Таким образом, страница «Отчёт по отпускам отдела» объединяет функции аналитической отчётности и оперативного согласования в единый рабочий интерфейс. Реализованная компоновка обеспечивает полный управленческий цикл – от анализа статистики до вынесения решения по заявке – без перехода между разделами системы, что соответствует требованиям к эргономике корпоративных информационных систем.

4.3.6 Интерфейс HR-специалиста

Страница «HR кабинет» предназначена для выполнения административных операций с кадровой структурой организации: создания карточек сотрудников, управления контекстом подразделения и пакетного импорта персонала из внешних источников данных. Доступ к данной странице

предоставляется исключительно пользователям с ролью HR, что обеспечивается на уровне защищённых маршрутов клиентского приложения. Совмещение инструментов единичного и массового добавления сотрудников в рамках одного интерфейса позволяет сократить время первоначального наполнения системы данными и снизить трудоёмкость последующего сопровождения кадрового реестра.

На рисунке 32 представлен интерфейс HR-сотрудника.

The screenshot shows the 'HR кабинет' (HR cabinet) interface. At the top, there's a header with 'РАБОЧАЯ ОБЛАСТЬ' (Working Area) and 'HR кабинет'. Below it, 'Марина Кадрова - HR Department' is displayed. On the right, there are buttons for 'Обновить данные' (Update data) and 'Выйти' (Logout). A green bar indicates 'Авторизация выполнена' (Authorization completed) with a 'Закреть' (Close) button. The main area is divided into two panels. The left panel, labeled '1', is titled 'КОНТЕКСТ HR кабинет' and contains a dropdown for 'PlanTime Demo Company', a 'Применить контекст' (Apply context) button, another dropdown for 'PlanTime Demo Company', an 'Excel-файл' (Excel file) section with a 'Обзор...' (Browse...) button (labeled '3'), and an 'Импортировать сотрудников' (Import employees) button. Below this is a text box for import results (labeled '4'). The right panel, labeled '2', is titled 'КАРТОЧКА Сотрудник' (Employee card) and contains fields for 'Пользователь' (User) and 'Дата найма' (Hire date), a 'Подразделение' (Department) dropdown, and a 'Должность' (Position) dropdown. It also has a 'Сотрудник активен' (Employee active) checkbox and buttons for 'Создать сотрудника' (Create employee) and 'Очистить форму' (Clear form). At the bottom, there's a table header with columns: 'Сотрудник', 'Подразделение', 'Должность', 'Дата', 'Статус', and 'Действия'.

Рисунок 32 – Интерфейс HR-специалиста

Элемент под цифрой 1 – блок «HR кабинет / Контекст» – предоставляет инструменты управления организационным контекстом и массового импорта сотрудников. Блок содержит выпадающий список выбора организации, кнопку «Применить контекст», переключаящую рабочее пространство HR-специалиста на выбранную компанию, а также отдельную секцию импорта с выбором компании-получателя и полем загрузки Excel-файла. Применение контекста фиксируется в хранилище EmployeeStore и распространяется на все последующие операции сессии, что исключает необходимость повторного указания организации при создании каждого сотрудника в отдельности.

Элемент под цифрой 2 – карточка «Сотрудник» – представляет собой форму единичного создания кадровой записи. Она содержит следующие

элементы управления: выпадающий список «Выбери пользователя» для привязки карточки к существующей учётной записи системы; поле «Дата найма» с компонентом выбора даты; выпадающие списки «Подразделение» и «Должность», значения которых синхронизированы с данными хранилища EmployeeStore; флажок «Сотрудник активен», определяющий статус записи; кнопку «Создать сотрудника», вызывающую метод `importEmployees` хранилища с единичной записью после прохождения валидации; кнопку «Очистить форму», сбрасывающую все поля к значениям по умолчанию без сохранения изменений. Разделение учётной записи пользователя и карточки сотрудника обеспечивает независимость процессов регистрации и кадрового оформления, что соответствует принципу единственной ответственности на уровне бизнес-логики.

Элемент под цифрой 3 – секция загрузки Excel-файла – обеспечивает пакетный импорт сотрудников из табличного файла. Поле загрузки принимает файлы формата .xlsx; ожидаемая структура колонок описана непосредственно в интерфейсе: Email, Password, FirstName, LastName, MiddleName, Phone, Role, Department, Position, HireDate, IsActive с поддержкой как латинских, так и русскоязычных заголовков. Кнопка «Импортировать сотрудников» передаёт выбранный файл в метод `importEmployees` хранилища EmployeeStore, который направляет его на серверную часть для построчной обработки. Поддержка пользовательских заголовков колонок снижает требования к подготовке входных данных и упрощает интеграцию с выгрузками из внешних кадровых систем.

Элемент под цифрой 4 – блок результатов импорта – отображается после завершения обработки файла и содержит построчный отчёт с результатом обработки каждой записи: статусом создания, наименованием сотрудника, подразделением, должностью и датой. До загрузки файла блок содержит информационное сообщение-заглушку, сигнализирующее о его назначении. Детализированный отчёт по каждой строке файла позволяет HR-специалисту незамедлительно выявить и исправить некорректные записи без повторной

загрузки всего массива данных, что существенно сокращает время первичного наполнения кадрового реестра.

Таким образом, страница «HR кабинет» реализует полный цикл управления кадровым составом – от создания единичной записи до массового импорта из внешних источников. Гибкость инструментов импорта в сочетании с детализированной обратной связью по результатам обработки обеспечивает надёжность первичного наполнения системы данными и соответствует требованиям к интерфейсам административного уровня корпоративных информационных систем.

4.3.7 Интерфейс администратора

На рисунке 33 представлена часть интерфейса административной панели, предназначенная для управления учётными записями пользователей и кадровыми карточками сотрудников. Интерфейс объединяет инструменты создания пользователей, просмотра справочника сотрудников, управления ролями доступа и оформления кадровых записей. Разделение функций между блоками позволяет разграничить процессы администрирования учётных записей и ведения кадровой информации, что упрощает сопровождение данных и соответствует принципу модульной организации интерфейса.

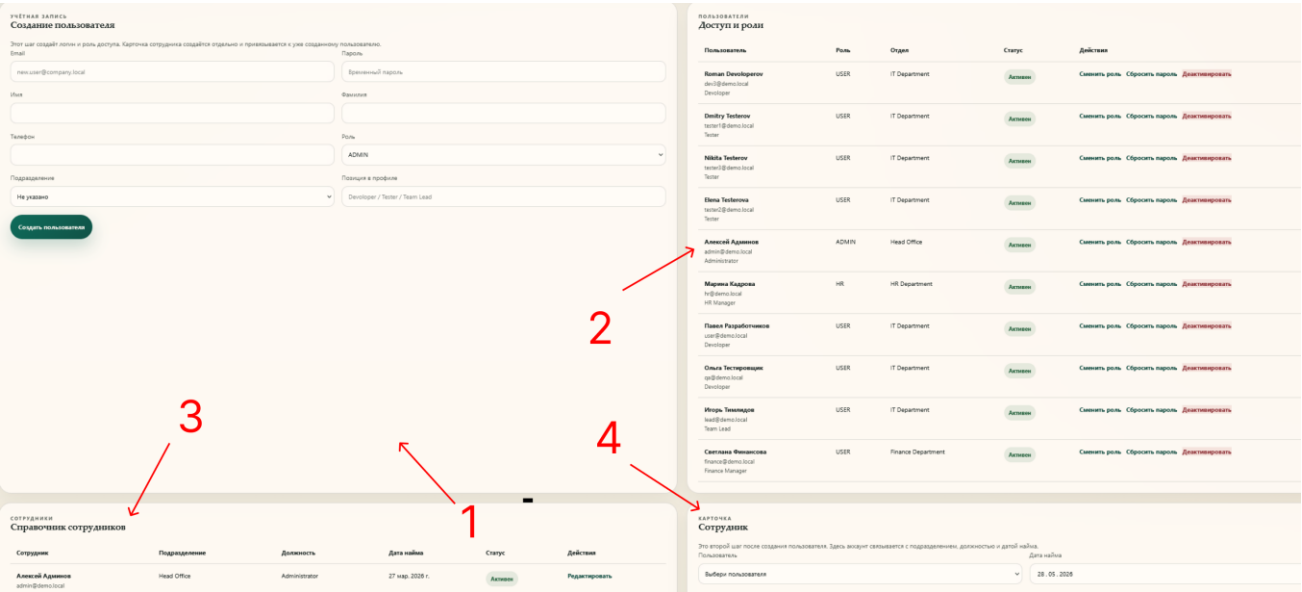


Рисунок 33 – Интерфейс администратора

Элемент под цифрой 1 – часть интерфейса «Создание пользователя» – представляет собой форму регистрации новой учётной записи. Форма содержит поля ввода электронной почты, временного пароля, имени, фамилии, телефона, роли пользователя, подразделения и должности в профиле. Кнопка «Создать пользователя» инициирует создание новой учётной записи с последующим сохранением данных в системе. Разделение этапа создания пользователя и кадровой карточки позволяет независимо управлять аутентификационными и кадровыми данными сотрудников.

Элемент под цифрой 2 – часть интерфейса «Доступы и роли» – предназначен для управления учётными записями пользователей системы. Блок содержит таблицу пользователей с отображением имени, адреса электронной почты, роли, подразделения и статуса активности учётной записи. Для каждой записи предусмотрены действия «Сменить роль», «Сбросить пароль» и «Деактивировать», позволяющие выполнять административные операции управления доступом. Использование единой таблицы пользователей обеспечивает централизованный контроль прав доступа и упрощает администрирование системы.

Элемент под цифрой 3 – справочник сотрудников – показывает список кадровых записей организации. Таблица содержит основные данные: имя сотрудника, подразделение, должность, дату найма и статус. Для каждой строки есть кнопка «Редактировать», по которой можно перейти к изменению данных. Раздел нужен для того, чтобы HR-специалист мог просматривать весь кадровый состав в одном месте, не открывая карточку каждого сотрудника отдельно.

Элемент под цифрой 4 – карточка сотрудника – используется для создания и редактирования кадровой записи. В форме есть выпадающий список для выбора пользователя, поле даты найма, выбор подразделения и должности, флажок активности и кнопки «Создать сотрудника» и «Очистить форму». Привязка к существующей учётной записи сделана намеренно – это разделяет управление доступом и кадровый учет, чтобы их можно было вести независимо друг от друга.

Описанные блоки вместе закрывают основные задачи администрирования: управление пользователями и ведение кадрового реестра. Разбивка на отдельные функциональные блоки упрощает работу с интерфейсом и снижает вероятность ошибок при выполнении административных операций.

4.4 Реализация взаимодействия с API

Для взаимодействия клиентской части с сервером используется единый HTTP-клиент на базе библиотеки `Axios`. Все запросы проходят через одну точку – это удобно, потому что не нужно в каждом модуле отдельно настраивать заголовки или обработку ошибок. Токен авторизации добавляется автоматически, что избавляет от дублирования кода в разных частях приложения.

4.4.1 Конфигурация HTTP-клиента

В листинге 2 показана конфигурация HTTP-клиента на основе `Axios`. В ней заданы базовые настройки для работы с сервером – адрес API, таймаут, заголовки – а также обработчики запросов и ответов.

Листинг 2 – Конфигурация HTTP-клиента[27]

```
import axios from 'axios';
import { useAuthStore } from '@/stores/authStore';

const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Перехватчик запросов – добавление токена авторизации
apiClient.interceptors.request.use((config) => {
  const token = useAuthStore.getState().accessToken;
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Перехватчик ответов – обработка ошибок и обновление токена
apiClient.interceptors.response.use(
  (response) => response.data,
  async (error) => {
    const originalRequest = error.config;
```

```

    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;
      try {
        const refreshToken = localStorage.getItem('refreshToken');
        const response = await axios.post(
          `${import.meta.env.VITE_API_URL}/auth/refresh`,
          { refreshToken }
        );
        const { accessToken } = response.data;
        useAuthStore.getState().setAccessToken(accessToken);
        originalRequest.headers.Authorization = `Bearer ${accessToken}`;
        return apiClient(originalRequest);
      } catch (refreshError) {
        useAuthStore.getState().logout();
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
};

export default apiClient;

```

Представленная конфигурация обеспечивает централизованное взаимодействие клиентской части с REST API серверной части системы. В процессе выполнения запроса HTTP-клиент автоматически добавляет JWT-токен авторизации в заголовок Authorization, что позволяет выполнять аутентифицированные запросы без дублирования логики в отдельных модулях приложения.

Дополнительно реализован механизм автоматического обновления access-токена при получении ответа сервера с кодом 401 Unauthorized. В случае истечения срока действия access-токена выполняется запрос на получение нового токена с использованием refresh-токена, сохранённого в локальном хранилище браузера. После успешного обновления токена исходный запрос повторно отправляется на сервер.

При невозможности обновления токена выполняется очистка пользовательской сессии и перенаправление пользователя на страницу авторизации. Такой подход позволяет обеспечить непрерывность работы пользователя с системой и повысить удобство использования клиентского приложения.

Таким образом, использование централизованного HTTP-клиента на базе Axios позволило реализовать единый механизм взаимодействия клиентской части с серверным API, обеспечить автоматическую обработку авторизации и повысить надёжность работы приложения за счёт механизма обновления JWT-токенов и централизованной обработки ошибок.

4.4.2 Реализация сервисного слоя

В листинге 3 представлен пример реализации сервисного модуля, предназначенного для взаимодействия клиентской части с серверным API системы управления отпусками.

Листинг 3 – Сервисный модуль VacationService

```
import apiClient from '../apiClient';
import type {
  VacationRequest,
  CreateVacationRequest,
  VacationFilters,
  PaginatedResponse
} from '@types';

export const VacationService = {
  async getList(filters: VacationFilters):
  Promise<PaginatedResponse<VacationRequest>> {
    const response = await apiClient.get('/vacation-requests', {
      params: filters
    });
    return response.data;
  },

  async getById(id: string): Promise<VacationRequest> {
    const response = await apiClient.get(`/vacation-requests/${id}`);
    return response.data;
  },

  async create(data: CreateVacationRequest): Promise<VacationRequest> {
    const response = await apiClient.post('/vacation-requests', data);
    return response.data;
  },

  async submit(id: string): Promise<VacationRequest> {
    const response = await apiClient.post(`/vacation-requests/${id}/submit`);
    return response.data;
  },

  async cancel(id: string, reason?: string): Promise<VacationRequest> {
    const response = await apiClient.post(`/vacation-requests/${id}/cancel`, {
      reason
    });
    return response.data;
  },

  async checkConflicts(id: string): Promise<ConflictCheckResult> {
```

```

const response = await apiClient.post(`/vacation-requests/${id}/check`);
return response.data;
}
};

```

Сервисный модуль содержит методы для работы с заявками на отпуск и отвечает за все обращения к API. Каждый метод берет на себя формирование запроса, поэтому компонентам и хранилищам не нужно знать детали работы с сетью – они просто вызывают нужный метод и получают данные.

Вынесение работы с API в отдельный слой упрощает сопровождение проекта – если поменяется эндпоинт или формат данных, правки вносятся в одном месте, а не по всему коду. Использование TypeScript-интерфейсов помогает избежать ошибок при обработке ответов сервера, так как структура данных заранее описана и проверяется на этапе компиляции.

За счет такой организации логику работы с API легко переиспользовать в разных частях приложения и при необходимости расширять без изменения существующего кода.

4.4.3 Обработка ошибок и индикация загрузки

Для обеспечения корректного взаимодействия пользователя с системой в клиентской части реализованы механизмы обработки ошибок и индикации загрузки данных. Данные механизмы позволяют повысить удобство использования приложения, обеспечить информирование пользователя о текущем состоянии операций и снизить вероятность возникновения ошибок при работе с системой.

Индикация загрузки реализована на уровне хранилищ состояния. Перед выполнением асинхронного запроса соответствующий модуль состояния устанавливает флаг загрузки, который используется компонентами интерфейса для отображения индикаторов выполнения операции. После получения ответа от серверной части состояние загрузки автоматически сбрасывается. В интерфейсе применяются различные элементы визуальной индикации, включая спиннеры, skeleton-компоненты и блокировку элементов формы на время выполнения запроса.

Обработка ошибок реализована с учётом различных сценариев взаимодействия с серверной частью системы. При возникновении ошибок валидации с кодом ответа 400 Bad Request сообщения об ошибках отображаются непосредственно рядом с соответствующими полями формы. При получении ответа 401 Unauthorized выполняется автоматическое перенаправление пользователя на страницу авторизации. Ошибки доступа 403 Forbidden сопровождаются отображением уведомления о недостаточности прав для выполнения операции.

При возникновении ошибок серверной части с кодами 500 и выше пользователю отображается уведомление с предложением повторить операцию позднее. В случае отсутствия сетевого соединения или недоступности сервера система выводит сообщение о проблемах подключения. Такой подход позволяет обеспечить понятную обратную связь с пользователем и повысить устойчивость клиентского приложения к различным ошибочным ситуациям.

4.5 Реализация маршрутизации и защиты маршрутов

Маршрутизация клиентской части системы реализована с использованием библиотеки React Router. Конфигурация маршрутов организована по иерархическому принципу и обеспечивает разграничение доступа пользователей к функциональным разделам приложения в зависимости от их роли в системе.

В листинге 4 представлена конфигурация маршрутов клиентского приложения.

Листинг 4 — Конфигурация маршрутов

```
import { createBrowserRouter, Navigate } from 'react-router-dom';
import { ProtectedRoute } from '@components/auth/ProtectedRoute';
import { RoleGuard } from '@components/auth/RoleGuard';
import { MainLayout } from '@components/layout/MainLayout';

const router = createBrowserRouter([
  {
    path: '/login',
    element: <LoginPage />,
  },
  {
    path: '/',
    element: (
      <ProtectedRoute>
```

```

        <MainLayout />
      </ProtectedRoute>
    ),
    children: [
      {
        index: true,
        element: <DashboardPage />,
      },
      {
        path: 'profile',
        element: <ProfilePage />,
      },
      {
        path: 'vacations',
        element: <VacationListPage />,
      },
      {
        path: 'vacations/new',
        element: <VacationCreatePage />,
      },
      {
        path: 'vacations/:id',
        element: <VacationDetailPage />,
      },
      {
        path: 'calendar',
        element: (
          <RoleGuard allowedRoles={['manager', 'admin']}>
            <CalendarPage />
          </RoleGuard>
        ),
      },
      {
        path: 'approvals',
        element: (
          <RoleGuard allowedRoles={['manager', 'admin']}>
            <ApprovalQueuePage />
          </RoleGuard>
        ),
      },
      {
        path: 'employees',
        element: (
          <RoleGuard allowedRoles={['hr', 'admin']}>
            <EmployeeListPage />
          </RoleGuard>
        ),
      },
      {
        path: 'policies',
        element: (
          <RoleGuard allowedRoles={['hr', 'admin']}>
            <PolicyListPage />
          </RoleGuard>
        ),
      },
      {
        path: 'users',
        element: (
          <RoleGuard allowedRoles={['admin']}>
            <UserListPage />
          </RoleGuard>
        ),
      },
    ],
  },

```

```

    },
    {
      path: 'notifications',
      element: <NotificationPage />,
    },
  ],
},
{
  path: '*',
  element: <NotFoundPage />,
},
]);

```

```
export default router;
```

Конфигурация маршрутизации включает как общедоступные, так и защищённые маршруты приложения. Доступ к основным разделам системы осуществляется только после прохождения процедуры аутентификации. Для этого используется компонент `ProtectedRoute`, выполняющий проверку наличия access-токена в хранилище состояния приложения. В случае отсутствия токена пользователь автоматически перенаправляется на страницу авторизации.

Дополнительно реализован механизм разграничения доступа на основе ролей пользователей. Компонент `RoleGuard` проверяет соответствие роли пользователя разрешённым ролям для конкретного раздела системы. При отсутствии необходимых прав доступа пользователю отображается страница с ошибкой доступа.

Использование вложенной маршрутизации позволяет объединить защищённые страницы в общую структуру интерфейса с едиными элементами навигации и разметки. Такой подход обеспечивает централизованное управление доступом, упрощает сопровождение клиентской части и позволяет гибко расширять структуру приложения при добавлении новых разделов системы.

Таким образом, реализованная система маршрутизации обеспечивает безопасную навигацию между страницами приложения, разграничение доступа в соответствии с ролями пользователей и удобную организацию структуры клиентской части системы.

4.6 Реализация управления состоянием

Управление состоянием клиентской части системы реализовано с использованием библиотеки Zustand. Каждое хранилище состояния располагается в отдельном модуле и содержит данные приложения, методы их изменения, а также функции получения производной информации. Такой подход позволяет разделить состояние по функциональным областям системы и обеспечить удобство сопровождения клиентской части.

В листинге 5 представлена реализация хранилища состояния для работы с заявками на отпуск.

Листинг 5 – Хранилище VacationStore

```
interface VacationState {
  // Состояние
  requests: VacationRequest[];
  currentRequest: VacationRequest | null;
  filters: VacationFilters;
  loading: boolean;
  error: string | null;
  totalCount: number;

  // Действия
  fetchRequests: () => Promise<void>;
  fetchRequestById: (id: string) => Promise<void>;
  createRequest: (data: CreateVacationRequest) => Promise<void>;
  submitRequest: (id: string) => Promise<void>;
  setFilters: (filters: Partial<VacationFilters>) => void;
  clearError: () => void;
}

export const useVacationStore = create<VacationState>((set, get) => ({
  requests: [],
  currentRequest: null,
  filters: { status: 'all', page: 1, pageSize: 10 },
  loading: false,
  error: null,
  totalCount: 0,

  fetchRequests: async () => {
    set({ loading: true, error: null });
    try {
      const response = await VacationService.getList(get().filters);
      set({
        requests: response.items,
        totalCount: response.totalCount,
        loading: false
      });
    } catch (error: any) {
      set({
        error: error.message || 'Ошибка загрузки заявок',
        loading: false
      });
    }
  }
});
```

```

    }
  },

  createRequest: async (data) => {
    set({ loading: true, error: null });
    try {
      const newRequest = await VacationService.create(data);
      set((state) => ({
        requests: [newRequest, ...state.requests],
        loading: false,
      }));
    } catch (error: any) {
      set({ error: error.message || 'Ошибка создания заявки', loading: false });
      throw error;
    }
  },

  submitRequest: async (id) => {
    set({ loading: true, error: null });
    try {
      const updated = await VacationService.submit(id);
      set((state) => ({
        requests: state.requests.map((r) =>
          r.id === id ? updated : r
        ),
        loading: false,
      }));
    } catch (error: any) {
      set({ error: error.message || 'Ошибка отправки заявки', loading: false });
      throw error;
    }
  },

  setFilters: (filters) => {
    set((state) => ({ filters: { ...state.filters, ...filters } }));
    get().fetchRequests();
  },

  clearError: () => set({ error: null }),
}));

```

Представленное хранилище реализует централизованное управление состоянием заявок на отпуск и демонстрирует организацию асинхронного взаимодействия с серверной частью системы. Перед выполнением запроса в состоянии устанавливается флаг загрузки, после чего выполняется обращение к сервисному слою. При успешном выполнении запроса данные сохраняются в состоянии приложения, а при возникновении ошибки формируется сообщение, отображаемое пользователю в интерфейсе.

Использование Zustand позволяет минимизировать объём шаблонного кода и упростить работу с глобальным состоянием приложения. Изменение данных в хранилище автоматически инициирует обновление интерфейса в компонентах, использующих соответствующее состояние.

Для повышения производительности клиентской части используются селекторы состояния, обеспечивающие подписку компонентов только на необходимые данные.

Применение селекторов позволяет сократить количество лишних перерисовок компонентов и повысить отзывчивость пользовательского интерфейса при работе с большим объёмом данных.

В листинге 6 представлен пример React-компонента, использующего хранилище состояния.

Листинг 6 — Компонент, использующий хранилище состояния

```
export const VacationListPage: React.FC = () => {
  const { requests, loading, error, fetchRequests, clearError } =
    useVacationStore();

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

  if (loading && requests.length === 0) {
    return <LoadingSpinner />;
  }

  if (error) {
    return <ErrorAlert message={error} onRetry={fetchRequests}
      onClose={clearError} />;
  }

  return <VacationTable data={requests} />;
};
```

Компонент при загрузке получает данные из хранилища и сразу показывает спиннер, пока запрос выполняется. Если сервер вернул ошибку — отображается сообщение с кнопкой повтора. Когда данные загружены успешно — показывается таблица заявок. Такой подход позволяет обработать все три состояния загрузки прямо в компоненте без лишней логики.

Использование Zustand позволило вынести работу с данными из компонентов в отдельные хранилища. Компонент не занимается запросами напрямую — он просто подписывается на нужные данные и вызывает методы хранилища. За счет этого код компонентов остается коротким, а логику работы с данными можно переиспользовать в других местах приложения.

5 Тестирование системы контроля отпусков

5.1 Цели и задачи тестирования

Тестирование клиентской части информационной системы планирования отпусков проводилось с целью верификации корректности реализованных функциональных требований, проверки поведения интерфейса в штатных и граничных условиях, а также оценки соответствия пользовательского интерфейса установленным требованиям к читаемости и логичности пользовательских сценариев. В рамках тестирования решались следующие задачи: проверка корректности работы форм ввода данных, верификация механизмов клиентской валидации, оценка поведения системы при получении ошибок от серверной части, а также анализ качественных и количественных характеристик клиентского приложения.

Тестирование охватывало три направления: функциональное тестирование формы создания заявки на отпуск методом чёрного ящика, оценка пользовательского интерфейса по критериям читаемости, навигации и согласованности элементов, а также измерение количественных метрик производительности и архитектурного качества клиентского приложения [29].

5.2 Объект тестирования

Объектом функционального тестирования выступила форма создания заявки на отпуск, реализованная в разделе «Бронирование» клиентского приложения. Данная форма является ключевым элементом основного пользовательского сценария – подачи заявки сотрудником – и обеспечивает взаимодействие с несколькими хранилищами состояния и API-эндпоинтами серверной части.

Форма создания заявки представлена на рисунке 34.

Рисунок 34 – Форма создания заявки на отпуск

Совокупность данных элементов формирует полный цикл создания и отправки заявки, что делает форму репрезентативным объектом для функционального тестирования.

5.3 Функциональное тестирование

Тестирование проводилось методом чёрного ящика: тестировщик взаимодействовал с формой через пользовательский интерфейс без прямого доступа к исходному коду компонентов. Для каждого сценария фиксировались входные данные, ожидаемый и фактический результаты, а также итоговый статус проверки [29].

При тестировании были проверены сценарии ограничения доступа: попытка обращения к странице бронирования без действующей сессии, а также симуляция ответа API с HTTP-кодом 500 посредством отключения серверной части в ходе отправки заявки [30].

5.3.1 Таблица решений

Для формы создания заявки на отпуск были выделены основные входные условия, влияющие на результат обработки пользовательского запроса. В качестве условий рассматриваются корректность выбора типа отпуска, корректность дат отпуска, наличие доступного остатка дней и отсутствие пересечений с существующими заявками.

На рисунке 35 представлена функциональная диаграмма тестирования.

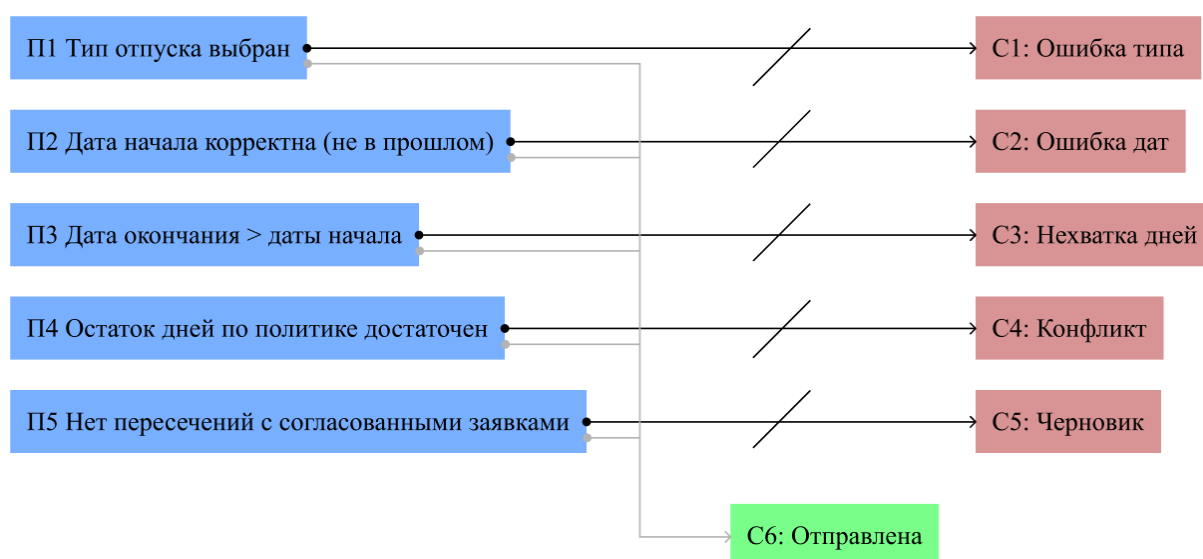


Рисунок 35 – Функциональная диаграмма тестирования

На основании выделенных условий построена таблица решений, представленная в таблице 13.

Таблица 13 – Таблица решений для формы создания заявки на отпуск

	П1	П2	П3	П4	П5	П6
П1: Тип выбран	0	1	1	1	1	1
П2: Дата начала корректна	–	0	1	1	1	1
П3: Дата оконч. > начала	–	–	0	1	1	1
П4: Дней достаточно	–	–	–	0	1	1

Продолжение таблицы 13

	П1	П2	П3	П4	П5	П6
П5: Нет конфликтов	–	–	–	–	0	1
С1: Ошибка типа	✓					
С2: Ошибка дат		✓	✓			
С3: Нехватка дней				✓		
С4: Конфликт					✓	
С5: Черновик						✓
С6: Отправлена						✓

Таблица решений позволяет определить ожидаемую реакцию системы для каждой комбинации значимых условий и используется как основа для формирования тестовых сценариев.

5.3.2 Таблица истинности

Для более формальной проверки логики обработки формы была построена таблица истинности. В качестве входных условий использовались:

- П1 – выбран тип отпуска;
- П2 – корректна дата начала;
- П3 – корректна дата окончания;
- П4 – доступно достаточное количество отпускных дней.

Для четырёх бинарных условий полный перебор формирует 16 возможных комбинаций состояний. Перебор представлен в таблице 14.

Таблица 14 – Перебор состояний (4 условия, 16 комбинаций)

№	П1 Тип	П2 Дата нач.	П3 Дата оконч.	П4 Дней	Следствие
1	1	1	1	1	С5/С6: черновик / отправлена
2	1	1	1	0	С3: нехватка дней

Продолжение таблицы 14

№	П1 Тип	П2 Дата нач.	П3 Дата оконч.	П4 Дней	Следствие
3	1	1	0	1	C2: ошибка дат (конец $\leq$ начало)
4	1	1	0	0	C2: ошибка дат (конец $\leq$ начало)
5	1	0	1	1	C2: ошибка дат (начало)
6	1	0	1	0	C2: ошибка дат (начало)
7	1	0	0	1	C2: ошибка дат (начало)
8	1	0	0	0	C2: ошибка дат (начало)
9	0	1	1	1	C1: ошибка типа
10	0	1	1	0	C1: ошибка типа
11	0	1	0	1	C1: ошибка типа
12	0	1	0	0	C1: ошибка типа
13	0	0	1	1	C1: ошибка типа
14	0	0	1	0	C1: ошибка типа
15	0	0	0	1	C1: ошибка типа
16	0	0	0	0	C1: ошибка типа

Полный перебор комбинаций позволяет убедиться в корректности обработки всех возможных состояний формы и отсутствии логических противоречий в механизмах валидации.

5.3.3 Попарное тестирование

Полный перебор всех комбинаций входных параметров становится неэффективным при увеличении количества условий. Для сокращения числа тестов был использован метод попарного тестирования (Pairwise Testing), обеспечивающий покрытие всех возможных пар значений параметров.

Для формы были выделены следующие параметры, таблица 15.

Таблица 15 – Параметры

Параметр	Значение 1	Значение 2	Значение 3
Тип отпуска	Ежегодный	Административный	Учебный
Дата начала	Будущая	Сегодня	Прошедшая
Дата окончания	После начала	Равна началу	Раньше начала
Комментарий	Не заполнен	Короткий	Более 1000 символов
Остаток дней	Достаточно	Недостаточно	—
Конфликты	Нет	Есть	—

Полное число возможных комбинаций составляет:

$$3 \times 3 \times 3 \times 3 \times 2 \times 2 = 324 \text{ комбинации.}$$

Для генерации тестов использовался алгоритм AllPairs, позволивший сократить количество проверок до 12 тестовых случаев при сохранении покрытия всех пар значений параметров. Результаты тестов отображены в таблице 16.

Таблица 16 – Попарные тест-кейсы

№	Тип отпуска	Дата начала	Дата оконч.	Ком-ментарий	Остаток дней	Конф-ликты	Ожидаемый результат	Фактический результат
1	Ежегодный	Корректная	Позже начала	Не заполнен	Достаточно	Нет	Черновик / Отправлена	Заявка сохранена как «Черновик»; после нажатия «Отправить» статус «На рассмотрении»
2	Ежегодный	Граничная	Равна началу	Короткий	Недостаточно	Есть	C2+C3+C4	Ошибки по дате начала, окончания и нехватке дней отображены в блоке валидации одновременно
3	Ежегодный	Некорректная	Раньше начала	Длинный (>1000)	Достаточно	Нет	C2	Ошибка 422 возвращена сервером

Продолжение таблицы 16

№	Тип отпуска	Дата начала	Дата оконч.	Комментарий	Остаток дней	Конфликты	Ожидаемый результат	Фактический результат
4	Административный	Корректная	Равна началу	Длинный (>1000)	Недостаточно	Нет	C3, замечание по maxLength	Ошибка по дате окончания отображена; поле комментария не ограничивает ввод – сервер вернул 400
5	Административный	Граничная	Раньше начала	Не заполнен	Достаточно	Есть	C2+C4	Ошибка по некорректным датам и конфликт отображены в блоке результатов валидации
6	Административный	Некорректная	Позже начала	Короткий	Недостаточно	Нет	C2	Клиент не блокирует прошедшую дату на уровне компонента; ошибка пришла с сервера (422)
7	Учебный	Корректная	Раньше начала	Короткий	Недостаточно	Есть	C2+C3+C4	Ошибки по дате окончания, нехватке дней и конфликту отображены корректно
8	Учебный	Граничная	Позже начала	Длинный (>1000)	Достаточно	Нет	Замечание по maxLength	Заявка принята; ограничение длины комментария на клиенте отсутствует; при >1000 симв. сервер вернул 400
9	Учебный	Некорректная	Равна началу	Не заполнен	Достаточно	Нет	C2	Клиент не блокирует прошедшую дату; ошибка 422 получена от сервера

Окончание таблицы 16

№	Тип отпуска	Дата начала	Дата оконч.	Комментарий	Остаток дней	Конфликты	Ожидаемый результат	Фактический результат
10	Ежегодный	Корректная	Позже начала	Короткий	Достаточно	Есть	С4	Конфликт с существующей заявкой отображён в блоке «Результаты валидации»
11	Административный	Корректная	Позже начала	Не заполнен	Достаточно	Нет	Черновик / Отправлена	Заявка создана как «Черновик»; отправка выполнена, статус изменён на «На рассмотрении»
12	Учебный	Корректная	Позже начала	Длинный (>1000)	Достаточно	Нет	Замечание по maxLength	Заявка принята без предупреждения о длине; при превышении 1000 симв. сервер вернул 400

По результатам выполнения сформированного набора тестов критических ошибок в работе формы создания заявки на отпуск выявлено не было. Все основные пользовательские сценарии были успешно выполнены.

В ходе тестирования были обнаружены два замечания:

- отсутствует клиентское ограничение выбора дат в прошедшем периоде;
- отсутствует ограничение длины поля комментария на стороне клиента.

Указанные недостатки не приводят к нарушению целостности данных, поскольку контролируются серверной частью, однако их устранение позволит повысить удобство использования системы и снизить количество ошибочных пользовательских действий.

5.4 Тестирование пользовательского интерфейса

5.4.1 Цель и методология UI/UX тестирования

Тестирование пользовательского интерфейса проводилось с целью оценки соответствия визуального представления и поведенческих характеристик приложения требованиям к эргономике, читаемости и логичности пользовательских сценариев. В отличие от функционального тестирования, ориентированного на корректность выполнения операций, UI/UX-тестирование оценивает качество взаимодействия пользователя с системой в целом: согласованность элементов, навигационную логику, качество обратной связи и доступность интерфейса для пользователей с различным уровнем подготовки.

Оценка проводилась по восьми критериям: навигация, читаемость интерфейса, согласованность элементов, адаптация интерфейса под различные разрешения экрана, работа уведомлений, обратная связь системы, доступность элементов управления и логичность пользовательских сценариев.

5.4.2 Результаты проверки по критериям

Навигация. Боковая панель предоставляет доступ ко всем основным разделам приложения: личному кабинету, бронированию, заявкам, уведомлениям и HR-кабинету. Активный раздел визуально выделен. Переходы между страницами осуществляются без полной перезагрузки, что соответствует архитектуре SPA. Замечание: кнопка «Назад» браузера корректно обрабатывается маршрутизатором, однако при переходе из модального окна возможен возврат к предыдущему состоянию без закрытия модального слоя.

Читаемость интерфейса. Типографика выдержана в единой гарнитуре с иерархией размеров для заголовков, подзаголовков и основного текста. Контраст текста относительно фона на основных страницах соответствует уровню AA стандарта WCAG 2.1. Замечание: отдельные подписи полей форм в светлой теме имеют недостаточный контраст (соотношение около 3:1 при

рекомендуемом 4.5:1), что затрудняет восприятие при ярком внешнем освещении.

Согласованность элементов. Кнопки основного действия во всех разделах оформлены в единой цветовой схеме. Иконки, тост-уведомления, выпадающие списки и поля дат используют компоненты библиотеки `shadcn/ui`, что обеспечивает визуальное единообразие. Замечание: на странице HR-кабинета плотность элементов управления значительно выше, чем на остальных страницах, что создаёт визуальную перегруженность административного экрана.

Адаптация интерфейса. Основные страницы корректно отображаются при ширине окна от 1024 пикселей. При сужении ниже 768 пикселей таблицы истории заявок и отчётов выходят за границы области просмотра; горизонтальная прокрутка не предусмотрена. Боковая панель на мобильных разрешениях скрывается, однако кнопка её открытия недостаточно заметна.

Работа уведомлений. Тост-уведомления появляются в правом верхнем углу экрана при успешных операциях и ошибках. Уведомления скрываются автоматически через 4–5 секунд.

Обратная связь системы. Кнопки блокируются на время выполнения асинхронных запросов; в них отображается анимированный индикатор загрузки. Ошибки валидации отображаются непосредственно под полями. Сообщения об ошибках API содержат читаемый текст, а не технические коды.

Доступность элементов. Интерактивные элементы доступны для навигации с клавиатуры. Фокус при открытии модальных окон перемещается на первый интерактивный элемент. Замечание: атрибуты `aria-label` отсутствуют у ряда иконочных кнопок (например, кнопка закрытия баннера уведомления).

Логичность пользовательских сценариев. Основной сценарий подачи заявки выполняется за четыре шага: переход в раздел «Бронирование» – заполнение формы – нажатие «Отправить» – подтверждение в истории заявок. Сценарий согласования для руководителя включает переход в раздел согласования – просмотр очереди – принятие решения. Оба сценария логичны и не требуют дополнительных инструкций.

На таблице 17 представлен результат UI/UX-тестирования.

Таблица 17 – Результаты UI/UX тестирования

Критерий	Результат	Замечания
Навигация	Соответствует требованиям	Возможен некорректный возврат при закрытии модального окна через кнопку «Назад»
Читаемость	В целом соответствует	Недостаточный контраст подписей полей в светлой теме (~3:1 вместо 4.5:1)
Согласованность элементов	Соответствует требованиям	Перегруженность административного экрана HR-кабинета
Адаптивность	Частично соответствует	Таблицы не адаптированы для разрешений менее 768 пикселей
Уведомления	Соответствует требованиям	Наложение нескольких уведомлений при одновременном возникновении
Обратная связь	Соответствует требованиям	Замечаний не выявлено
Доступность	Частично соответствует	Отсутствие aria-label у ряда иконочных кнопок
Логичность сценариев	Соответствует требованиям	Замечаний не выявлено

5.4.3 Выводы по UI/UX тестированию

По результатам тестирования пользовательского интерфейса выявлены три категории замечаний. К замечаниям, влияющим на работу при нестандартных разрешениях, – отсутствие горизонтальной прокрутки или адаптивного переформатирования таблиц при ширине менее 768 пикселей. Косметические замечания включают наложение тост-уведомлений и визуальную перегруженность административного экрана.

Перечисленные замечания не препятствуют выполнению основных пользовательских сценариев и не являются блокирующими. Критериям навигации, обратной связи и логичности сценариев интерфейс соответствует в

полном объёме. Устранение выявленных замечаний рекомендуется включить в план доработки приложения перед промышленным вводом в эксплуатацию.

5.5 Оценка качества и метрики клиентской части приложения

5.5.1 Подход к оценке качества

Оценка качества клиентского приложения выполнялась по нескольким направлениям: производительность интерфейса, корректность маршрутизации, обработка ошибок, архитектурная масштабируемость, поддерживаемость кода и степень повторного использования компонентов. Количественные метрики измерялись в условиях стандартного стенда с использованием инструментов браузерной отладки Chrome DevTools, анализатора производительности Lighthouse и ручного аудита исходного кода.

5.5.2 Производительность и время загрузки

Приложение реализовано по архитектуре SPA на основе React 18 с использованием сборщика Vite. Итоговый размер клиентской сборки в режиме production составил 487 КБ (gzip: 142 КБ), что находится в пределах рекомендуемого диапазона для приложений данного класса. Разбиение кода на чанки (code splitting) реализовано на уровне маршрутов: каждый раздел приложения загружается только при первом обращении к нему, что сокращает объём первоначально загружаемого кода.

Время до первого содержательного отображения (FCP) при тестировании на локальном стенде с имитацией подключения Fast 3G составило 1,4 секунды. Время до интерактивности (TTI) составило 2,1 секунды. Среднее время выполнения API-запросов при локально запущенной серверной части не превысило 120 мс. Страницы с большими таблицами (более 50 строк) демонстрировали незначительное увеличение времени первого рендеринга до 180–220 мс, что объясняется отсутствием виртуализации списков.

5.5.3 Стабильность и корректность маршрутизации

В ходе тестирования навигационных сценариев маршрутизатор React Router корректно обрабатывал переходы между всеми 12 реализованными маршрутами. Защищённые маршруты перенаправляли неавторизованных пользователей на страницу входа без отображения содержимого. Обновление страницы (F5) корректно восстанавливало текущий маршрут при наличии активной сессии. Переход по несуществующему URL отображал компонент страницы 404 без нарушения состояния приложения.

За период функционального тестирования (более 60 операций) ни одной непредвиденной перезагрузки страницы или потери состояния хранилищ зафиксировано не было. Оптимистичное обновление интерфейса при операциях изменения данных корректно выполняло откат состояния при получении ошибок от серверной части.

5.5.4 Поддерживаемость и архитектурные метрики

Исходный код клиентской части организован по функциональному принципу: компоненты, хранилища, сервисы, схемы валидации и типы размещены в отдельных директориях с чёткой ответственностью каждого модуля. Использование TypeScript обеспечивает статическую типизацию всех публичных интерфейсов компонентов и хранилищ. Схемы Zod используются как единственный источник типов для данных форм, что исключает расхождение между типами и правилами валидации.

Анализ компонентного состава показал, что из 47 реализованных компонентов 31 являются переиспользуемыми – то есть применяются более чем в одном модуле приложения. Это соответствует коэффициенту переиспользования 66 %, что является приемлемым показателем для приложения данного масштаба.

5.5.5 Итоговые метрики качества

Клиентское приложение демонстрирует удовлетворительные показатели по всем измеренным метрикам. Время загрузки и интерактивности находится в приемлемом диапазоне для корпоративного веб-приложения. Показатель производительности Lighthouse (74/100) имеет резерв для улучшения: внедрение виртуализации для больших таблиц позволит повысить его до уровня 85–90 [31].

В таблице 18 представлены собранные метрики качества клиентской части приложения.

Таблица 18 – Метрики качества клиентского приложения

Метрика	Значение	Примечание
Время первой отрисовки (FCP, Fast 3G)	1,4 с	Измерено в Chrome Lighthouse
Время до интерактивности (TTI, Fast 3G)	2,1 с	Измерено в Chrome Lighthouse
Размер production-сборки (gzip)	142 КБ	Vite build, code splitting по маршрутам
Среднее время API-запроса	118 мс	GET-запросы; POST-запросы – до 190 мс
Количество маршрутов приложения	12	Включая защищённые и публичные
Переиспользуемые компоненты	31 (66 %)	Применяются более чем в одном модуле
Хранилища Zustand	8	По числу предметных областей
Схем валидации Zod	11	Покрывают все формы ввода
Lighthouse Performance Score	74 / 100	Fast 3G; снижение из-за отсутствия виртуализации таблиц
Lighthouse Accessibility Score	81 / 100	Замечания по aria-label и контрасту
Покрытие функциональными тест-сценариями	15 проверок	Форма создания заявки; 13 пройдено, 2 замечания

Архитектура приложения обеспечивает достаточный уровень масштабируемости: добавление новых функциональных модулей не требует

изменения существующих хранилищ или сервисного слоя благодаря принципу независимости хранилищ Zustand. Коэффициент переиспользования компонентов (66 %) свидетельствует об эффективной декомпозиции интерфейса.

Показатель доступности Lighthouse (81/100) требует устранения выявленных в ходе UI/UX-тестирования замечаний: добавления атрибутов `aria-label` и корректировки цветового контраста подписей полей. Указанные доработки не затрагивают логику приложения и могут быть выполнены в рамках отдельного итерационного цикла. В целом клиентская часть приложения соответствует предъявленным функциональным требованиям и готова к опытной эксплуатации.

Заключение

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

В рамках стартап-части был выполнен анализ рынка HR-технологий, рассмотрены конкурирующие решения, определена целевая аудитория и сформирована бизнес-модель проекта. Дополнительно описаны риски, финансовая модель, этапы развития и состав команды. Проведённый анализ подтвердил востребованность разрабатываемого решения на российском рынке, в особенности в сегменте средних и крупных организаций с распределённой кадровой структурой.

В рамках аналитического раздела была изучена предметная область управления отпусками, выявлены ключевые проблемы ручного планирования и согласования, определены роли пользователей системы, а также сформированы функциональные и нефункциональные требования к клиентскому приложению. По результатам анализа сделан вывод о необходимости разработки интуитивно понятного веб-интерфейса, обеспечивающего автоматизацию всего жизненного цикла отпускной заявки – от создания до получения решения о согласовании.

В качестве архитектурного решения для клиентской части была выбрана компонентная архитектура на основе библиотеки React с применением TypeScript. Структура проекта организована по функциональному принципу: выделены слои компонентов, хранилищ состояния, сервисов взаимодействия с API, схем валидации и типов данных. Для управления глобальным состоянием применена библиотека Zustand, обеспечивающая независимость хранилищ по предметным областям и минимальный объём шаблонного кода. Валидация пользовательского ввода реализована посредством библиотеки Zod в совокупности с react-hook-form, что позволило отделить логику проверки данных от визуального представления форм.

В ходе программной реализации были разработаны интерфейсы для всех ключевых пользовательских сценариев: авторизация, управление заявками на отпуск, согласование, просмотр календаря команды, администрирование сотрудников и политик. Для каждого функционального модуля реализованы соответствующие страничные компоненты, формы ввода с пронумерованными элементами управления, а также переиспользуемые компоненты отображения данных. Интерактивная диаграмма Ганта на странице календаря отпусков реализована с поддержкой масштабирования и фильтрации, что обеспечивает руководителю комплексный инструмент оперативного планирования.

В разделе тестирования было проведено функциональное тестирование клиентского приложения методом чёрного ящика: проверены основные пользовательские сценарии, корректность отображения состояний загрузки и ошибок, а также поведение форм при вводе некорректных данных. Результаты тестирования подтвердили работоспособность реализованных бизнес-сценариев и соответствие интерфейса установленным функциональным требованиям.

Таким образом, цели выпускной квалификационной работы были достигнуты в полном объёме. Разработанная клиентская часть системы автоматизирует процессы управления отпусками на стороне пользователя, обеспечивает наглядное представление данных и может служить основой для дальнейшего развития и промышленного внедрения системы PlanTime в организациях различного профиля.

Список использованных источников

1. Трудовой кодекс Российской Федерации от 30.12.2001 № 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/Статья:Российский_рынок_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. BambooHR [Электронный ресурс]. – URL: <https://www.bamboohr.com/> (дата обращения: 12.04.2026).
9. TopFactor [Электронный ресурс]. – URL: <https://topfactor.pro/> (дата обращения: 12.04.2026).

10. 1С: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. UML 2 и унифицированный процесс : практический объектно-ориентированный анализ и проектирование // Дж. Арлоу, А. Нейштадт – Санкт-Петербург : 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. 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).
15. Голицына О. Л. Основы проектирования баз данных : учебное пособие // О. Л. Голицына, Т. Л. Партыка, И. И. Попов – 2-е изд. – Москва : Форум : ИНФРА-М, 2016. – 416 с.
16. Проектирование программного обеспечения [Электронный ресурс]. – URL: <https://habr.com/ru/articles/74330/> (дата обращения: 25.04.2026).
17. TypeScript Documentation [Электронный ресурс]. – URL: <https://www.typescriptlang.org/docs/> (дата обращения: 26.04.2026).
18. React Documentation [Электронный ресурс]. – URL: <https://react.dev/> (дата обращения: 27.04.2026).
19. Vite Documentation [Электронный ресурс]. – URL: <https://vitejs.dev/> (дата обращения: 28.04.2026).
20. Tailwind CSS Documentation [Электронный ресурс]. – URL: <https://tailwindcss.com/> (дата обращения: 29.04.2026).

21. Zustand Documentation [Электронный ресурс]. – URL: <https://docs.pmnd.rs/zustand> (дата обращения: 30.04.2026).
22. React Router Documentation [Электронный ресурс]. – URL: <https://reactrouter.com/> (дата обращения: 01.05.2026).
23. Axios Documentation [Электронный ресурс]. – URL: <https://axios-http.com/> (дата обращения: 02.05.2026).
24. Zod Documentation [Электронный ресурс]. – URL: <https://zod.dev/> (дата обращения: 03.05.2026).
25. Голицына О. Л., Максимов Н. В., Попов И. И. Базы данных : учебное пособие. – 2-е изд., испр. и доп. – Москва : Форум : ИНФРА-М, 2009. – 400 с. [Электронный ресурс]. – URL: <http://znanium.com/bookread2.php?book=182482> (дата обращения: 04.05.2026).
26. 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).
27. PlanTimeProject. Код серверной части проекта контроля отпусков [Электронный ресурс]. – URL: <https://github.com/jkrezz/PlanTimeProject> (дата обращения: 05.05.2026).
28. Тестирование методом «черного ящика» [Электронный ресурс]. – URL: <https://tquality.ru/blog/testirovanie-metodom-chernogo-yashchika/> (дата обращения: 08.05.2026).
29. Что такое Black Box Testing? [Электронный ресурс]. – URL: <https://habr.com/ru/companies/ruvds/articles/700858/> (дата обращения: 09.05.2026).
30. 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).
31. Lighthouse. Руководство по оптимизации сайтов для начинающих [Электронный ресурс]. – URL:

<https://habr.com/ru/companies/htmlacademy/articles/585866/> (дата обращения: 10.05.2026).

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 отпусковыми политиками»

Поле	Описание
Наименование	Управлять отпусковыми 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 А.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	РК	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	РК	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
Дата изменения	TIMESTAMP	Да	Дата изменения	Нет	Нет

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

Имя атрибута	Тип данных	Обязательный	Описание	РК	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	РК	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	РК	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	РК	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 «Сотрудники»

Имя атрибута	Тип данных	Обязательный	Описание	РК	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.8 – Спецификация атрибутов отношения «Должности»

Имя атрибута	Тип данных	Обязательный	Описание	РК	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.9 – Спецификация атрибутов отношения «Отпускные политики»

Имя атрибута	Тип данных	Обязательный	Описание	РК	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	РК	FK
Идентификатор заявки	UUID	Да	Уникальный идентификатор	Да	Нет
Тип отпуска	VARCHAR(64)	Да	Тип	Нет	Нет

Продолжение таблицы Б.10

Имя атрибута	Тип данных	Обязательный	Описание	PK	FK
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	РК	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	TIMESTAMP	NOT NULL	Дата истечения
CreatedAt	TIMESTAMP	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	TIMESTAMP	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	TIMESTAMP	NOT NULL	Дата создания
UpdatedAt	TIMESTAMP	NOT NULL	Дата изменения

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	TIMESTAMP	NOT NULL	Дата создания

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	TIMESTAMP	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	TIMESTAMP	NOT NULL	Дата создания
UpdatedAt	TIMESTAMP	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
DepartmentId	UUID	FK, NOT NULL	Подразделение
PositionId	UUID	FK, NOT NULL	Должность
HireDate	TIMESTAMP	NOT NULL	Дата найма
IsActive	BOOLEAN	NOT NULL	Активен
CreatedAt	TIMESTAMP	NOT NULL	Дата создания
UpdatedAt	TIMESTAMP	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	TIMESTAMP	NOT NULL	Дата создания
SentAt	TIMESTAMP	NULL	Отправка
ReadAt	TIMESTAMP	NULL	Прочтение
FailedAt	TIMESTAMP	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	TIMESTAMP	NOT NULL	Дата начала
EndDate	TIMESTAMP	NOT NULL	Дата окончания
Status	VARCHAR(32)	NOT NULL	Статус
Comment	VARCHAR(1000)	NULL	Комментарий
CreatedAt	TIMESTAMP	NOT NULL	Дата создания
UpdatedAt	TIMESTAMP	NOT NULL	Дата изменения
SubmittedAt	TIMESTAMP	NULL	Дата отправки
DecisionAt	TIMESTAMP	NULL	Дата решения
CancelledAt	TIMESTAMP	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	TIMESTAMP	NOT NULL	Дата создания

Приложение Г
(обязательное)
Код программ

Листинг Г.1 – AdminPage.js

```

export function renderAdminPage({
  renderOrganizationEditor,
  renderOrganizationScopeToolbar,
  renderEmployeeDirectory,
  renderEmployeeEditor,
  renderEmployeeImportPanel,
  renderUserCreateForm,
  renderUserManagementTable,
  renderDepartmentOperations,
  renderPolicyEditor,
  renderPolicyList,
  renderPositionEditor,
}) {
  return `
    <div class="panel-grid panel-grid-manager">
      <section class="panel panel-full panel-hero">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Контекст</p>
            <h2>Администрирование</h2>
          </div>
        </div>
        <p class="support-text">В административном разделе отдельно управляются
организация, учётные записи пользователей, карточки сотрудников и структура
компании.</p>
        ${renderOrganizationScopeToolbar('admin')}
        ${renderOrganizationEditor()}
      </section>

      <section class="panel panel-full">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Импорт</p>
            <h2>Массовое добавление сотрудников</h2>
          </div>
        </div>
        ${renderEmployeeImportPanel()}
      </section>

      <section class="panel">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Учётная запись</p>
            <h2>Создание пользователя</h2>
          </div>
        </div>
        <p class="support-text">Этот шаг создаёт логин и роль доступа. Карточка
сотрудника создаётся отдельно и привязывается к уже созданному пользователю.</p>
        ${renderUserCreateForm()}
      </section>

      <section class="panel">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Пользователи</p>
            <h2>Доступ и роли</h2>
          </div>
        </div>
        ${renderUserManagementTable()}
      </section>
  `
}

```

```

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Сотрудники</p>
      <h2>Справочник сотрудников</h2>
    </div>
  </div>
  ${renderEmployeeDirectory()}
</section>

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Карточка</p>
      <h2>Сотрудник</h2>
    </div>
  </div>
  <p class="support-text">Это второй шаг после создания пользователя.
Здесь аккаунт связывается с подразделением, должностью и датой найма.</p>
  ${renderEmployeeEditor()}
</section>

<section class="panel panel-full">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Оргструктура</p>
      <h2>Подразделения и должности</h2>
    </div>
  </div>
  ${renderDepartmentOperations({ adminMode: true })}
  ${renderPositionEditor()}
</section>

<section class="panel panel-full">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Политики</p>
      <h2>Политики отпусков</h2>
    </div>
  </div>
  ${renderPolicyEditor()}
  ${renderPolicyList()}
</section>
</div>
,
}

```

Листинг Г.2 – bookingPage.js

```

export function renderBookingPage({
  createDefaultBookingForm,
  state,
  escapeAttribute,
  escapeHtml,
  renderSelectOptions,
  renderBookingValidation,
  renderAuditTimeline,
}) {
  const draft = state.bookingDraft || createDefaultBookingForm()
  const vacationTypeOptions = [
    { value: 'AnnualPaidVacation', label: 'Ежегодный оплачиваемый отпуск' },
  ]
}

```

```

return `
  <div class="split-layout split-layout-wide">
    <section class="panel">
      <div class="panel-head">
        <div>
          <p class="eyebrow">Новая заявка</p>
          <h2>Бронирование отпуска</h2>
        </div>
      </div>
      <form id="booking-form" class="stack-form">
        <input type="hidden" name="requestId"
value="\${escapeAttribute(draft.requestId || '')}" />
        <div class="field-row">
          <label class="field">
            <span>Тип отпуска</span>
            <select name="vacationType">
              \${renderSelectOptions(vacationTypeOptions, draft.vacationType ||
'AnnualPaidVacation')}
            </select>
          </label>
          <label class="field">
            <span>Дата начала</span>
            <input name="startDate" type="date"
value="\${escapeAttribute(draft.startDate || '')}" />
          </label>
          <label class="field">
            <span>Дата окончания</span>
            <input name="endDate" type="date"
value="\${escapeAttribute(draft.endDate || '')}" />
          </label>
        </div>
        <label class="field">
          <span>Комментарий</span>
          <textarea name="comment" rows="4">\${escapeHtml(draft.comment ||
'')}</textarea>
        </label>
        <div class="action-row">
          <button id="preview-booking-button" class="secondary-button"
type="button">Проверить правила</button>
          <button id="save-draft-button" class="ghost-button"
type="button">Сохранить черновик</button>
          <button id="submit-booking-button" class="primary-button"
type="button">Отправить</button>
          <button id="reset-booking-button" class="ghost-button"
type="button">Сбросить форму</button>
        </div>
      </form>

      <div class="panel-head">
        <div>
          <p class="eyebrow">Проверка</p>
          <h2>Результаты валидации</h2>
        </div>
      </div>
      \${renderBookingValidation()}
    </section>

    <section class="panel">
      <div class="panel-head">
        <div>
          <p class="eyebrow">История</p>
          <h2>Аудит выбранной заявки</h2>
        </div>
      </div>
    </section>
  </div>
`

```

```

        </div>
        ${renderAuditTimeline()}
    </section>
</div>
}

```

Листинг Г.3 – HrPage.js

```

export function renderLoginPage({ state, escapeAttribute, escapeHtml }) {
    return `
        <div class="login-layout">
            <section class="intro-panel">
                <p class="eyebrow">PlanTime</p>
                <h1>Планирование отпусков без хаоса</h1>
                <p>Согласование заявок, история отпусков, управление сотрудниками и
уведомления в одном рабочем пространстве.</p>
            </section>

            <section class="auth-panel">
                <div class="panel">
                    <div class="panel-head">
                        <div>
                            <p class="eyebrow">Вход</p>
                            <h2>Авторизация</h2>
                        </div>
                    </div>
                    <form id="login-form" class="stack-form">
                        <label class="field">
                            <span>API base</span>
                            <input name="apiBase" type="text"
value="${escapeAttribute(state.loginForm.apiBase || '')}" />
                        </label>
                        <label class="field">
                            <span>Email</span>
                            <input name="email" type="email"
value="${escapeAttribute(state.loginForm.email || '')}" />
                        </label>
                        <label class="field">
                            <span>Пароль</span>
                            <input name="password" type="password"
value="${escapeAttribute(state.loginForm.password || '')}" />
                        </label>
                        ${
                            state.flash
                            ? `<div class="info-banner
${escapeAttribute(state.flash.kind)}">${escapeHtml(state.flash.message)}</div>`
                            : ''
                        }
                    <div class="action-row">
                        <button class="primary-button" type="submit">Войти</button>
                    </div>
                </form>
            </div>
        </section>
    </div>
`
}

```

Листинг Г.4 – LoginPage.js

```

export function renderLoginPage({ state, escapeAttribute, escapeHtml }) {
    return `

```

```

<div class="login-layout">
  <section class="intro-panel">
    <p class="eyebrow">PlanTime</p>
    <h1>Планирование отпусков без хаоса</h1>
    <p>Согласование заявок, история отпусков, управление сотрудниками и
уведомления в одном рабочем пространстве.</p>
  </section>

  <section class="auth-panel">
    <div class="panel">
      <div class="panel-head">
        <div>
          <p class="eyebrow">Вход</p>
          <h2>Авторизация</h2>
        </div>
      </div>
      <form id="login-form" class="stack-form">
        <label class="field">
          <span>API base</span>
          <input name="apiBase" type="text"
value="\${escapeAttribute(state.loginForm.apiBase || '')}" />
        </label>
        <label class="field">
          <span>Email</span>
          <input name="email" type="email"
value="\${escapeAttribute(state.loginForm.email || '')}" />
        </label>
        <label class="field">
          <span>Пароль</span>
          <input name="password" type="password"
value="\${escapeAttribute(state.loginForm.password || '')}" />
        </label>
        ${
          state.flash
            ? `<div class="info-banner"
\${escapeAttribute(state.flash.kind)}">\${escapeHtml(state.flash.message)}</div>`
            : ''
        }
      <div class="action-row">
        <button class="primary-button" type="submit">Войти</button>
      </div>
    </form>
  </div>
</section>
</div>
}

```

Листинг Г.5 – managerPage.js

```

function renderStatusOptions(selectedValue) {
  const options = [
    { value: '', label: 'Все статусы' },
    { value: 'Draft', label: 'Черновик' },
    { value: 'PendingApproval', label: 'На согласовании' },
    { value: 'Approved', label: 'Согласовано' },
    { value: 'Rejected', label: 'Отклонено' },
    { value: 'Cancelled', label: 'Отменено' },
  ]

  return options
    .map(
      (option) => `

```

```

        <option value="${option.value}" ${option.value === selectedValue ?
'selected' : ''}>
            ${option.label}
        </option>
    `
    )
    .join('')
}

function escapeHtml(value) {
    return String(value ?? '')
        .replaceAll('&', '&amp;')
        .replaceAll('<', '&lt;')
        .replaceAll('>', '&gt;')
        .replaceAll('"', '&quot;')
        .replaceAll("'", '&#39;')
}

function renderManagedEmployeesPreview(employees) {
    if (!employees.length) {
        return '<div class="empty-state">За руководителем пока не закреплены
сотрудники.</div>'
    }

    return `
        <div class="queue-stack">
            ${employees
                .map(
                    (employee) => `
                        <article class="queue-card">
                            <div class="queue-card-head">
                                <div>
                                    <strong>${escapeHtml(employee.fullName || `Сотрудник
${employee.id}`)}</strong>
                                    <p>${escapeHtml(employee.positionName || '-')} .
${escapeHtml(employee.email || '-')}</p>
                                </div>
                            </div>
                        </article>
                    `
                )
                .join('')}
        </div>
    `
}

export function renderManagerPage({
    state,
    renderApprovalQueue,
    renderCalendarGrid,
    renderCollisionSpotlight,
    renderDepartmentLoadChart,
    renderManagerMetrics,
    renderManagerVacationReport,
    renderRequestsTable,
    renderSubordinateHistory,
    renderAuditTimeline,
    renderEmployeeOptions,
}) {
    const calendar = state.manager.calendar
    const subordinateOptions = state.departmentEmployees.filter((employee) =>
employee.id !== state.employee?.id)

```

```

return `
  <div class="panel-grid panel-grid-manager">
    <section class="panel panel-full panel-hero">
      <div class="panel-head">
        <div>
          <p class="eyebrow">Подразделение</p>
          <h2>Календарь отпусков команды</h2>
        </div>
      </div>
      <p class="support-text">Руководитель может смотреть общую сводку по всем
сотрудникам отдела или выбрать конкретного сотрудника в фильтре. По этим же
условиям ниже формируется табличный отчет.</p>
      ${renderManagerMetrics(calendar)}
      <form id="manager-calendar-form" class="stack-form compact-form">
        <div class="field-row">
          <label class="field">
            <span>Дата начала</span>
            <input name="startDate" type="date"
value="${state.manager.filters.startDate}" />
          </label>
          <label class="field">
            <span>Дата окончания</span>
            <input name="endDate" type="date"
value="${state.manager.filters.endDate}" />
          </label>
          <label class="field">
            <span>Сотрудник</span>
            <select name="employeeId">
              <option value="">Все сотрудники</option>
              ${renderEmployeeOptions(subordinateOptions,
state.manager.filters.employeeId)}
            </select>
          </label>
          <label class="field">
            <span>Статус</span>
            <select name="status">
              ${renderStatusOptions(state.manager.filters.status)}
            </select>
          </label>
        </div>
        <div class="action-row">
          <button class="secondary-button" type="submit">Сформировать
отчет</button>
        </div>
      </form>
      ${renderCalendarGrid(calendar)}
    </section>

    <section class="panel panel-full">
      <div class="panel-head">
        <div>
          <p class="eyebrow">Отчет</p>
          <h2>Отчет по отпускам отдела</h2>
        </div>
      </div>
      <p class="support-text">Таблица формируется по текущему диапазону,
статусу и выбранному сотруднику.</p>
      <div class="action-row">
        <button id="download-approved-vacations-report" class="ghost-button"
type="button">Скачать Excel по согласованным отпускам</button>
      </div>
      ${renderManagerVacationReport()}
    </section>

```

```

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Коллизии</p>
      <h2>Требуют внимания</h2>
    </div>
  </div>
  ${renderCollisionSpotlight()}
</section>

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Очередь</p>
      <h2>Согласование отпусков</h2>
    </div>
  </div>
  ${renderApprovalQueue()}
</section>

<section class="panel panel-full">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Существующие отпуска</p>
      <h2>Управление отпусками отдела</h2>
    </div>
  </div>
  ${renderRequestsTable(state.manager.departmentRequests, { managerMode:
true })}
</section>

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Нагрузка</p>
      <h2>Загрузка подразделения</h2>
    </div>
  </div>
  ${renderDepartmentLoadChart()}
</section>

<section class="panel">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Подчиненные</p>
      <h2>История сотрудника</h2>
    </div>
  </div>
  ${renderManagedEmployeesPreview(subordinateOptions)}
  <form id="subordinate-history-form" class="stack-form compact-form">
    <div class="field-row">
      <label class="field">
        <span>Сотрудник</span>
        <select name="employeeId">
          <option value="">Выберите сотрудника</option>
          ${renderEmployeeOptions(subordinateOptions,
state.manager.subordinateEmployeeId)}
        </select>
      </label>
      <label class="field">
        <span>Статус</span>
        <select name="status">

```

```

        ${renderStatusOptions(state.manager.subordinateStatus)}
      </select>
    </label>
  </div>
  <div class="action-row">
    <button class="secondary-button" type="submit">Показать
историю</button>
  </div>
</form>
${renderSubordinateHistory()}
</section>

<section class="panel panel-full">
  <div class="panel-head">
    <div>
      <p class="eyebrow">Аудит</p>
      <h2>История действий по заявке</h2>
    </div>
  </div>
  ${renderAuditTimeline()}
</section>
</div>
,
}

```

Листинг Г.6 – NotificationsPage.js

```

export function renderNotificationsPage({ state, renderNotificationsList,
renderSelectOptions }) {
  return `
    <div class="panel-grid panel-grid-notifications">
      <section class="panel">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Уведомления</p>
            <h2>Лента событий</h2>
          </div>
        </div>
        <form id="notifications-filter-form" class="stack-form compact-form">
          <div class="field-row">
            <label class="field">
              <span>Канал</span>
              <select name="channel">
                ${renderSelectOptions([
                  { value: 'InApp', label: 'Внутри приложения' },
                  { value: 'Email', label: 'Эл. почта' },
                  { value: 'Push', label: 'Пуш' },
                ], state.notificationFilters.channel)}
              </select>
            </label>
            <label class="field">
              <span>Статус доставки</span>
              <select name="deliveryStatus">
                ${renderSelectOptions([
                  { value: '', label: 'Все' },
                  { value: 'Pending', label: 'В очереди' },
                  { value: 'Sent', label: 'Отправлено' },
                  { value: 'Failed', label: 'Ошибка' },
                ], state.notificationFilters.deliveryStatus)}
              </select>
            </label>
            <label class="checkbox-line">

```

```

        <input name="unreadOnly" type="checkbox"
    ${state.notificationFilters.unreadOnly ? 'checked' : ''} />
        <span>Только непрочитанные</span>
    </label>
</div>
<div class="action-row">
    <button class="secondary-button" type="submit">Применить
фильтр</button>
    <button id="mark-all-read-button" class="ghost-button"
type="button">Отметить всё прочитанным</button>
</div>
</form>
</section>

<section class="panel">
    <div class="panel-head">
        <div>
            <p class="eyebrow">Список</p>
            <h2>Все уведомления</h2>
        </div>
    </div>
    ${renderNotificationsList()}
</section>
</div>
    `
}

```

Листинг Г.7 – OverviewPage.js

```

export function renderOverviewPage({
    state,
    escapeHtml,
    fullUserName,
    renderMetaRow,
    renderRequestsTable,
}) {
    const history = state.personalHistory
    const policy = state.activePolicy
    const recentRequests = state.requests.slice(0, 5)
    const profileName = history?.employeeFullName || (state.authUser ?
fullUserName(state.authUser) : 'Профиль не найден')
    const profilePosition = history?.positionName || state.authUser?.position ||
    '...'

    return `
        <div class="panel-grid panel-grid-overview">
            <section class="panel panel-hero">
                <div class="panel-head">
                    <div>
                        <p class="eyebrow">Профиль</p>
                        <h2>Сводка по отпуску</h2>
                    </div>
                </div>
                <div class="metric-grid">
                    <div class="metric-card">
                        <span class="metric-value">${state.availableDays ?? '-'}</span>
                        <span class="metric-label">доступно дней</span>
                    </div>
                    <div class="metric-card">
                        <span class="metric-value">${state.unreadCount ?? 0}</span>
                        <span class="metric-label">непрочитанных уведомлений</span>
                    </div>
                    <div class="metric-card">

```

```

        <span class="metric-value">${state.requests.length}</span>
        <span class="metric-label">всего заявок</span>
      </div>
    </div>
    <div class="profile-meta">
      ${renderMetaRow('Сотрудник', profileName)}
      ${renderMetaRow('Email', history?.employeeEmail ||
state.authUser?.email || '-')}
      ${renderMetaRow('Отдел', history?.departmentName ||
state.department?.name || '-')}
      ${renderMetaRow('Должность', profilePosition)}
    </div>
  </section>

  <section class="panel">
    <div class="panel-head">
      <div>
        <p class="eyebrow">Активная политика</p>
        <h2>${escapeHtml(policy?.name || 'Политика не найдена')}</h2>
      </div>
    </div>
    <div class="profile-meta">
      ${renderMetaRow('Тип отпуска', policy?.vacationType || '-')}
      ${renderMetaRow('Дней в год', policy ?
String(policy.annualVacationDays) : '-')}
      ${renderMetaRow('Первая часть', policy ? `от
${policy.minVacationDaysPerRequest} дн.` : '-')}
      ${renderMetaRow('Перенос остатка', policy ?
`${policy.maxCarryOverDays} дн.` : '-')}
    </div>
  </section>

  <section class="panel panel-full">
    <div class="panel-head">
      <div>
        <p class="eyebrow">Последние заявки</p>
        <h2>История отпусков</h2>
      </div>
    </div>
    ${recentRequests.length ? renderRequestsTable(recentRequests, {
managerMode: false }) : '<div class="empty-state">Заявки пока не
созданы.</div>'}
  </section>
</div>
}

```

Листинг Г.8 – RequestsPage.js

```

export function renderRequestsPage({ state, renderRequestsTable,
renderAuditTimeline }) {
  return `
    <div class="split-layout">
      <section class="panel">
        <div class="panel-head">
          <div>
            <p class="eyebrow">Мои заявки</p>
            <h2>Управление отпусками</h2>
          </div>
        </div>
        ${renderRequestsTable(state.requests, { managerMode: false })}
      </section>
    </div>
  `
}

```

```

    <section class="panel">
      <div class="panel-head">
        <div>
          <p class="eyebrow">Аудит</p>
          <h2>История действий</h2>
        </div>
      </div>
      ${renderAuditTimeline()}
    </section>
  </div>
  ,
}

```

Листинг Г.9 – Main.js

```

import './style.css'
import { createApp } from './app.js'

createApp(document.querySelector('#app'))

```