

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»
Специализация: «Эксплуатация вычислительных машин,
комплексов, систем и сетей специального
назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

STAFFPLAN: ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
КАДРОВОГО УЧЕТА

Студент Волков 09.06.25 Волков Валентин Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель Егоров 09.06.25 Егоров В.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков 09.06.25 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент Дочкин Пенз ГЭК 09.06.25 Тегерский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 28.05.2025 №16)

Заведующий кафедрой Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19.06.2025 № 2)

Секретарь ГЭК Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

Митрохин М.А.

«6» 05 2025 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Волков Валентин Алексеевич гр. 20ВОЭ1 факультета вычислительной техники.

2. Тема работы: «StaffPlan: программное обеспечение кадрового учета».

Тема утверждена приказом ПГУ №601/о от 06.05.2025 г.

3. Руководитель работы к.т.н., доцент каф. ВТ «ПГУ» Егоров В.Ю.

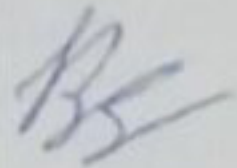
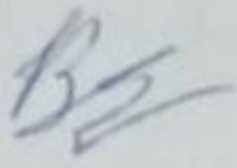
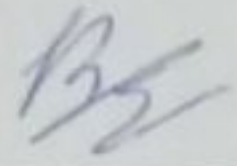
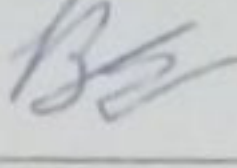
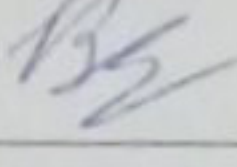
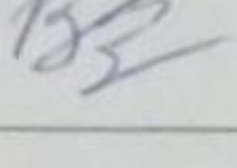
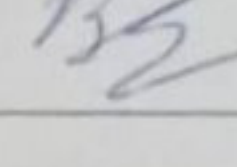
4. Задание на дипломную работу (назначение разработки, исходные данные и т.п.).

Разработать программное обеспечение кадрового учета и управления штатным расписанием на предприятии. Программное обеспечение должно хранить в себе сведения о структурных подразделениях, должностях и сотрудников предприятия. Формировать и вести штатное расписание, отражающего информацию о количестве сотрудников по каждой должности, в каждом структурном подразделении и установленные оклады. Программное обеспечение должна разрабатываться при помощи среды разработки “React” и “Node.js”, а база данных – при помощи “PostgreSQL”.

5. Перечень подлежащих разработке вопросов.

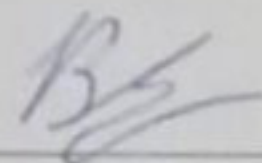
Требуется произвести анализ предметной области, определить функциональный состав программы, выбрать стек технологий для реализации программы, разработать диаграммы: вариантов использования и развертывания, спроектировать модели базы данных: концептуальная, логическая и физическая, разработать структуры программы: клиентская и серверная части, с последующим написанием программного кода.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	10	06.05.25 – 08.05.25	
2	Проектирование и отладка программного обеспечения	10	08.05.25 – 09.05.25	
3	Разработка системы и базы данных	20	09.05.25 – 11.05.25	
4	Отладка и тестирование	10	11.05.25 – 17.05.25	
5	Разработка экономической и БЖД частей	10	17.05.25 – 19.05.25	
6	Оформление графической части	10	19.05.25 – 25.05.25	
7	Оформление выпускной работы	30	25.05.25 – 30.05.25	

Дата выдачи задания «06» мая 2025г.

Руководитель дипломной работы

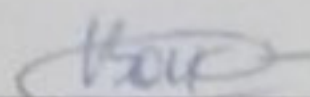


(подпись, дата)

06.05.2025

Егоров В.Ю.

Задание к исполнению принял студент

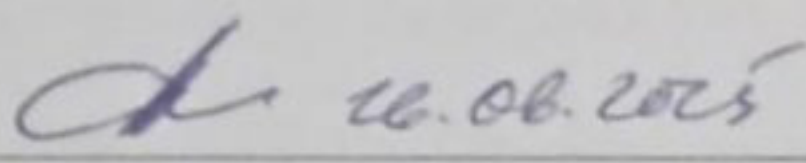


(подпись, дата)

06.05.2025

Волков В.А.

Работу к защите допустить
Декан ФВТ



(подпись, дата)

16.06.2025 Л.Р. Фионова

Реферат

Пояснительная записка содержит 145 листов, 26 рисунков, 29 таблицы, 9 источников и 4 приложения.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ШТАТНОЕ РАСПИСАНИЕ ПРЕДПРИЯТИЯ, СОТРУДНИКИ ПРЕДПРИЯТИЯ, ОТДЕЛ КАДРОВ, РАЗРАБОТКА САЙТА, БАЗА ДАННЫХ

Объектом исследования является программное обеспечение для работников отдела кадров, хранящей сведения о сотрудниках, отделах и штатном расписании предприятия.

Цель работы – разработать программное средство для помощи в работе сотрудников отдела кадров по взаимодействию с сотрудниками, должностями и штатным расписанием предприятия, а также контекстную и детализирующую диаграмму. Спроектировать концептуальную, реляционную и физическую модели данных.

Технологии разработки – клиентская и серверная часть системы реализована на языке JavaScript при помощи среды визуального программирования “React” и “Node.js”. Хранение данных организовано с помощью СУБД PostgreSQL.

В результате проделанной работы разработано программное средство для работы отдела кадров, хранящей в себе сведения о штатном расписании предприятия, с помощью которого можно вести учет информации о сотрудниках, об отделах и должностях.

					ПГУ 1.090501.06.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	StaffPlan: программное обеспечение кадрового учета Пояснительная записка	Лит.	Лист	Листов
Разраб.		Волков В.А.		09.06.25				
Пров.		Егоров В.Ю.		09.06.25			2	145
Реценз.		Петерсен		09.06.25		ФВТ, каф. ВТ гр. 20ВОЭ1		
Н. Контр.		Бычков А.С.		09.06.25				
Утв.		Митрохин М.А.		09.06.25				

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Анализ предметной области.....	8
1.1 Описание предметной области предприятия	8
1.2 Анализ текущего порядка ведения учета	9
1.3 Цели и задачи разрабатываемого программного средства	11
1.4 Анализ целевой аудитории	12
2 Техническое задание	13
2.1 Назначение разработки.....	13
2.2 Требования к программе	13
2.2.1 Требования к функциональным характеристикам.....	13
2.2.2 Требования к надежности	14
2.2.3 Требования к составу и параметрам технических средств	14
2.2.4 Требования к информационной и программной совместимости	15
2.3 Требования к программной документации	15
2.4 Порядок контроля и приемки.....	16
3 Обоснование выбранных средств разработки.....	17
3.1 Существующие аналоги	18
4 Проектирование системы	21
4.1 Концептуальное проектирование баз данных.....	25
4.2 Разработка логической модели данных	26
4.3 Физическое проектирование базы данных	30
4.4 Анализ функций приложения	34
4.5 Описание логической структуры.....	37
4.5 Методы испытаний	41
5 Оценка затрат на разработку и внедрение программного средства	44
5.1 Организация производства разработки программного средства	44
5.2 Расчет налогового обложения сотрудников.....	50

5.3 Оценка эффективности программного средства.....	51
5.4 Анализ рисков при разработке и внедрении программного средства.....	52
6 Безопасность жизнедеятельности.....	54
6.1 Требование к рабочему месту инженера-разработчика	54
6.2 Требования к микроклимату в рабочем помещении	55
6.3 Техника безопасности при работе с персональным компьютером.....	57
Заключение	59
Список использованных источников	60
Приложение А – UML – диаграммы	61
Приложение А.1 – Диаграмма последовательностей.....	62
Приложение А.2 – Диаграмма развертывания	63
Приложение А.3 – Диаграмма компонентов	64
Приложение А.4 – Диаграмма вариантов использования.....	65
Приложение А.5 – Диаграмма деятельности	66
Приложение Б – Результаты функционального тестирования.....	67
Приложение В – Листинг программы.....	78
Приложение Г – Презентация	139

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

ИТ – информационные технологии

ПО – программное обеспечение

ПК – персональный компьютер

ПФР – пенсионный фонд России

СУБД – система управления базами данных

ФОМС – фонд обязательного медицинского страхования

ФСС – фонд социального страхования

ЭВМ – электронная вычислительная машина

1НФ – первая нормальная форма

2НФ – вторая нормальная форма

3НФ – третья нормальная форма

FK (Foreign Key) – внешний ключ

ID (Identifier) – идентификатор сущности

PK (Primary Key) – первичный ключ

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

					ПГУ 1.090501.06.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В условиях современной цифровизации всё больше задач, ранее решавшихся вручную, переносятся в электронную среду. Одной из таких задач является ведение штатного расписания предприятия. Для эффективной работы отдела кадров требуется точный, структурированный и доступный в любой момент времени учёт сведений о сотрудниках, должностях и отделах. Бумажные документы, а также разрозненные таблицы в электронных редакторах не обеспечивают необходимого уровня надёжности, скорости и функциональности.

Наличие специализированного программного средства, позволяющего автоматизировать работу с информацией о штатной структуре, становится не просто удобством, а необходимостью. Такое приложение облегчает повседневную работу кадрового специалиста: позволяет быстро формировать списки сотрудников по отделам, получать сведения о начислении окладов, выводить отчётность и контролировать актуальность записей. Уменьшается вероятность ошибок при ручном вводе данных, упрощается поиск и фильтрация информации, повышается общая эффективность документооборота.

Кроме того, современное программное обеспечение даёт возможность интеграции с другими системами предприятия, обеспечивая единое информационное пространство. Это особенно важно в условиях динамичного изменения структуры компании, текучести кадров и необходимости соответствия нормативным требованиям. В результате использование автоматизированного приложения для работы с данными о штатном расписании позволяет значительно сократить временные и трудовые затраты, минимизировать ошибки и повысить точность учёта персонала.

В условиях активной цифровизации кадрового документооборота стало особенно важно использовать программные средства, которые упрощают и ускоряют выполнение рутинных задач сотрудников предприятия. Одной из таких задач является ведение штатного расписания – основного инструмента управления кадровым составом, должностями и структурными

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

подразделениями. В связи с этим было принято решение о создании специализированного приложения, позволяющего автоматизировать работу с данными о штатном расписании предприятия и обеспечить удобный доступ к информации.

Объектом дипломной работы является автоматизация учёта штатного расписания предприятия.

Предметом дипломной работы выступает приложение для работы с базой данных, содержащей сведения о штатном расписании.

Целью дипломной работы является разработка программного средства, позволяющего хранить, редактировать и обрабатывать информацию о штатной структуре предприятия с использованием базы данных и графического интерфейса.

В соответствии с поставленной целью сформулированы следующие задачи:

- проанализировать предметную область и определить требования к функциональности приложения;
- спроектировать структуру базы данных с учётом логики хранения информации о сотрудниках, отделах, должностях и начислении окладов;
- реализовать физическую базу данных средствами PostgreSQL и разработать пользовательский интерфейс в среде React и Node.js;
- обеспечить выполнение основных операций: добавление, изменение, удаление и поиск записей, а также формирование отчётности;
- провести тестирование приложения и проанализировать результаты его работы на демонстрационной базе.

1 Анализ предметной области

1.1 Описание предметной области предприятия

Предметная область разрабатываемого программного обеспечения охватывает процессы формирования, хранения и обработки информации, связанной с ведением штатного расписания на предприятии. Штатное расписание представляет собой внутренний регламентирующий документ, в котором фиксируются все должности, структурные подразделения, количественные и финансовые параметры (такие как число штатных единиц и размер окладов). Этот документ используется для расчёта фонда оплаты труда, обеспечения нормативного контроля численности персонала, а также для планирования и анализа кадровой политики организации.

В рамках текущей практики многие предприятия продолжают вести учёт штатного расписания вручную или с помощью универсальных офисных программ, таких как Microsoft Excel или Word. Это приводит к целому ряду проблем: дублирование данных, отсутствие связей между сущностями (например, между сотрудником и отделом), невозможность быстрого поиска, фильтрации и анализа информации. Изменения в структуре предприятия требуют ручной корректировки сразу в нескольких местах, что повышает риск ошибок и ведёт к потере актуальности данных.

На предприятии, для которого разрабатывается программное обеспечение, имеется определённая организационная структура, состоящая из отделов. Каждый отдел выполняет конкретные производственные или административные функции и включает в себя один или несколько сотрудников, занимающих различные должности. Должность, в свою очередь, связана с конкретным окладом, наименование должности может повторяться в разных отделах, но её привязка к конкретному подразделению определяет уникальность штатной единицы [1].

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

Каждому сотруднику присваивается табельный номер — уникальный идентификатор, по которому осуществляется дальнейшая идентификация в базе данных. Также фиксируются персональные сведения: фамилия, имя, отчество, адрес, дата приёма на работу. Данные об отделе включают его уникальный номер и наименование. Для каждой должности указывается название и базовая ставка заработной платы. Штатное расписание фактически связывает между собой эти сущности, определяя, какие сотрудники занимают какие должности в каком отделе, с какой оплатой.

Особое внимание в предметной области уделяется корректности и целостности информации. Поскольку ошибки в кадровом учёте могут повлечь за собой юридические, финансовые и организационные последствия, требуется строгая структура хранения данных и ограничение на дублирование, пропуски или некорректные значения. Также важным аспектом является возможность анализа: формирование отчётности, выборки по отделам, должностям, окладам и другим параметрам.

Программное обеспечение должно обеспечить надёжное и удобное средство для работы с этой информацией. Пользователь — сотрудник отдела кадров — должен иметь возможность легко добавлять новых сотрудников, изменять данные, переводить работников между должностями и отделами, удалять устаревшие записи, а также формировать сводные таблицы и отчёты по текущему состоянию штатного расписания. Вся работа должна происходить в интуитивно понятном интерфейсе с минимальной вероятностью ошибки. При этом сама структура базы данных должна обеспечивать логичную и чёткую взаимосвязь между всеми объектами предметной области [2].

1.2 Анализ текущего порядка ведения учета

На большинстве предприятий, где автоматизация внутренних процессов находится на начальном уровне, учёт данных о штатном расписании ведётся вручную или с использованием стандартных офисных программ, таких как Microsoft Excel, Word или Access. Такой подход распространён в небольших

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

организациях, где отсутствуют выделенные средства или ИТ-специалисты для внедрения полноценных информационных систем. Однако даже в этом случае объёмы обрабатываемой информации и её значение для кадрового управления делают ручной учёт неэффективным (таблица 1).

Таблица 1 – Общие сведения о программных продуктах аналогичного назначения

Аспект	Пример	Проблема
Инструменты учета	Excel-файл	Отсутствуют связи между таблицами, возможность пропуска данных, нарушение структуры таблицы
Обновление данных	Word-документ	Изменения фиксируются вручную, Excel остаётся без обновления – данные несогласованные, ошибки фильтрации, нарушение структуры
Проверка корректности	Access-база	Не настроены уникальные ограничения – дублирование записей и сбои в расчетах, при поломке риск полной потери информации

В текущей практике кадрового делопроизводства таблицы Excel часто используются для фиксации информации о сотрудниках, распределении их по отделам, указания должностей и окладов. Отдельные документы могут содержать сведения об организационной структуре, в других – о штатных единицах, а в-третьих – о назначенных сотрудниках. Это приводит к дублированию данных, их расхождению и невозможности обеспечить оперативную актуализацию информации. Изменение в одной таблице требует ручного редактирования нескольких других, что создаёт риск возникновения ошибок [3].

Кроме того, при использовании офисных программ отсутствует строгий контроль целостности и связности данных. Пользователь может случайно

удалить или изменить важную информацию без каких-либо ограничений или предупреждений. Также не реализуются механизмы проверки корректности вводимых значений, что нередко приводит к нарушению логики учёта: например, указание несуществующего отдела или присвоение оклада не по утверждённой ставке.

Сложности также возникают при необходимости быстрого поиска информации. Например, для формирования отчёта по всем вакантным должностям в конкретном отделе требуется вручную фильтровать данные, что занимает значительное время. Формирование отчётов для руководства или бухгалтерии сопровождается копированием, выравниванием и дополнительным форматированием таблиц, что увеличивает нагрузку на кадровый отдел.

При ручном учёте затруднено ведение истории изменений – нет возможности отследить, кто и когда внёс правки, какие данные были изменены. Это особенно критично при проведении проверок или разрешении спорных ситуаций. Также отсутствует централизованное хранилище данных, что снижает надёжность хранения информации и увеличивает риски её потери, например, при сбоях в работе компьютера или случайном удалении файла.

1.3 Цели и задачи разрабатываемого программного средства

Цели и задачи программного средства предприятия:

- повышение скорости обработки информации о сотрудниках, отделах и должностях;
- обеспечение контроля над структурой и текущим состоянием штатного расписания;
- упрощение учёта, обновления и анализа кадровой информации.

В результате внедрения программного обеспечения планируется достичь следующих задач:

- повысить эффективность работы сотрудников отдела кадров;
- снизить трудозатраты на выполнение рутинных операций;

- увеличить скорость поиска и обновления информации о штатных единицах;
- обеспечить точность, актуальность и согласованность кадровых данных;
- упростить формирование отчётности по персоналу и структуре предприятия.

1.4 Анализ целевой аудитории

Целевая аудитория разрабатываемого программного приложения – это круг пользователей, для которых оно предназначено, чьи задачи, потребности и особенности напрямую учитываются при проектировании функциональности, интерфейса и архитектуры системы. В случае системы для учёта штатного расписания предприятие, как правило, включает в себя структурированные подразделения и персонал, взаимодействующий с кадровой информацией.

Основными представителями целевой аудитории выступают:

- сотрудники отдела кадров (это ключевые пользователи системы. Именно они ежедневно взаимодействуют с информацией о штатной структуре, формируют и обновляют записи о сотрудниках, контролируют распределение должностей и создают отчётные документы для руководства);
- руководители отделов (хотя они не работают в приложении напрямую, часто получают от кадровой службы отчёты по своему подразделению);
- бухгалтерия (пользуется частью данных о штатных единицах и окладах для расчёта фонда заработной платы и других финансовых показателей);
- административный ИТ-персонал (отвечает за установку, сопровождение и техническую поддержку приложения) [4].

2 Техническое задание

2.1 Назначение разработки

Данная система разработана для автоматизации процесса учёта штатного расписания компании, что позволяет значительно облегчить работу сотрудников кадрового отдела, повысить точность ведения данных о должностях, подразделениях и персонале, а также ускорить подготовку необходимых отчётных материалов.

Главная задача проекта — создание программного продукта, обеспечивающего ведение и управление штатным расписанием предприятия с применением базы данных и удобного графического интерфейса.

2.2 Требования к программе

2.2.1 Требования к функциональным характеристикам

Разрабатываемая программа должна выполнять следующие функции:

- вести учет информации об отделах;
- вести учет информации о должностях;
- вести учет информации о сотрудниках;
- вести учет информации о штатном расписании предприятия;
- осуществлять выполнение запросов пользователя к базе данных, таких как: список должностей по алфавиту, список должностей указанного отдела, список должностей с окладом в указанном диапазоне;
- формировать отчетные документы: список отделов по алфавиту, штатное расписание указанного отдела, список сотрудников указанного отдела;
- осуществлять проверку правильности ввода данных;
- осуществлять вывод информации об авторе приложения.

Входными данными программного средства является штатное расписание предприятия. Выходными данными — информация, выводимая из базы данных по запросам пользователя.

2.2.2 Требования к надежности

Программное обеспечение должно обладать высокой устойчивостью к ошибкам и сбоям во время работы. Предотвращение аварийного завершения при действиях пользователя обеспечивается за счёт тщательно продуманной обработки исключений, встроенного контроля корректности вводимых данных и последовательной проверки их валидности на всех этапах взаимодействия с базой данных. Каждое поле ввода в интерфейсе снабжено механизмами валидации, которые не допускают сохранение пустых, некорректных или повторяющихся значений. Дополнительный уровень надёжности достигается обработкой исключительных ситуаций на стороне клиентского приложения: при возникновении ошибки выполнение критических операций приостанавливается, пользователю выводится информативное сообщение, а система возвращается в стабильное состояние без потери информации.

В самой базе данных реализованы триггеры, предотвращающие нарушения ссылочной целостности и автоматически управляющие полями с автоинкрементом. Генерация первичных ключей организована так, чтобы минимизировать вероятность конфликтов при одновременном доступе к базе данных. Приложение построено на событийно-ориентированной архитектуре, что гарантирует строгий порядок вызова обработчиков и облегчает контроль выполнения операций. Надёжность взаимодействия клиента с сервером обеспечивается использованием проверенных компонентов для работы с СУБД PostgreSQL и применением устойчивых проектных паттернов. Все перечисленные меры способствуют обеспечению высокой надёжности и стабильности работы системы в реальных условиях эксплуатации.

2.2.3 Требования к составу и параметрам технических средств

Программа должна быть предназначена для использования как на локальном компьютере, так и в сети. Локальный компьютер или рабочая станция в сети должны иметь следующие технические характеристики:

- тактовая частота процессора – не менее 3,99 МГц;
- оперативная память – не менее 16 Гбайт;

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

- на жестком диске при установке используется около 1 Гбайт;
- объем жесткого диска должен быть не менее 500 Гбайт.

Локальный компьютер или сервер в сети должен иметь следующие технические характеристики:

- тактовая частота процессора – не менее 3,99 МГц;
- оперативная память – не менее 16 Гбайт;
- на жестком диске при установке используется около 1 Гбайт;
- объем жесткого диска должен быть не менее 500 Гбайт.

2.2.4 Требования к информационной и программной совместимости

Программное обеспечение должно поддерживать работу как на локальной машине, так и в сетевой среде.

При использовании на локальном компьютере необходимо установить СУБД PostgreSQL, разместить файл базы данных, а также обеспечить наличие файла с приложением для взаимодействия с этой базой.

В случае эксплуатации программы на рабочей станции в сети, СУБД PostgreSQL и файл базы данных должны располагаться на сервере баз данных, тогда как на рабочей станции будет размещён файл клиент-серверного приложения. Разрабатываемое приложение рассчитано на работу под управлением операционной системы Windows 10.

2.3 Требования к программной документации

Программная документация к разрабатываемому приложению должна содержать следующие разделы:

- техническое задание;
- диаграмма вариантов использования;
- диаграмма развертывания;
- описание контекстного проектирования БД;
- описание логического проектирования БД;
- описание физического проектирования БД;
- проектирование приложения;

- описание программного средства;
- описание применения;
- текст программы.

2.4 Порядок контроля и приемки

Для проверки корректности работы программы и всех её функций в соответствии с техническим заданием необходимо разработать набор тестовых сценариев и проверить, как программа реагирует на различные действия пользователя: запуск приложения, ввод данных в поля, взаимодействие с базой данных, выбор пунктов меню и нажатие кнопок.

Программа будет считаться соответствующей техническому заданию, если при различных вариантах входных данных из тестовых сценариев она корректно выполняет все предусмотренные функции.

Приёмка программного продукта должна проводиться после успешного выполнения всех функций, а также при наличии полного комплекта документации, включающего пояснительную записку с приложениями: исходным кодом программы и результатами тестирования.

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		16

3 Обоснование выбранных средств разработки

Для реализации программного средства, предназначенного для хранения и обработки сведений о штатном расписании предприятия, были выбраны технологии JavaScript, React, Node.js и PostgreSQL. Такой стек выбран не случайно – каждая из технологий закрывает конкретные потребности проекта и решает реальные задачи, которые возникают при разработке удобного, быстрого и надёжного инструмента для отдела кадров.

React выбран как основа клиентского интерфейса по нескольким причинам. Во-первых, он позволяет быстро собирать интерфейс из компонентов – каждый раздел (отделы, сотрудники, должности) оформляется как независимый модуль, который легко тестируется, переиспользуется и масштабируется. Во-вторых, React прекрасно справляется с динамическим обновлением данных на странице – это особенно актуально для живых таблиц, фильтров и поиска по штатному расписанию. За счёт Virtual DOM интерфейс работает плавно даже при большом объёме данных.

Кроме того, React имеет большое и активное сообщество, массу готовых решений – от таблиц до графиков – что позволяет ускорить разработку без потери качества. Работа с формами, валидацией и REST-запросами реализуется легко и понятно, что критично при создании удобного инструмента для HR-специалистов.

Серверная часть реализована на Node.js, так как он идеально сочетается с клиентским JavaScript-кодом, что позволяет использовать единый язык на всём протяжении разработки. Это упрощает логику взаимодействия и ускоряет работу над проектом. Node.js отлично справляется с задачами, требующими быстрого обмена данными между клиентом и сервером, особенно при реализации REST API для работы с базой данных штатного расписания.

Асинхронная модель Node.js позволяет эффективно обрабатывать множество запросов без блокировок, что делает систему отзывчивой при одновременной работе нескольких пользователей. Благодаря большому

количеству модулей и библиотек (например, Express.js), реализовать маршрутизацию, авторизацию, валидацию и работу с PostgreSQL можно быстро и без лишней сложности.

В качестве СУБД выбрана PostgreSQL, поскольку она отлично подходит для хранения структурированных данных, которые имеют чёткие связи между таблицами – именно такой подход используется при работе со штатным расписанием. PostgreSQL поддерживает сложные SQL-запросы, транзакции и процедуры, что критично при реализации отчётности и аналитики.

Поддержка внешних ключей, ограничений и триггеров позволяет обеспечить целостность и согласованность данных – никакой сотрудник не попадёт в расписание, если его нет в таблице сотрудников, и наоборот. Кроме того, PostgreSQL хорошо масштабируется, а значит, в будущем систему можно будет использовать на предприятиях с большим количеством сотрудников и отделов без переписывания архитектуры.

3.1 Существующие аналоги

Сравнительный анализ аналогов программного обеспечение приведен в таблице 2.

Таблица 2 – Сравнение положительных и отрицательных сторон аналогичных программных обеспечений

Система	Плюсы	Минусы
1С:Зарплата и управление персоналом	Широкий функционал Поддержка законодательства Интеграция с бухгалтерией	Высокая стоимость Сложный интерфейс Перенасыщение для малого бизнеса
Контур.Персонал	Удобный интерфейс Работа из браузера Поддержка электронной отчётности	Нет локального развёртывания Зависимость от интернета Платная подписка

Система	Плюсы	Минусы
ЗУП Cloud (БухСофт)	Установка на ПК Экспорт в Excel Прост в использовании	Ограниченный функционал Не поддерживает сложные структуры, только онлайн
Kadroff.NET	Установка на ПК Экспорт в Excel Прост в использовании	Устаревший интерфейс Нет web-версии Плохо масштабируется
SAP SuccessFactors	Мощный функционал Гибкая настройка Международный стандарт	Очень высокая стоимость Долгое внедрение Требует специалистов
Разрабатываемая система	Простота и понятность Современные технологии Локальное развёртывание Упор на штатное расписание Быстрое внедрение	Нет встроенной интеграции с бухгалтерией Ограничена рамками дипломного проекта (пока нет коммерческой поддержки)

1С:Зарплата и управление персоналом (1С:ЗУП). Один из наиболее распространённых и мощных продуктов на рынке автоматизации кадрового учёта. Программа предлагает широкий функционал. Также реализована возможность формирования отчётности по различным формам и интеграции с бухгалтерским и налоговым учётом. Однако для небольших предприятий и локальных задач 1С оказывается избыточной. Она требует значительных финансовых затрат на приобретение лицензии, внедрение, сопровождение и обучение сотрудников. Кроме того, интерфейс системы достаточно сложный. Часто используется только часть её функций, в то время как большая часть системы остаётся невостребованной.

Контур.Персонал. Облачное решение от компании СКБ Контур, ориентированное на кадровиков и бухгалтеров. Доступ осуществляется через браузер, данные хранятся на серверах разработчика. Главное преимущество – это

простота интерфейса и отсутствие необходимости установки на компьютер. Однако у облачных решений есть и слабые стороны: отсутствие локального хранения данных, зависимость от интернет-соединения, ограниченная возможность настройки под специфические структуры предприятия. Кроме того, работа по подписке требует регулярных финансовых вложений.

ЗУП Cloud (БухСофт). Онлайн-сервис, который предоставляет базовый функционал по ведению кадрового учёта и штатного расписания. Подходит для ИП и малого бизнеса. Позволяет вести справочник сотрудников и должностей, указывать оклады, оформлять приём и увольнение, формировать штатное расписание в простой форме. Несмотря на удобство и доступность, платформа подходит в основном для небольших организаций с простой структурой. В ней отсутствует гибкость настройки, нет поддержки сложной иерархии подразделений, отсутствует возможность локального размещения системы на предприятии.

Kadroff.NET. Программа для учёта кадров и штатного расписания, предназначенная для установки на ПК. Позволяет вести списки сотрудников, подразделений, должностей, указывать ставки и оклады. Может экспортировать данные в Excel, поддерживает базовые отчёты. Основной минус — морально устаревший интерфейс и ограниченная функциональность. Отсутствует web-доступ, нет многопользовательского режима, система плохо масштабируется. Также отсутствует API для расширения или интеграции с другими продуктами.

SAP SuccessFactors. Крупная международная HRM-платформа, предназначенная для комплексного управления персоналом. В рамках системы реализовано стратегическое планирование, развитие сотрудников, оценка эффективности, а также ведение штатного расписания и организационной структуры. Продукт ориентирован на крупные компании и корпорации. Он требует значительных вложений, внедрение занимает месяцы, необходима команда ИТ-специалистов и обучение персонала. Для средних и малых предприятий система оказывается чрезмерно громоздкой и финансово недоступной.

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		20

4 Проектирование системы

Сотрудник кадрового отдела предприятия имеет в распоряжении данные о сотрудниках, включая табельный номер, фамилию, адрес и занимаемую должность согласно штатному расписанию.

Кроме того, этот сотрудник владеет информацией об отделах предприятия, в том числе их наименованиями.

Также в его распоряжении находятся сведения о должностях, действующих на предприятии, включая название каждой должности и соответствующий оклад по штатному расписанию.

Приложение предназначено для обработки сведений о выдаче оклада сотрудникам и выполняет следующие функции:

- вести учет информации о сотрудниках;
- вести учет информации об отделах;
- вести учет информации о штатном расписании предприятия;
- вести учет информации о должностях;
- осуществлять выполнение запросов пользователя к базе данных, таких как: список должностей по алфавиту, список должностей указанного отдела, список должностей с окладом в указанном диапазоне;
- формировать отчетные документы: список отделов по алфавиту, штатное расписание указанного отдела, список сотрудников указанного отдела.

Диаграмма содержит два внешних объекта — «Сотрудник отдела кадров» и «Сотрудники», а также обобщённый процесс под названием «Учет информации о сотрудниках».

Внешний объект «Сотрудник отдела кадров» взаимодействует с обобщённым процессом «Учет информации о сотрудниках» через входящий поток данных «Сведения о штатном расписании», по которому передаётся информация о штатном расписании. Также этот объект связан с тем же процессом посредством исходящего потока «Сведения о должностях», через который передаются данные о должностях.

На основе предметной области предприятия была создана контекстная диаграмма, представленная на рисунке 1.

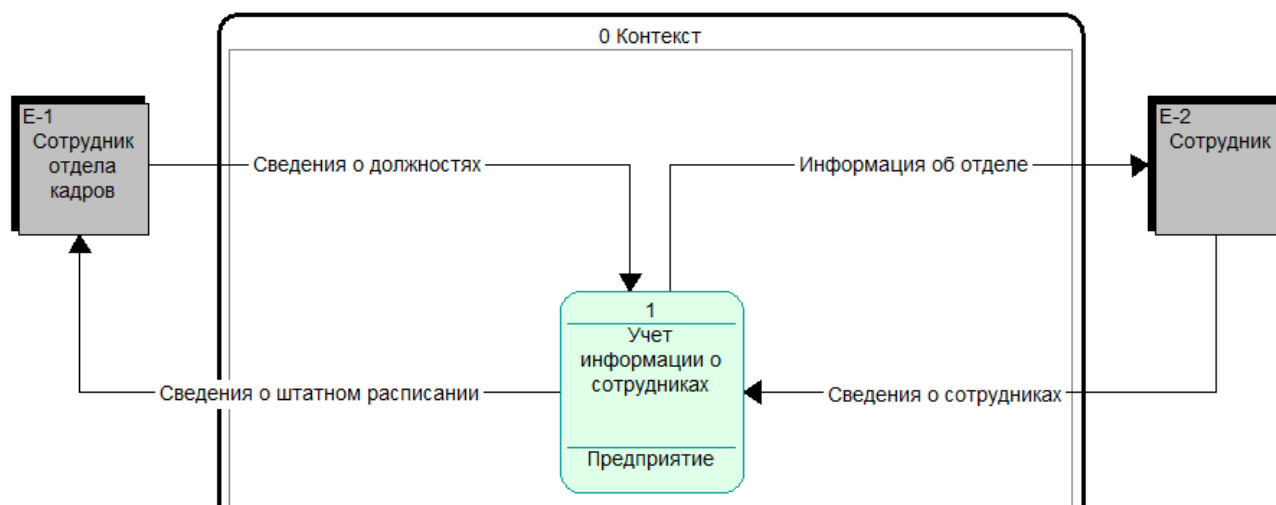


Рисунок 1 – Контекстная диаграмма

Внешний объект «Сотрудник» взаимодействует с обобщённым процессом «Учет информации о сотрудниках» через входящий поток «Информация об отделе», по которому передаётся информация об отделе. Кроме того, этот объект связан с тем же процессом посредством исходящего потока «Сведения о сотрудниках», передающего данные о сотрудниках.

На основе анализа предметной области в модель были добавлены следующие накопители данных: «Otdel», «ShtRas», «Doljn» и «Sotrud».

Внешний объект «Сотрудник отдела кадров» связан с процессом «Учет информации о должностях» через исходящий поток «Сведения информации о должностях», а также с процессом «Учет информации о штатном расписании» через входящий поток «Сведения о штатном расписании».

Внешний объект «Сотрудник» взаимодействует с процессом «Учет информации о сотрудниках» посредством исходящего потока «Сведения о сотруднике» и с процессом «Учет информации об отделе» через входящий поток «Информация об отделе».

Детализированная диаграмма представлена на рисунке 2.

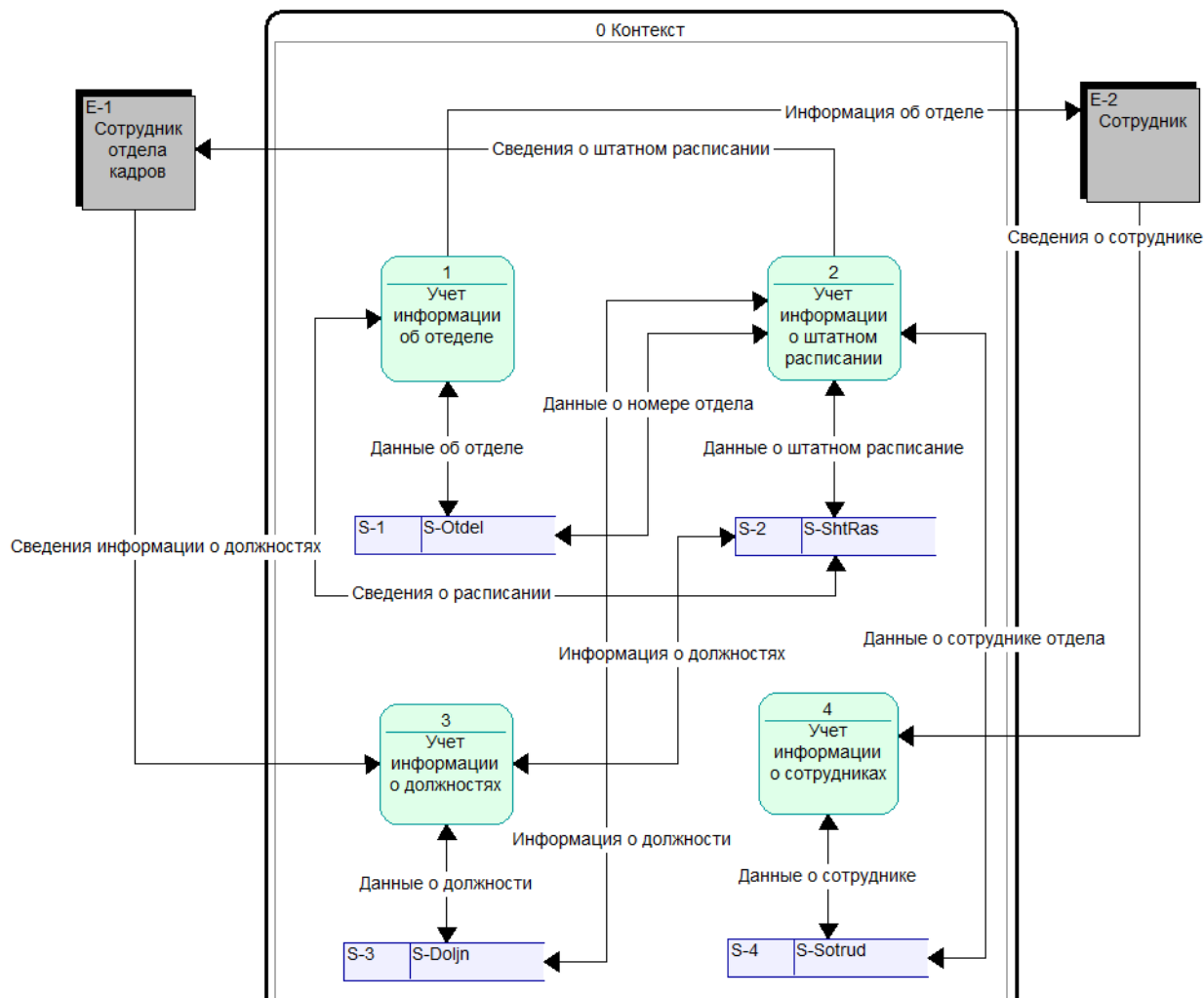


Рисунок 2 – Детализирующая диаграмма

Процесс «Учет информации об отделе» связан с накопителями данных «Otdel» и «ShtRas» посредством двунаправленных потоков «Данные об отделе» и «Сведения о расписании» соответственно.

Процесс «Учет информации о штатном расписании» взаимодействует с накопителями «ShtRas», «Otdel», «Doljn» и «Sotrud» через двунаправленные потоки «Данные о штатном расписании», «Данные о номере отдела», «Информация о должности» и «Данные о сотруднике отдела» соответственно.

Процесс «Учет информации о должностях» имеет двунаправленные связи с накопителями «Doljn» и «ShtRas» через потоки «Данные о должности» и «Информация о должности» соответственно.

Процесс «Учет информации о сотрудниках» связан с накопителем «Sotrud»

с помощью двунаправленного потока «Данные о сотруднике».

В ходе анализа предметной области в модель были добавлены следующие накопители данных: «Otdel», «ShtRas», «Doljn» и «Sotrud».

Описание структур данных для накопителей «Otdel», «ShtRas», «Doljn», «Sotrud» приведено в таблицах 3–6 соответственно.

Таблица 3 – Описание структуры данных Otdel

Имя атрибута	Назначение
NO	Уникальный номер отдела
NAZV	Название отдела

Таблица 4 – Описание структуры данных ShtRas

Имя атрибута	Назначение
NSH	Уникальный номер штатного расписания

Таблица 5 – Описание структуры данных Doljn

Имя атрибута	Назначение
ND	Уникальный номер должности
OKLAD	Оклад по штатному расписанию
NAZV	Название должности

Таблица 6 – Описание структуры данных Sotrud

Имя атрибута	Назначение
TB	Табельный номер
FAM	Фамилия сотрудника
SADRES	Адрес сотрудника

Вследствие выполнения этих действий были разработаны контекстная и детализирующая диаграммы, а также структуры данных.

4.1 Концептуальное проектирование баз данных

На основе анализа предметной области предприятия разработана концептуальная модель данных.

Концептуальная модель данных приведена на рисунке 3 и содержит 4 сущности.

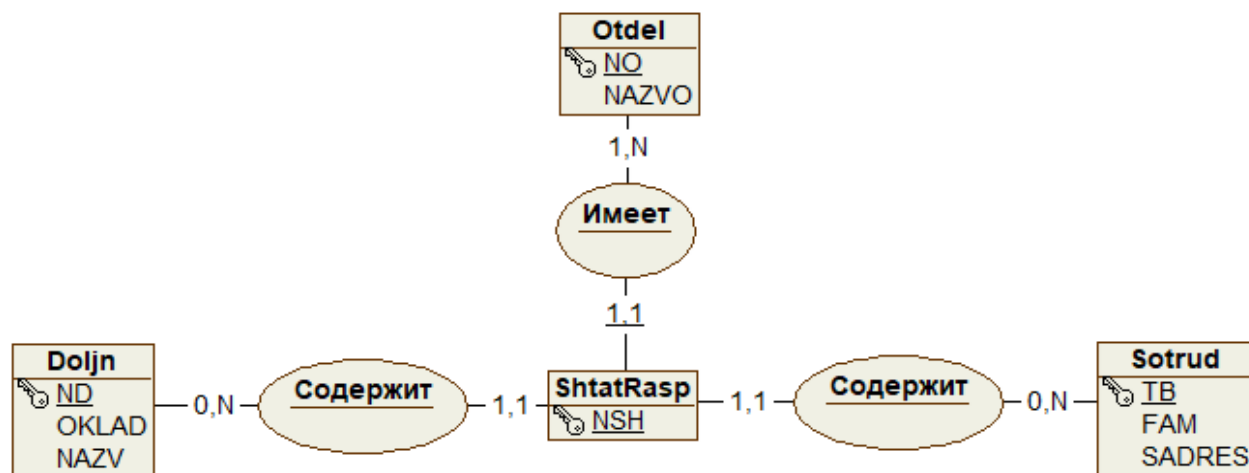


Рисунок 3 – Концептуальная модель данных

Описание сущностей и их атрибутов приведено в таблицах 7–10.

Таблица 7 – Описание структуры данных Doljn

Имя атрибута	Назначение	Первичный ключ
ND	Номер должности	РК
OKLAD	Оклад по должности	-
NAZV	Название отдела	-

Таблица 8 – Описание структуры данных ShtatRasp

Имя атрибута	Назначение	Первичный ключ
NSH	Номер штат. расписания	РК

Таблица 9 – Описание структуры данных Otdel

Имя атрибута	Назначение	Первичный ключ
NO	Номер отдела	РК

Имя атрибута	Назначение	Первичный ключ
NAZVO	Название отдела	-

Таблица 10 – Описание структуры данных Sotrud

Имя атрибута	Назначение	Первичный ключ
TB	Табельный номер сотрудника	РК
FAM	Фамилия сотрудника	-
SADRES	Адрес сотрудника	-

Сущность «Doljn» связана с сущностью «ShtatRasp» отношением с кардинальностью 0,N–1,1, поскольку одна конкретная должность входит в штатное расписание одного отдела, но одинаковые должности могут присутствовать в разных отделах. Степень участия этой связи неполная, так как сущность «Doljn» способна существовать независимо от сущности «ShtatRasp».

Сущность «Otdel» связана с сущностью «ShtatRasp» отношением с кардинальностью 1,N–1,1, поскольку в одном отделе может работать несколько сотрудников, тогда как каждый конкретный сотрудник относится к одному отделу. Степень участия связи между «ShtatRasp» и «Otdel» является полной, так как «ShtatRasp» не может существовать без связанного отдела.

Сущность «ShtatRasp» связана с сущностью «Sotrud» отношением с кардинальностью 1,1–0,N, поскольку один сотрудник может занимать несколько должностей, однако конкретную должность занимает только один сотрудник. Степень участия связи неполная, поскольку сущность «Sotrud» может существовать без привязки к «ShtatRasp».

В результате выполнения этих действий была создана концептуальная модель данных.

4.2 Разработка логической модели данных

На основе концептуальной модели данных была разработана логическая (реляционная) модель.

Реляционная модель базы данных приведена на рисунке 4 и содержит 4 сущности.

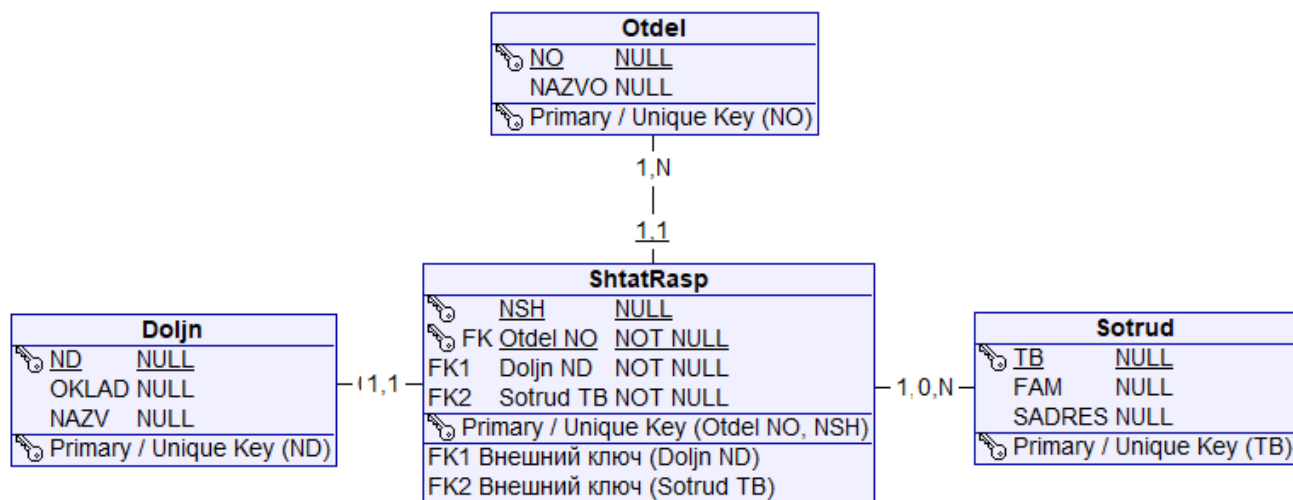


Рисунок 4 – Реляционная модель базы данных

Описание атрибутов сущностей логической модели приведено в таблицах 11–15.

Таблица 11 – Описание атрибутов сущности Doljn

Имя атрибута	Назначение	Первичный ключ	Внешний ключ
ND	Номер должности	РК	-
OKLAD	Оклад по должности	-	-
NAZV	Название отдела	-	-

Таблица 12 – Описание атрибутов сущности ShtatRasp

Имя атрибута	Назначение	Первичный ключ	Внешний ключ
NSH	Номер штатного расписания	РК	-
NO	Номер отдела	РК	FK
ND	Номер должности	-	FK

Имя атрибута	Назначение	Первичный ключ	Внешний ключ
ТВ	Табельный номер сотрудника	-	FK

Таблица 13 – Описание атрибутов сущности Otdel

Имя атрибута	Назначение	Первичный ключ	Внешний ключ
NO	Номер отдела	PK	-
NAZVO	Название отдела	-	-

Таблица 14 – Описание атрибутов сущности Sotrud

Имя атрибута	Назначение	Первичный ключ	Внешний ключ
ТВ	Табельный номер сотрудника	PK	-
FAM	Фамилия сотрудника	-	-
SADRES	Адрес сотрудника	-	-

База данных находится в 1НФ, если на пересечении каждой строки и столбца содержится только одно значение (отсутствуют повторяющиеся группы данных).

База данных, логическая модель которой представлена на рисунке 1, находится в 1НФ. Для подтверждения того, что она находится в 1НФ необходимо рассмотреть каждую сущность:

- сущность Doljn имеет ключевой атрибут ND. Атрибуты OKLAD и NAZV данной сущности являются простыми и неделимыми;
- сущность Otdel имеет ключевой атрибут NO. Атрибут NAZVO данной сущности является простым и неделимым;
- сущность ShtatRasp имеет составной ключевой атрибут, состоящий из атрибутов NO и NSH. Атрибуты ND и ТВ данной сущности являются простыми и неделимыми;

– сущность Sotrud имеет ключевой атрибут ТВ. Атрибуты FAM и SADRES являются простыми и неделимыми.

База данных находится во 2НФ, если она находится в 1НФ и каждый атрибут, не входящий в состав первичного ключа, характеризуется полной функциональной зависимостью от этого первичного ключа.

База данных, логическая модель которой представлена на рисунке 1, находится во 2НФ, так как она находится в 1НФ и все не ключевые атрибуты всех сущностей функционально полно зависят от первичных ключей этих сущностей. Для подтверждения того, что она находится во 2НФ необходимо рассмотреть каждую сущность:

– сущность Doljn находится во 2НФ, так как имеет единственный ключевой атрибут ND. Не ключевые атрибуты OKLAD и NAZV данной сущности функционально полно зависят от данного первичного ключа. Следовательно, можно сделать вывод о том, что данная сущность находится во 2НФ;

– сущность Otdel находится во 2НФ, так как имеет единственный ключевой атрибут NO. Не ключевой атрибут NAZVO данной сущности функционально полно зависит от данного первичного ключа. Следовательно, можно сделать вывод о том, что данная сущность находится во 2НФ;

– сущность ShtatRasp находится во 2НФ, так как имеет составной ключевой атрибут, состоящий из атрибутов NO и NSH. Не ключевые атрибуты ND и ТВ данной сущности функционально полно зависят от данного первичного ключа. Следовательно, можно сделать вывод о том, что данная сущность находится во 2НФ;

– сущность Sotrud находится во 2НФ, так как имеет единственный ключевой атрибут ТВ. Не ключевые атрибуты FAM и SADRES данной сущности функционально полно зависят от данного первичного ключа. Следовательно, можно сделать вывод о том, что данная сущность находится во 2НФ.

База данных находится в 3НФ, если она находится в 1 и 2 нормальных формах и не имеет не входящих в первичный ключ атрибутов, которые

находились бы в транзитивной зависимости от этого первичного ключа.

Ни в одной из сущностей логической модели данных, представленной на рисунке 1, не существует транзитивных зависимостей:

- в сущности Doljn не ключевые атрибуты OKLAD и NAZV не зависят друг от друга. Следовательно, можно сделать вывод о том, что данная сущность логической модели находится в 3НФ;

- в сущности Otdel не ключевой атрибут NAZVO ни от кого не зависит. Следовательно, можно сделать вывод о том, что данная сущность логической модели находится в 3НФ;

- в сущности ShtatRasp не ключевые атрибуты ND и ТВ не зависят друг от друга. Следовательно, можно сделать вывод о том, что данная сущность логической модели находится в 3НФ;

- в сущности Sotrud не ключевые атрибуты FAM и SADRES не зависят друг от друга. Следовательно, можно сделать вывод о том, что данная сущность логической модели находится в 3 НФ.

В логическую модель базы данных, представленную на рисунке 1, в сущность ShatRasp добавлены внешние ключи:

- NO из сущности Otdel;
- ND из сущности Doljn;
- ТВ из сущности Sotrud;

Так как между сущностями ShtatRasp и Otdel полная степень участия, первичный ключ сущности ShtatRasp состоит из собственного первичного ключа NSH и внешнего ключа NO, являющегося первичным ключом сущности Otdel.

Вследствие выполнения этих действий была разработана логическая модель данных.

4.3 Физическое проектирование базы данных

На основе логической модели данных была разработана физическая модель данных.

Физическая модель базы данных разработана для СУБД PostgreSQL и приведена на рисунке 5.

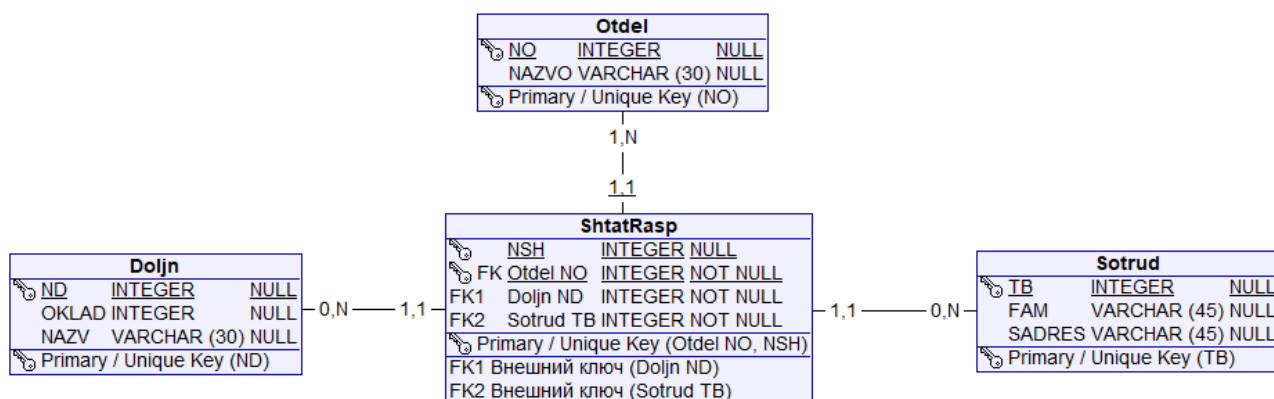


Рисунок 5 – Физическая модель данных

Физическая модель данных содержит 4 таблицы. Описание полей таблиц физической модели приведено в таблицах 15–18.

Таблица 15 – Описание таблицы Doljn

Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
ND	Номер должности	РК	Недопустимо	INTEGER
Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
OKLAD	Оклад по должности	-	Допустимо	INTEGER
NAZV	Название отдела	-	Допустимо	VARCHAR(30)

На основе физической модели данных сгенерирован файл «SHTRAS.sql», содержащий скрипт базы данных на языке PostgreSQL. Текст скрипта приведен в приложении А.

Таблица 16 – Описание таблицы ShtatRasp

Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
NSH	Уникальный номер штатного расписания	PK	Недопустимо	INTEGER
NO	Номер отдела	PK, FK	Недопустимо	INTEGER
ND	Номер должности	FK	Допустимо	INTEGER
TB	Табельный номер сотрудника	FK	Допустимо	INTEGER

Таблица 17 – Описание таблицы Otdel

Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
NO	Уникальный номер отдела	PK	Недопустимо	INTEGER
NAZVO	Название отдела	-	Допустимо	VARCHAR(30)

Таблица 18 – Описание таблицы Sotrud

Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
TB	Табельный номер сотрудника	PK	Недопустимо	INTEGER

Имя атрибута	Физическое имя	Нулевое значение	Ключевое поле	Тип данных
FAM	Фамилия сотрудника	-	Допустимо	VARCHAR(45)
SADRES	Адрес сотрудника	-	Допустимо	VARCHAR(45)

Были созданы 4 генератора для таблиц базы данных, описание генераторов приведено в таблице 19.

Таблица 19 – Описание генераторов

Таблица	Генератор
Doljn	GDOLJN
ShtatRasp	GOTDEL
Otdel	GSHTATR
Sotrud	GSOTRUD

Было разработано 4 триггера для таблиц базы данных. Описание триггеров приведено в таблице 20.

Таблица 20 – Описание триггеров

Таблица	Триггер	Назначение
Doljn	DOLJN_BI0	Заполняет автоинкрементное поле Doljn
ShtatRasp	SHTATR_BI0	Заполняет автоинкрементное поле ShtatRasp
Otdel	OTDEL_BI0	Заполняет автоинкрементное поле Otdel
Sotrud	SOTRUD_BI0	Заполняет автоинкрементное поле Sotrud

Был создан 1 триггер обновления для таблиц базы данных, описание триггеров приведено в таблице 21.

Таблица 21 – Описание триггеров

Таблица	Триггер	Поле
Otdel	OTDEL_BIU0	Обновляет автоинкрементное поле Otdel

Вследствие выполнения этих действий была разработана база данных для СУБД PostgreSQL.

4.4 Анализ функций приложения

Анализ функций приложения проведен на основе анализа предметной области и требований к функциональным характеристикам.

Результат анализа функций представлен на рисунке 6.

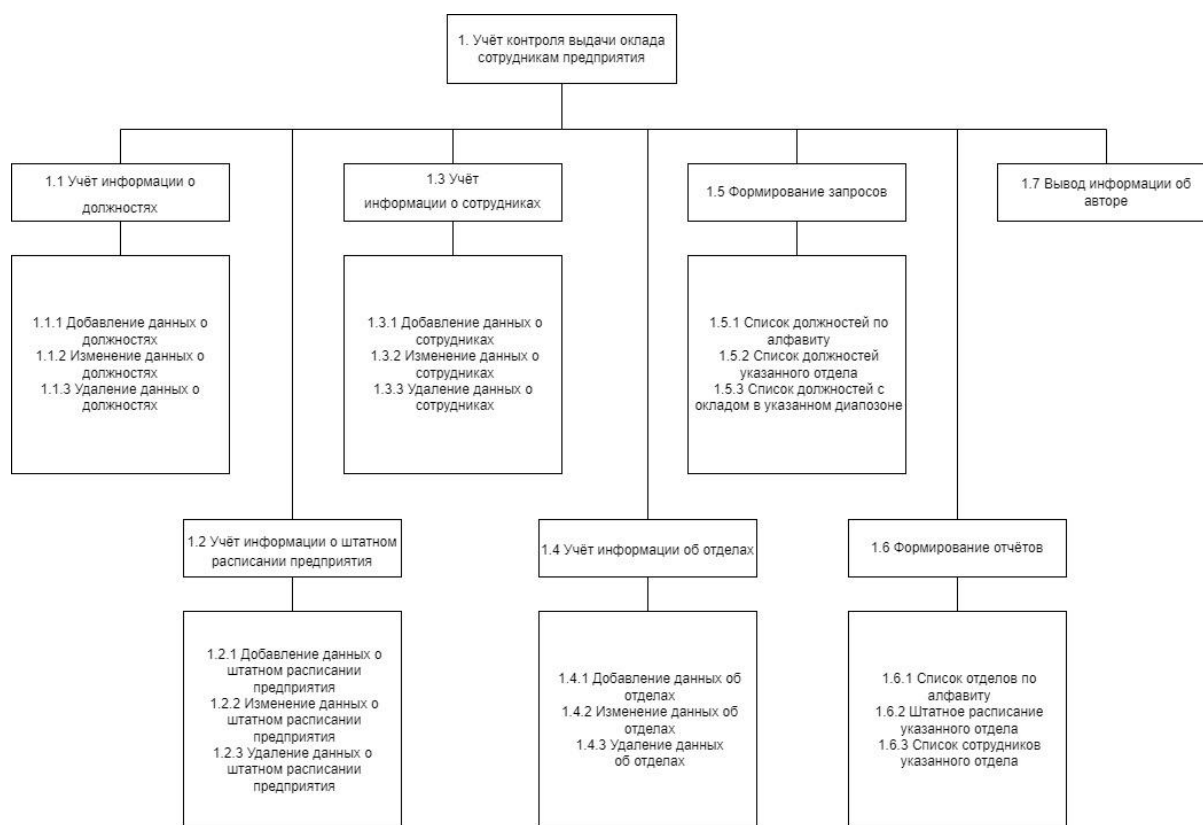


Рисунок 6 – Иерархия функций приложения

Функция «Учет информации об отделе» включает три подфункции:

— добавление данных об отделе. Для добавления информации необходимо открыть набор данных «AddOtdel», заполнить форму добавления

записи нужными сведениями о новом отделе, подтвердить добавление, затем закрыть набор данных;

– изменение информации об отделе. Для изменения данных следует открыть набор данных «EditOtdel», внести изменения в форму редактирования записи, подтвердить изменения, закрыть набор данных; – удаление информации об отделе. Для удаления необходимо открыть набор данных «Otdel», выбрать запись для удаления, подтвердить удаление, закрыть набор данных. Функция «Учет информации о штатном расписании предприятия» включает три подфункции:

– добавление данных о штатном расписании. Для этого открыть набор данных «AddSht», заполнить форму добавления новыми сведениями о штатном расписании, подтвердить добавление, закрыть набор данных; – изменение информации о штатном расписании. Для изменения открыть набор данных «EditSht», внести корректировки в форму редактирования, подтвердить изменения, закрыть набор данных;

– удаление информации о штатном расписании. Для удаления открыть набор данных «OtdelSht», выбрать запись, подтвердить удаление, закрыть набор данных.

Функция «Учет информации о должностях» состоит из трёх подфункций:

– добавление данных о должностях. Открыть набор данных «AddDoljn», заполнить форму добавления сведениями о должностях, подтвердить добавление, закрыть набор данных;

– изменение информации о должностях. Открыть набор данных «EditDoljn», внести изменения в форму редактирования, подтвердить изменения, закрыть набор данных;

– удаление информации о должностях. Открыть набор данных «Doljn», выбрать данные для удаления, подтвердить удаление, закрыть набор данных.

Функция «Выполнение запросов пользователя» к базе данных содержит три подфункции:

- вывод списка должностей по алфавиту. Для выполнения открыть набор данных «Zapros1», запустить обработчик события по нажатию соответствующей кнопки, закрыть набор данных;
- вывод списка должностей указанного отдела. Для этого открыть набор данных «Zapros2», ввести необходимые параметры, запустить обработчик события кнопкой, закрыть набор данных;
- вывод должностей с окладом в заданном диапазоне. Открыть набор данных «Zapros3», ввести параметры, запустить обработчик события, закрыть набор данных.

Функция «Формирование отчетных документов» содержит три подфункции:

- формирование отчета со списком отделов по алфавиту. Для этого выбрать пункт меню «Отчеты», выбрать соответствующий подпункт, открыть набор данных «Main», выбрать нужные данные, закрыть отчет;
- формирование отчета о штатном расписании указанного отдела. Выбрать пункт меню «Отчеты», нужный подпункт, открыть набор данных «Otchet2», выбрать данные, закрыть отчет;
- формирование отчета со списком сотрудников указанного отдела. Выбрать пункт меню «Отчеты», соответствующий подпункт, открыть набор данных «Otchet3», выбрать данные, закрыть отчет.

Функция вывода информации об авторе не содержит подфункций. Для её выполнения достаточно выбрать соответствующий подпункт меню.

Функция контроля правильности ввода данных пользователем реализуется непосредственно при вводе информации в поля форм приложения.

Соотношение функций и модулей представлено в таблице 22.

Таблица 22 - Соотношение функций и модулей

Функция	Модуль
Добавление информации об отделе	Учет информации об отделе
Изменение информации об отделе	
Удаление информации об отделе	
Добавление информации о штатном расписании	Учет информации о штатном расписании
Изменение информации о штатном расписании	
Удаление информации о штатном расписании	
Добавление информации о должностях	Учет информации о должностях
Изменение информации о должностях	
Удаление информации о должностях	
Добавление информации о сотрудниках	Учет информации о сотрудниках
Изменение информации о сотрудниках	
Удаление информации о сотрудниках	
Вывод списка должностей по алфавиту	Выполнение запросов пользователя
Вывод списка должностей указанного отдела	
Вывод списка должностей с окладом в указанном диапазоне	
Вывод списка отделов по алфавиту	Формирование отчетных документов
Вывод штатного расписания указанного отдела	
Список сотрудников указанного отдела	
Вывод информации о разработчике	Главный модуль

4.5 Описание логической структуры

Программа является событийно-управляемой.

При запуске приложения открывается главное окно программы.

Приложение имеет модульную структуру. Схема взаимодействия модулей приложения представлена на рисунке 7.

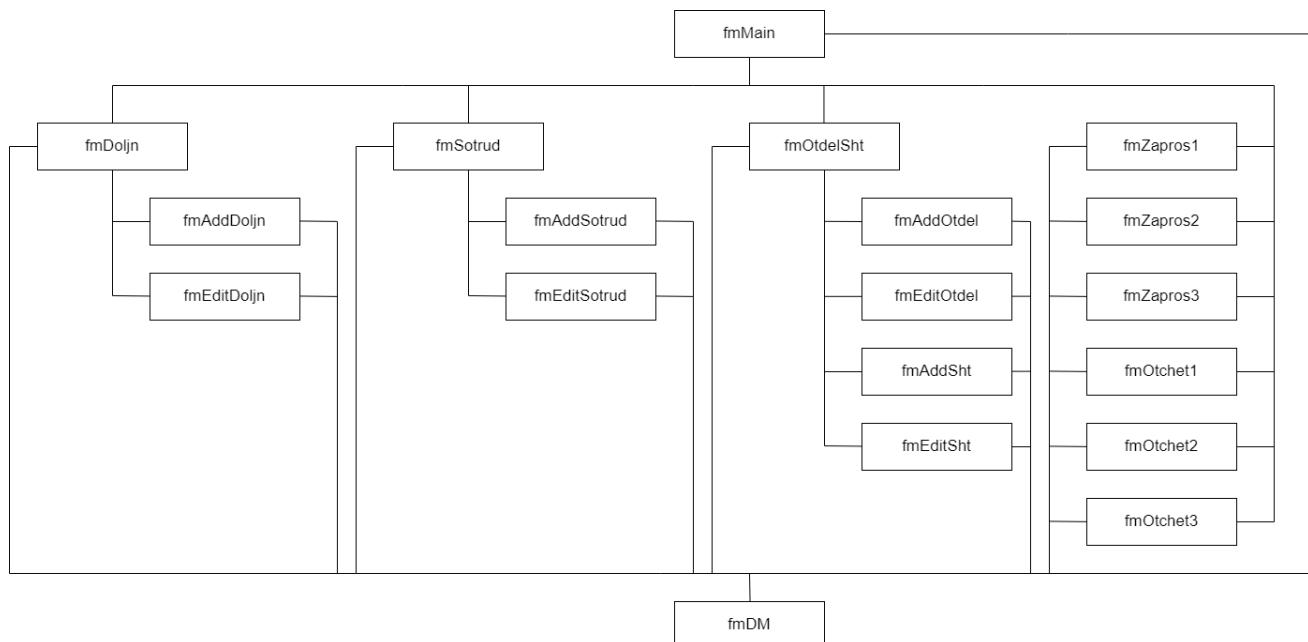


Рисунок 7 – Схема взаимодействия программных модулей

Вызов модулей приложения производится в соответствии с рисунком 6. При запуске на выполнение запускается главный модуль fmMain. Главный модуль связан с модулем fmDM. С помощью компонентов, установленных на модуле данных fmDM, осуществляется связь приложения с базой данных. Модуль данных fmDM является не визуальным и не доступен пользователю.

С помощью главного модуля fmMain можно вызвать модули:

- fmOtdelSht (для просмотра информации об отделах и просмотра, добавления, изменения, удаления о штатном расписании);
- fmDoljn (для просмотра, добавления, изменения, удаления информации о должностях);
- fmSotrud (для просмотра, добавления, изменения, удаления информации о сотрудниках);
- fmZapros1 (для выполнения запроса – вывода списка должностей по алфавиту);
- fmZapros2 (для выполнения запроса – вывода списка должностей указанного отдела);
- fmZapros3 (для выполнения запроса – вывода списка должностей с окладом в указанном диапазоне);

- fmOtchet2 (для формирования отчета о штатном расписании указанного отдела);
- fmOtchet3 (для формирования отчета о списке сотрудников указанного отдела).

С помощью модуля fmDoljn можно вызвать модули:

- fmAddDoljn (для добавления информации о должностях);
- fmEditDoljn (для изменения информации о должностях).

С помощью модуля fmSotrud можно вызвать модули:

- fmAddSotrud (для добавления информации о сотрудниках);
- fmEditSotrud (для изменения информации о сотрудниках).

С помощью модуля fmOtdelSht можно вызвать модули:

- fmAddOtdel (для добавления информации об отделах);
- fmEditOtdel (для изменения информации об отделах);
- fmAddSht (для добавления информации о штатном расписании);
- fmEditSht (для изменения информации о штатном расписании).

Все вышеуказанные модули связаны с модулем данных fmDM. С помощью компонентов, установленных на модуле данных fmDM, осуществляется связь приложения с базой данных. Данный модуль является не визуальным и не доступен пользователю.

Описание модулей и соответствующих им форм приведено в таблице 23.

Таблица 23 – Описание модулей программы

Наименование модуля	Наименование формы	Описание
fmMain	Main	Отображает главную форму приложения

fmDM	DM	Модуль взаимодействия с базой данных
fmDoljn	Doljn	Отображает форму приложения с таблицами, содержащими информацию о должностях
fmAddDoljn	AddDoljn	Отображает форму добавления данных в таблицу «DOLJN»
fmEditDoljn	EditDoljn	Отображает форму изменения данных в таблице «DOLJN»
fmSotrud	Sotrud	Отображает форму приложения с таблицами, содержащими информацию о сотрудниках
fmAddSotrud	AddSotrud	Отображает форму добавления данных в таблицу «SOTRUD»
fmEditSotrud	EditSotrud	Отображает форму изменения данных в таблице «SOTRUD»
fmOtdelSht	OtdelSht	Отображает форму приложения с таблицами, содержащими информацию об отделах и штатном расписании предприятия
fmAddOtdel	AddOtdel	Отображает форму добавления данных в таблицу «OTDEL»
fmEditOtdel	EditOtdel	Отображает форму изменения данных в таблице «OTDEL»
Наименование модуля	Наименование формы	Описание
fmAddSht	AddSht	Отображает форму добавления данных в таблицу «SHT»

fmEditSht	EditSht	Отображает форму изменения данных в таблице «SHT»
fmZapros1	Zapros1	Осуществляет выполнение запроса о выводе списка должностей по алфавиту
fmZapros2	Zapros2	Осуществляет выполнение запроса о выводе списка должностей указанного отдела
fmZapros3	Zapros3	Осуществляет выполнение запроса о выводе списка должностей с окладом в указанном диапазоне
fmOtchet2	Otchet2	Осуществляет формирование отчета о штатном расписании указанного отдела
fmOtchet3	Otchet3	Осуществляет формирование отчета о списке сотрудников указанного отдела

4.5 Методы испытаний

Для проверки корректности работы программного обеспечения необходимо в адресной строке браузера ввести localhost:3000, после чего откроется веб-сайт. Результат запуска приведён на рисунке приложения Б.1. Следует выполнить тестовые примеры и сопоставить их результаты с фактической работой программы. Если результаты совпадут, программа считается работоспособной.

Разработаны следующие тестовые примеры для проверки функционирования программы:

Тестовый пример 1. Проверка функций учета информации об отделах.

Следует проверить корректность добавления записи в таблицу «Отдел». Для этого нужно кликнуть мышью по пункту «Отдел». Откроется вкладка «Отдел» (Рисунок Б.2). На этой вкладке необходимо нажать кнопку «Добавить отдел». Откроется форма «Добавление записи». В этой форме заполнить поле «Название отдела» значением «IT отдел» (Рисунок Б.3). Затем нажать кнопку «Сохранить». Запись должна быть успешно внесена в таблицу «Отдел».

Тестовый пример 2. Проверка функций учета информации о должностях.

Необходимо проверить добавление записи в таблицу «Должности». Для этого выбрать пункт «Должности». Откроется вкладка «Должности» (Рисунок Б.4). Нажать кнопку «Добавить должность». Откроется форма добавления записи, в которой заполнить поля «Название должности» – «Программист» и «Оклад» – «80000» (Рисунок Б.5). Затем сохранить запись. Она должна появиться в таблице «Должности».

Тестовый пример 3. Проверка функций учета информации о сотрудниках.

Следует проверить добавление записи в таблицу «Сотрудники». Для этого выбрать пункт «Сотрудники». Откроется вкладка «Сотрудники» (Рисунок Б.6). Нажать кнопку «Добавить сотрудника». В форме «Добавление записи» заполнить поля «ФИО сотрудника» – «Иванов И.И.», «Адрес» – «г. Пенза, ул. Московская, 10» (Рисунок Б.7). Нажать кнопку «Сохранить». Запись должна быть добавлена в таблицу «Сотрудники».

Тестовый пример 4. Проверка функций учета информации о штатном расписании.

Проверить добавление записи в таблицу «Отпуск». Выбрать пункт «Отпуск». Откроется вкладка «Отпуск» (Рисунок Б.10). Нажать кнопку «Добавить отпуск». В форме заполнить поля: «Отдел» – «IT отдел», «Должность» – «Программист», «Сотрудник» – «Иванов И.И.» (Рисунок Б.9). Затем сохранить запись. Она должна появиться в таблице «Штатное расписание».

Тестовый пример 5. Проверка управления отпусками.

Проверить добавление записи в таблицу «Должности». Выбрать пункт «Штатное расписание». Откроется вкладка «Штатное расписание» (Рисунок Б.8). Нажать кнопку «Добавить запись». Заполнить форму (Рисунок Б.11) и сохранить. Запись должна быть внесена в таблицу «Отпуск». Далее проверить обновление записей в таблице «Отпуск» по диаграмме Ганта — после добавления убедиться в наличии записи (Рисунок Б.12).

Тестовый пример 6. Проверка выполнения запросов к базе данных.

Выбрать пункт «Отчеты». Откроется вкладка «Отчеты и запросы» (Рисунок Б.13). Для проверки получения ответов на запросы пользователя выбрать меню «Запросы». Из выпадающего списка выбрать «Список должностей по алфавиту». Откроется форма с соответствующим списком (Рисунок Б.14). Далее выбрать «Список должностей указанного отдела» (Рисунок Б.13). Откроется форма с результатом (Рисунок Б.15). Затем выбрать «Список должностей с окладом в указанном диапазоне» (Рисунок Б.13). Откроется форма с результатами (Рисунок Б.16).

Тестовый пример 7. Проверка формирования отчетных документов.

Для проверки выбрать меню «Отчеты» (Рисунок Б.13), затем подменю «Отделы по алфавиту». Сформированный отчет представлен на рисунке Б.17. Далее выбрать подменю «Штатное расписание отдела». Отчет представлен на рисунке Б.18. Затем выбрать подменю «Список сотрудников отдела». Отчет представлен на рисунке Б.19.

Тестовый пример 8. Проверка корректности ввода данных пользователем в поля форм.

Если программа выполняет все функции в соответствии с тестовыми примерами, то она работает верно.

5 Оценка затрат на разработку и внедрение программного средства

5.1 Организация производства разработки программного средства

Нахождение себестоимости продукции – одна из важнейших частей процесса производства товара. Она включается в себя стоимость оборудования, заработную плату и необходимых условий для сотрудников, лицензионные продукты для эффективной разработки продукции, сравнение с существующими продуктами и много другое.

В состав нахождения себестоимости входит ее планирование, основными целями которого является:

- определение ожидаемых затрат на производство и реализацию;
- контроль расходов, благодаря которому можно понять какие затраты можно уменьшить, либо отказаться от них. В это число входит расходы на трудовые и накладные затраты;
- анализ эффективности – сравнение фактических с запланированными затратами, понимание окупается ли продукт.

Для определения стоимости создания специализированного программного обеспечения по управлению персоналом необходимо учесть следующие расходы:

- аренда и обслуживание офисного пространства;
- покупка технического оснащения;
- приобретение лицензий на софт;
- расходные материалы;
- фонд оплаты труда;
- страховые отчисления;
- эксплуатационные расходы.

Разработка системы велась в собственном офисе площадью 35 кв.м. Месячные затраты на содержание помещения – 25000 руб., а за весь период создания продукта (5 месяцев) общая сумма составила 100000 руб.

Для реализации проекта потребовалось закупить:

- компьютерную технику;
- лицензионные программы;
- вспомогательные материалы.

Требования к рабочей станции:

- Центральный процессор: Intel Core i5 (или аналог);
- Количество ядер: не менее 8;
- Оперативная память: минимум 16 ГБ;
- Графический адаптер: NVIDIA GTX 1050 (или эквивалент);
- Накопитель: SSD от 256 ГБ;
- операционная система – Windows 10;
- графический драйвер (Nvidia) – 496.76.

Исходя из минимальных требований к оборудованию был собран компьютер на базе Intel Core i5, оперативной памятью 32 ГБ, видеокартой GeForce GTX 1060 и графическим драйвером версии 576.52. Остальные параметры соответствуют минимальным требованиям.

Данные об оборудовании представлены в таблице 24.

На основании данных из таблицы 21 затраты на оборудование в период разработки модели составили 144700 рублей.

Таблица 24 – Потребность в оборудовании

Комплектующие	Источник	Количество	Цена за штуку, рублей	Энергопотребление, Ватт
Системный блок на Intel Core i7	ООО «Компьютер Маркет»	1	90000	700

Комплектуемые	Источник	Количество	Цена за штуку, рублей	Энергопотребление, Ватт
Клавиатура HyperX Alloy FPS	ООО «Электрон Сервис»	1	3300	1,5
Мышь Razer DeathAdder	ООО «Техно Профи»	1	11200	1,6
Монитор Samsung Odyssey G5	ООО «Неттехн ика»	2	20100	26

Для создания и поддержки программного обеспечения, предназначенного для управления штатным расписанием компании, было закуплено лицензионное ПО. Информация о необходимости приобретения лицензионных продуктов представлена в таблице 25.

Таблица 25 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена, рублей	Срок действия лицензии, дней
Microsoft Windows 10 Pro	Официальный реселлер ООО «Технопрофиль»	16000	бессрочная
Антивирусное ПО Dr.Web Security Space	Сайт производителя URL: https://www.drweb.ru/	2200	365
Пакет офисных приложений «МойОфис»	ООО «Новые облачные технологии»	5800	бессрочная

На основании таблицы 25 затраты на лицензионное программное обеспечение (в расчете на 5 месяца разработки) составят 34200 рублей.

На основании таблицы 27 затраты на дополнительные материалы составят 35190 рублей.

Разработка программного обеспечения для предприятия осуществлялась командой разработчиков. Состав команда в зависимости от поставленных задач может быть разный. Минимальный набор специалистов для разработки модели: руководитель проекта, инженер по архитектуре сети, разработчик модели, тестировщик.

Для разработки приобретены дополнительные материалы. Перечень дополнительных материалов представлен в таблице 26.

Таблица 26 – Потребность в дополнительных материалах

Материал	Источник	Количество, штук	Ориентировочная цена за единицу, рублей
Компьютерный стол	ООО «Интернет Решения»	1	2680
Кресло компьютерное	ООО «Интернет Решения»	1	16970
Лампа настольная с регулировкой яркости LED	ООО «Интернет Решения»	1	2290
Принтер Pantum 6500	ООО «Интернет Решения»	1	11750
Канцтовары	ООО «Интернет Решения»	1	1500

Состав коллектива разработчиков модели представлен в таблице 27.

Таблица 27 – Состав коллектива разработчиков

Специалист	Количество, человек	Основные задачи
Руководитель проекта	1	Контроль за разработкой проекта, набор участников команды разработчиков, регулирование финансовых отношений
Проектировщик	1	Создание диаграмм классов и взаимодействий, проектирование базы данных
Программист	2	Реализация компонентов системы, написание и отладка кода, модульное тестирование, документирование кода
Тестировщик	1	Анализ полученных данных, выбор стратегии тестирования, выявление ошибок и недоработок, аргументированные сведения по улучшению работы модели

Заработная плата сотрудников может состоять из двух частей:

- основная заработная плата – обязательная, письменно установленная часть занятости специалистов в процессе разработки модели (в днях);
- дополнительная заработная плата – не обязательная часть заработной платы, в состав которой может входить: премия, оклад за часы переработки и другие.

Данные по основной заработной и дополнительной плате разработчиков программного средства управления штатным расписанием предприятия приведены в таблице 28.

Таблица 28 – Затраты на основную и дополнительную заработные платы

Специалист	Численность работников, чел	Размер оплаты труда на одного сотрудника, руб.	Длительность занятости, дней.	Основная заработная плата на одного работника, руб.	Дополнительные выплаты на одного сотрудника, руб.	Всего, рублей
Руководитель проекта	1	150000	100	750000	112500	862500
Проектировщик	1	90000	60	270000	40500	310500
Программист	2	120000	100	600000	90000	1380000
Тестировщик	1	80000	40	160000	24000	184000
Итого	5	660000	400	1780000	267000	2737000

Согласно таблице 29, затраты на заработную плату разработчиков составят 2737000 рублей.

Таблица 29 – Сравнение программного обеспечения и традиционных методов

№	Наименование	Программное обеспечение, рублей	Традиционные методы, рублей
1	Покупка необходимого оборудования	144700	210000
2	Настройка компонентов	70000	120000
3	Обслуживание	35000	80000
4	Потери при сбоях	5000	25000
5	Обучение сотрудников	30000	45000
6	Итого	284700	480000

Согласно таблице 29, затраты на заработную плату разработчиков составят 2737000 рублей.

Не маловажной частью является изучение существующих аналог на рынке труда. Реализация программного средства управления штатным расписанием предприятия может быть как нецелесообразна, ввиду существования менее инновационных, но более экономичных средства, так и экономична, ввиду их отсутствия.

Исходя из таблицы 29 можно заметить, что разработка и сопровождение программного средства управления штатным расписанием предприятия экономичнее, чем традиционные варианты. Реализация целесообразна, а общая сумма выгоды составила 195300 рублей.

5.2 Расчет налогового обложения сотрудников

В соответствии с положениями статьи 419 Налогового кодекса Российской Федерации, работодатель несёт обязательство по уплате страховых взносов за своих работников в государственные внебюджетные фонды. Расчёт и перечисление данных взносов регламентируется нормами части второй НК РФ, утверждённой Федеральным законом №117-ФЗ от 05 августа 2000 года (в редакции от 29 мая 2024 года), в частности положениями раздела XI, посвящённого вопросам обязательных страховых начислений.

Установленный единый страховой тариф применяется ко всем плательщикам, осуществляющим выплаты в пользу физических лиц. В состав обязательных направлений, на которые распространяется данный тариф, входят:

- система обязательного пенсионного страхования (ОПС);
- обязательное социальное страхование в случае временной нетрудоспособности и в связи с материнством (Фонд социального страхования РФ);
- обязательное медицинское страхование (ОМС);
- другие отчисления, предусмотренные действующими нормативами в рамках системы государственного социального обеспечения.

До достижения установленного предельного размера базы для начисления страховых взносов применяется совокупная ставка 30% от суммы выплат

работникам. При превышении этого порога страховые взносы уплачиваются по пониженному тарифу, составляющему 15,1% [5].

Суммарные отчисления на страховые взносы составят 821100 рублей.

Общая сумма затрат на разработку продукта составляет 3872190 рублей.

5.3 Оценка эффективности программного средства

Оценка необходимости и рациональности внедрения инвестиционного проекта базируется на анализе его предполагаемой экономической эффективности. Под эффективностью в данном контексте следует понимать соответствие запланированных затрат тем результатам, которые ожидаются на этапе проектирования и реализации системы. Разработка и интеграция специализированного программного обеспечения для автоматизации управления штатным расписанием предприятия позволяют достичь ряда значимых преимуществ, способствующих улучшению качества администрирования и оптимизации внутренних процессов. В частности, можно выделить следующие положительные эффекты:

- оперативное формирование и корректировка штатного расписания с учетом организационных изменений без необходимости ручного пересчета должностных единиц;
- повышение прозрачности структуры предприятия, благодаря автоматическому отображению штатных позиций, вакантных мест и связей между подразделениями;
- снижение вероятности ошибок при расчётах фонда оплаты труда, за счёт встроенных алгоритмов, учитывающих надбавки, коэффициенты и нормативные ограничения;
- облегчение планирования численности персонала, что способствует точной оценке кадровых потребностей в разрезе подразделений и временных периодов;
- автоматическое формирование отчетности для бухгалтерии, отдела кадров и руководства по установленным формам без дублирования данных;

- повышение согласованности работы между отделами, участвующими в планировании и изменении штатной структуры (кадровая служба, экономика труда, финансы);
- снижение временных затрат на подготовку документов при изменении штатного расписания, благодаря готовым шаблонам и автоматизированным операциям согласования;
- ускорение процедур согласования и утверждения изменений, за счёт цифрового маршрута согласования, минимизирующего бумажный документооборот.

5.4 Анализ рисков при разработке и внедрении программного средства

На каждом этапе реализации проекта существуют риски, которые могут частично или полностью приостановить его или повлиять на сроки и качество исполнения. Основные риски на этапах создания программного средства управления штатным расписанием предприятия:

- плохая обратная связь между руководителем проекта и клиентом. Любая недостоверная или не вовремя преподнесенная информация способна навредить проекту, что влечет за собой откладывание сроков его сдачи;
- риски, связанные со временем – плохой менеджмент времени как со стороны руководства, так и со стороны разработчиков;
- технические риски – риски, связанные с проблемами оборудования, неправильно настройкой или несовместимости между друг другом;
- информационные риски – риски, связанные с недостоверной информацией или источниками;
- риски, связанные со сменой сотрудников. Ключевые разработчики проекта, обладающие полной информацией, могут уволиться, что может повлиять на сроки проекта;
- ресурсные риски;
- потеря мотивации;

— использование не лицензионного ПО.

Все вышеперечисленные проблемы могут появиться на любом этапе разработки. Во избежание и минимизации неприятностей необходимо заранее предусмотреть все альтернативные исходы, на каждом этапе копии проекта, а также проводить тестирование ключевых узлов. При плохой работоспособности сотрудников необходимо давать больше времени на изучение научного материала и поощрять членов команды, проявивших успехи в разработке.

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		53

6 Безопасность жизнедеятельности

Правила техники безопасности регламентируются государством с помощью нормативных документов. Они необходимы чтобы минимизировать вредное воздействие на организм человека при различных видах работ. Для профессии инженера-программиста вредными факторами являются: сидячая работа и воздействие компьютера на организм человека. Из нормативных документов можно выделить:

- санитарно-гигиенические правила и нормативы СанПин 2.2.2/2.4.1340-03 [6];
- трудовой кодекс;
- межгосударственный стандарт ГОСТ 12.0.003–2015, определяющий и классифицирующий основные вредные и опасные производственные факторы.

6.1 Требование к рабочему месту инженера-разработчика

Длительное пребывание сотрудника за рабочим столом в течение всего трудового дня оказывает комплексное воздействие на организм, особенно при выполнении обязанностей, связанных с использованием компьютера. Существенная статическая нагрузка ложится на опорно-двигательный аппарат, в частности на мышцы спины и шеи, а также на зрительный анализатор. При отсутствии регламентированных перерывов и несоблюдении принципов эргономики рабочего места возрастает риск развития неблагоприятных изменений в состоянии здоровья: возможны ухудшение остроты зрения, деформация осанки, искривление позвоночного столба, а также снижение качества периферического кровообращения.

Для того чтобы облегчить или минимизировать негативное воздействие необходимо следовать следующим правилам:

- расстояние глаз от монитора должно составлять от 50 до 70 см;
- положение монитора должно находиться ниже уровня глаз;

- необходим монитор, соответствующий современным технологиям, во избежание плохой работы матрицы;
- положение спинки кресла должно находиться в таком положении, чтобы плечи были расслаблены, а спина имела ровную осанку без чувства болей в спине и пояснице;
- руки должны находиться на столе или ручках кресла и образовывать угол в 90 градусов;
- ноги полной стопой должны находиться на земле, в случаях, когда ноги не достают до земли, необходимо использовать подставку;
- ногам под столом ничего не должно мешать;
- локти и кисти рук должны находиться на одной высоте;
- рабочее пространство должно быть хорошо освещено и свободно.

По мимо вышесказанных правил каждые 30–40 минут необходимо выполнять разминку для ног, спины, глаз и всего тела в целом. Для большего комфорта глаз можно использовать специальные очки с насечками.

Согласно санитарно-эпидемиологическим требованиям к условиям труда на рабочем месте, предназначенном для работы в положении сидя, производственное оборудование и рабочие столы должны иметь пространство для размещения ног высотой не менее 600 мм, глубиной – не менее 450 мм на уровне колен и 600 мм на уровне стоп, шириной не менее 500 мм.

6.2 Требования к микроклимату в рабочем помещении

Одним из значимых факторов, влияющих на продуктивность сотрудников в рамках должностных обязанностей, закреплённых в штатном расписании, является уровень фонового шума в рабочем пространстве. В соответствии с положениями санитарных норм, изложенных в СанПиН 2.2.3670–20 «Санитарно-эпидемиологические требования к условиям труда» [7], допустимое значение эквивалентного звукового давления в офисной среде не должно превышать 50 дБА.

Систематическое превышение данного показателя в течение рабочей смены может привести к объективному снижению качества условий труда. Более того, при длительном воздействии повышенного уровня шума возможно развитие хронической усталости, раздражительности и ухудшения функционального состояния нервной системы, что в конечном счёте негативно сказывается на здоровье сотрудников. Ключевые факторы присущие негативному влиянию:

- снижается концентрация внимания;
- повышается усталость;
- снижается работоспособность сотрудника.

Для того, чтобы снизить уровень шума в помещении необходимо использовать шумоизоляцию стен и окон. Пиковой корректированной нормой уровня звука является 127 дБА.

Один из ключевых аспектов правильного обустройства рабочего места сотрудника является освещение. Требования к свету в помещении осуществляются по формуле:

Площадь одного рабочего места должно быть не менее 5 квадратных метров. Для лучшего освещения комнаты необходимо иметь 7 ламп, расположенных на равном расстоянии друг от друга. Экраны мониторов должны быть развернуты к освещению таким образом, чтобы избежать попадания на экраны мониторов прямых солнечных лучей, а норма составляла 300–500 люкс. Допускается установка жалюзи и разделение включения световых приборов.

Температура в офисном помещении зависит от времени года. В холодное время года температура должна варьироваться в пределах от 22 °С до 23 °С. В теплое время года – от 22 °С до 25 °С. При этом допустимый диапазон температур в холодное время года 19–24 °С, а в теплое время года 20–29 °С [8].

Для регулировки температуры в особо жарких или холодных помещениях необходимо иметь кондиционеры с регулировкой температуры и направления воздуха. Требования к влажности помещения могут варьироваться от 55% до 65% при температуре около +20 °С. В иных случаях существует возможность

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

установить увлажнителей воздуха, которые необходимо периодически пополнять водой. Умеренно-влажный воздух благотворно влияет на иммунитет и состояние сотрудников.

6.3 Техника безопасности при работе с персональным компьютером

В процессе формирования штатного расписания предприятия, особенно при проектировании должностей, связанных с использованием персональных компьютеров, необходимо учитывать требования охраны труда, изложенные в Типовой инструкции по безопасности труда при работе на ЭВМ (ТОИ Р-45-084-01), утверждённой приказом Минсвязи РФ №162 от 02.07.2001 г. [9].

Согласно данной инструкции, рабочие операции, выполняемые с применением средств вычислительной техники, подразделяются на три категории, каждая из которых требует соблюдения определённого режима труда и отдыха:

- первая категория (А) включает должности, ориентированные на чтение информации с экрана монитора. Для предотвращения зрительного и психоэмоционального переутомления работникам данной категории следует предоставлять два регламентированных перерыва по 15 минут, не включая перерыв на обед;

- вторая категория (Б) охватывает штатные единицы, деятельность которых связана преимущественно с набором текстов и обработкой документов. Здесь предусматриваются 10-минутные перерывы после каждого полного часа работы;

- третья категория (В) включает творческие и интеллектуально сложные виды труда (например, проектные, аналитические или инженерные должности), в рамках которых рекомендуется устанавливать 15-минутные перерывы после каждого часа работы за компьютером.

Для безопасной работы необходимо соблюдать ряд правил нахождения на рабочем месте:

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

- техническое задание;
- соблюдать требования по работе с ПК;
- соблюдать правила обращения с рабочей техникой;
- соблюдать температурный режим;
- прослушать инструктаж по технике безопасности;
- иметь памятку о правилах пожарной безопасности;
- перед началом каждого рабочего дня проверять основные узлы и агрегаты;
- производить самоконтроль медицинского состояния.

Благодаря вышеперечисленным правилам осуществляется корректная работа с персональным компьютером.

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

Заключение

В ходе выполнения ВКР разработано программное средство для хранения, редактирования и обработки информации о штатной структуре предприятия с использованием базы данных и графического интерфейса.

В ВКР проанализирована предметная область, на основании чего сформулированы и реализованы требования к функциональности создаваемого приложения.

Структура базы данных спроектирована с учётом логики хранения информации о сотрудниках, отделах, должностях и начислении окладов. Для реализации физического уровня хранения данных использована реляционная СУБД PostgreSQL, а клиентская и серверная части программного средства разработаны с использованием современных JavaScript-технологий – React и Node.js.

В процессе разработки обеспечено выполнение всех основных операций, включая добавление, редактирование, удаление и поиск записей, а также формирование отчётности по заданным критериям.

Реализован графический интерфейс, позволяющий пользователю эффективно взаимодействовать с системой.

Дальнейшая разработка программного обеспечения включает в себя добавление искусственного интеллекта и машинного обучения.

Работа программного средства была протестирована. Результаты тестирования подтвердили его корректность, стабильность и соответствие заданной функциональности.

Список использованных источников

1. Базы данных [Электронный ресурс] – Режим доступа: URL – <https://itglobal.com/ru-ru/company/blog/bazy-dannyh-kakie-byvayut-i-v-chem-osobennosti-kazhdogo-vida/> (дата обращения 25.05.2025).

2. Системы управления базами данных [Электронный ресурс] – Режим доступа: URL – <https://kuzneck.1cbit.ru/bazy-dannykh-i-subd/> (дата обращения 25.05.2025).

3. Еременко А.В., Долгова И.А., Щербаков С.В. Базы данных. Учебник. - Пенза: Издательство ПГУ, 210. - 110 с.

4. Долгова И.А., Еременко А.В., Измайлова М.В., Строганов Д.В., Терякова Ю.В. Использование OpenModelSphere для разработки информационных систем. - Пенза: Издательство ПГУ, 2014. - 84 с.

5. «Налоговый кодекс Российской Федерации (часть вторая)» от 05.08.2000 N 117-ФЗ (ред. от 25.05.2025) (с изм. и доп., вступ. в силу с 01.06.2024).

6. СанПиН 2.2.2/2.4.1340-03. Гигиенические требования к персональным электронно-вычислительным машинам и организации работы (с изм. вступ. в силу с 21.06.2016).

7. СанПиН 2.2.3670–20. Санитарно-эпидемиологические требования к условиям труда. Изд-во стандартов, 2012. – 47с.

8. Температурный режим в помещениях [Электронный ресурс] – URL - <https://cgon.rosпотреbnadzor.ru/naseleniyu/gramotnyy-potrebitel/temperaturnyi-rezim-v-pomeshheniyax-zilyx-i-obshhestvennyx-zdanii/> (дата обращения 25.05.2025).

9. Типовая инструкция по охране труда при работе на персональном компьютере. – 2001. – 3 с.

Приложение А

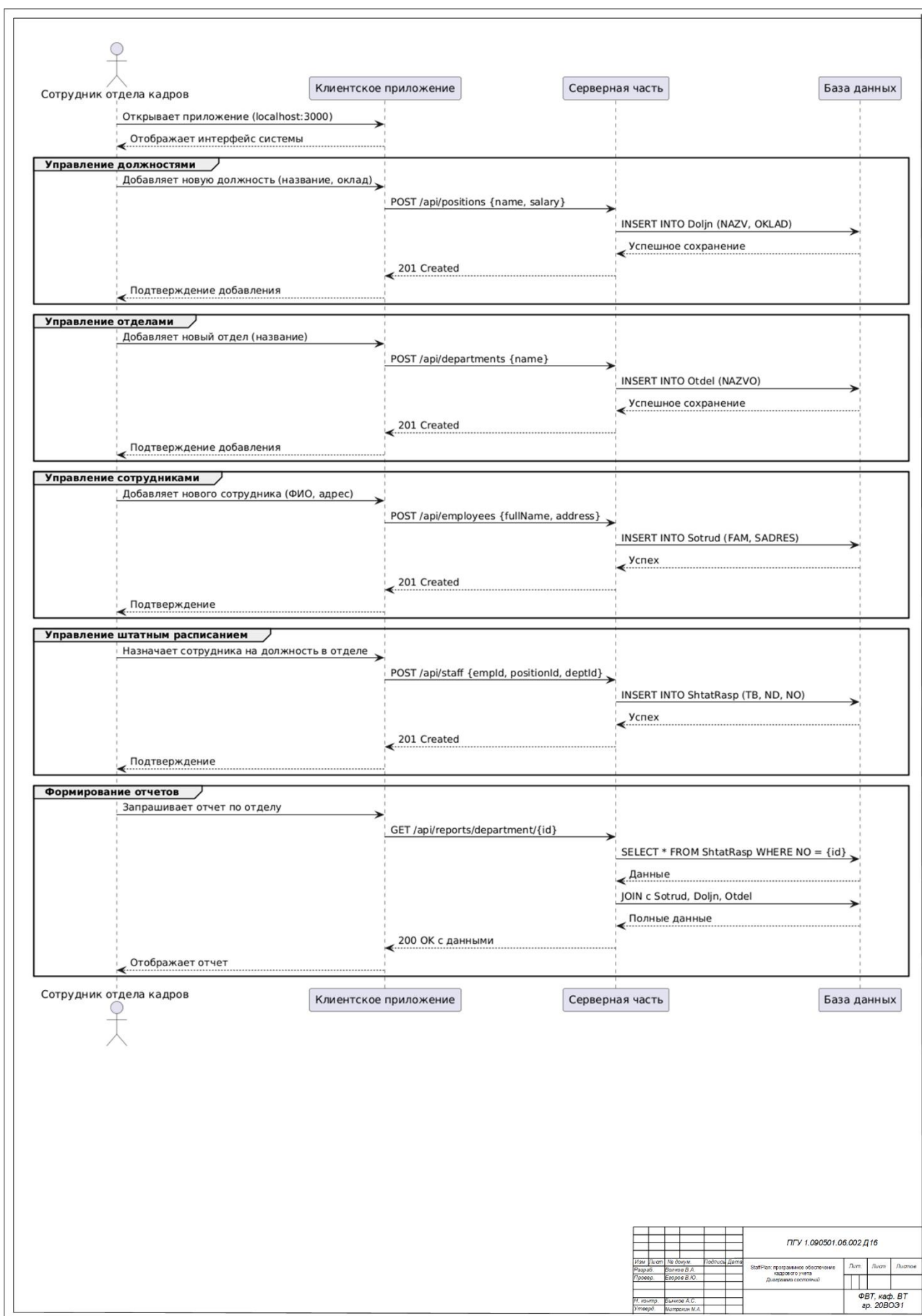
(Обязательное)

StaffPlan: программное обеспечение кадрового учета

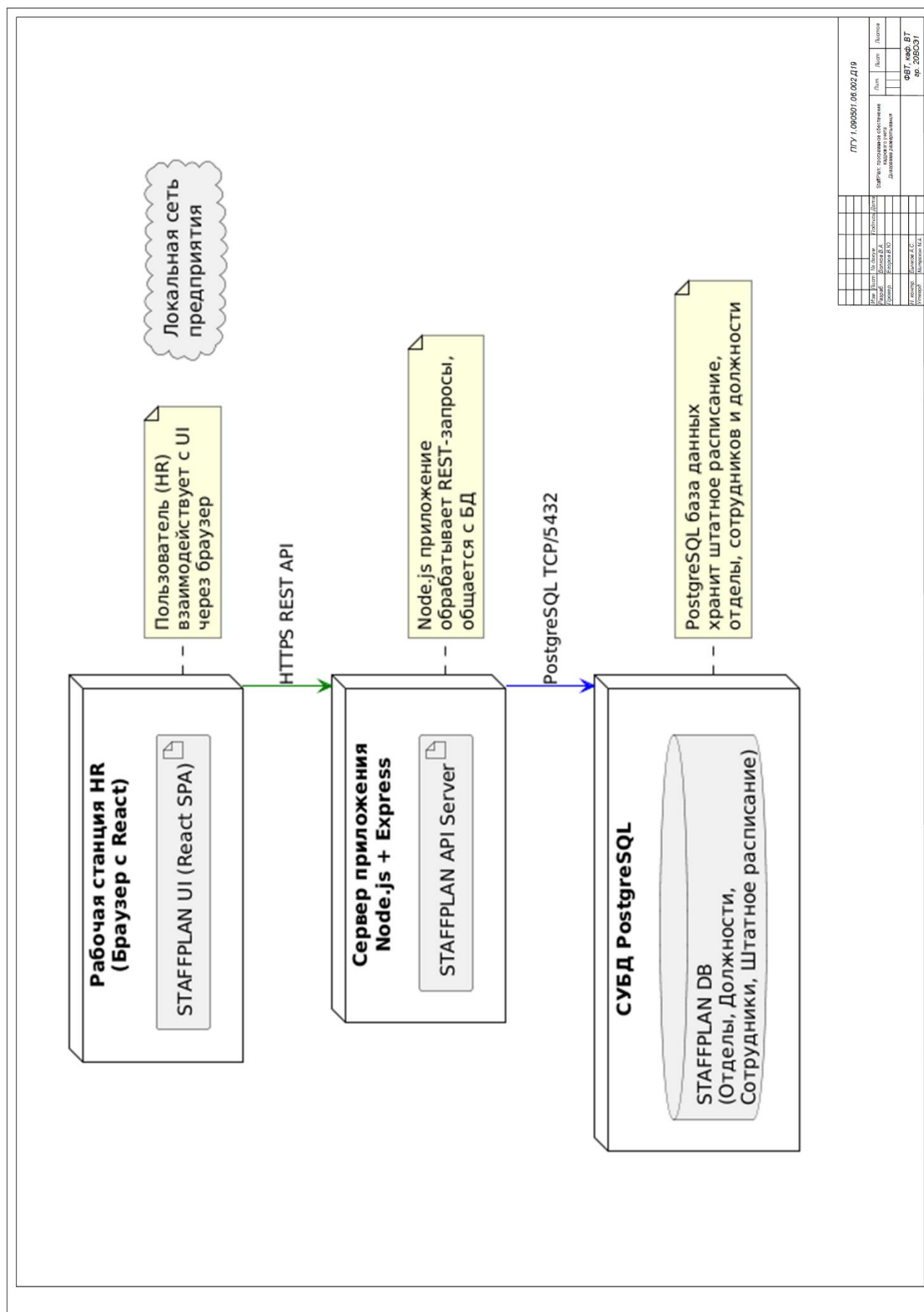
UML-диаграммы

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

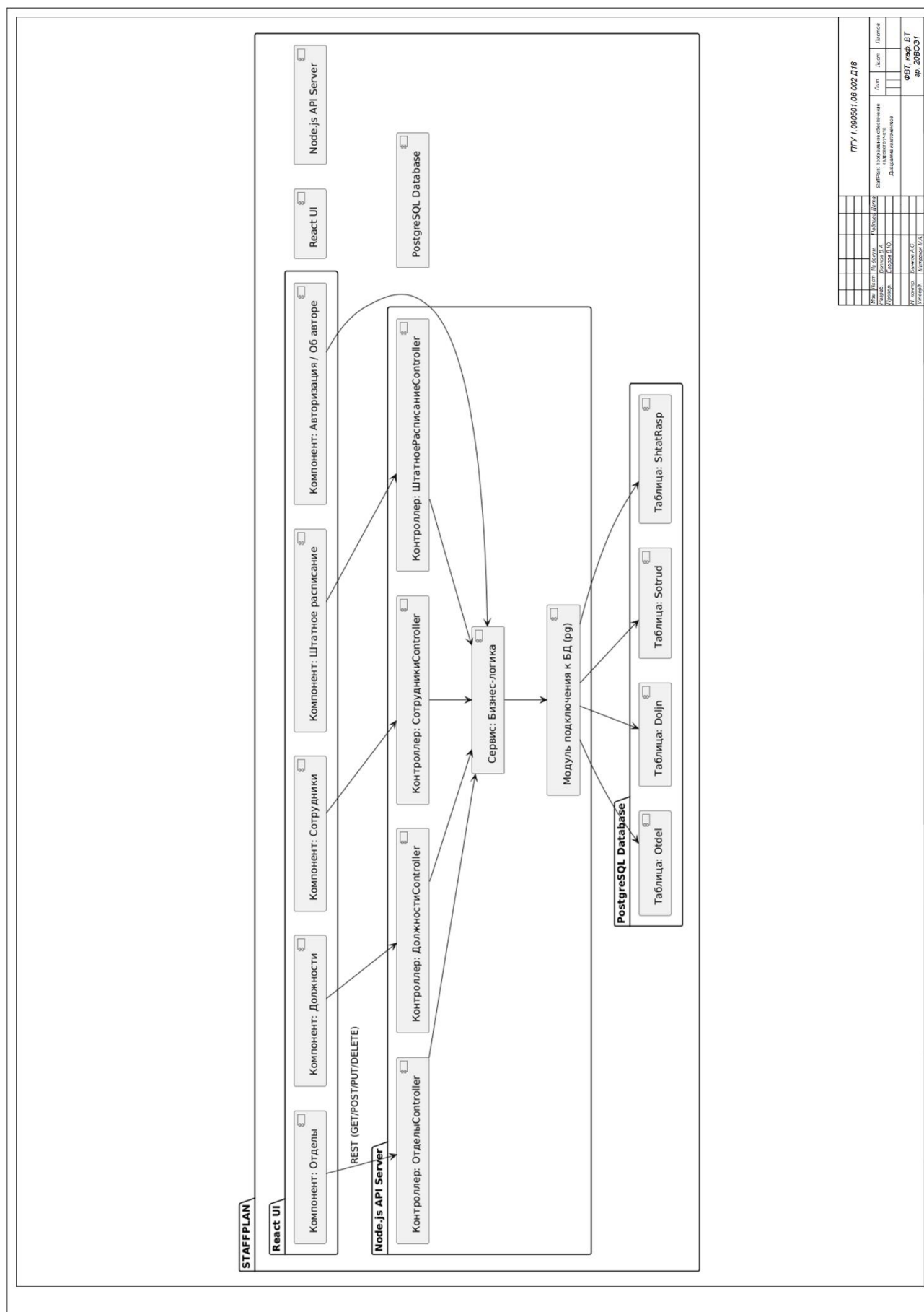
Приложение А.1-Диаграмма последовательностей



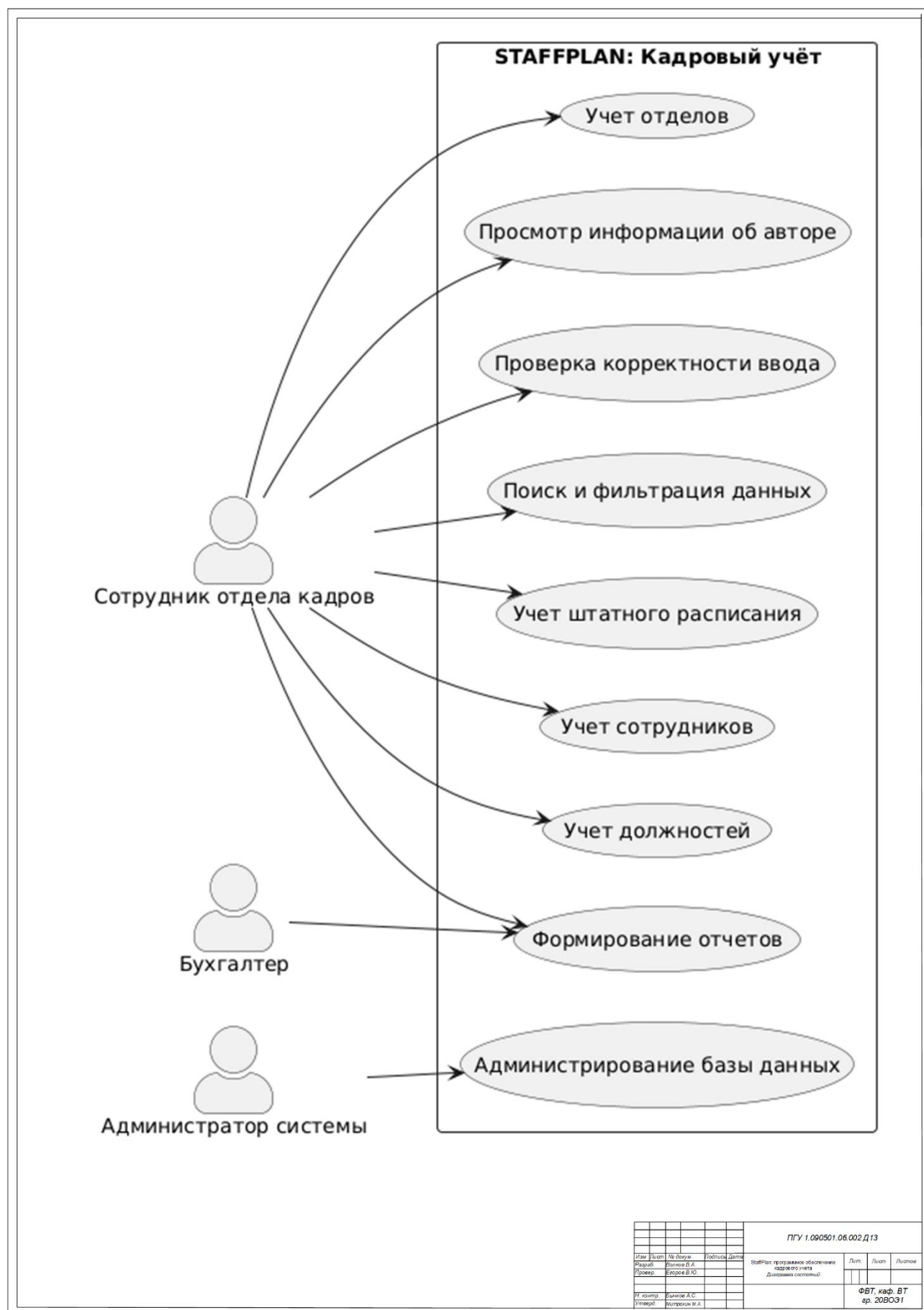
Приложение А.2-Диаграмма развертывания



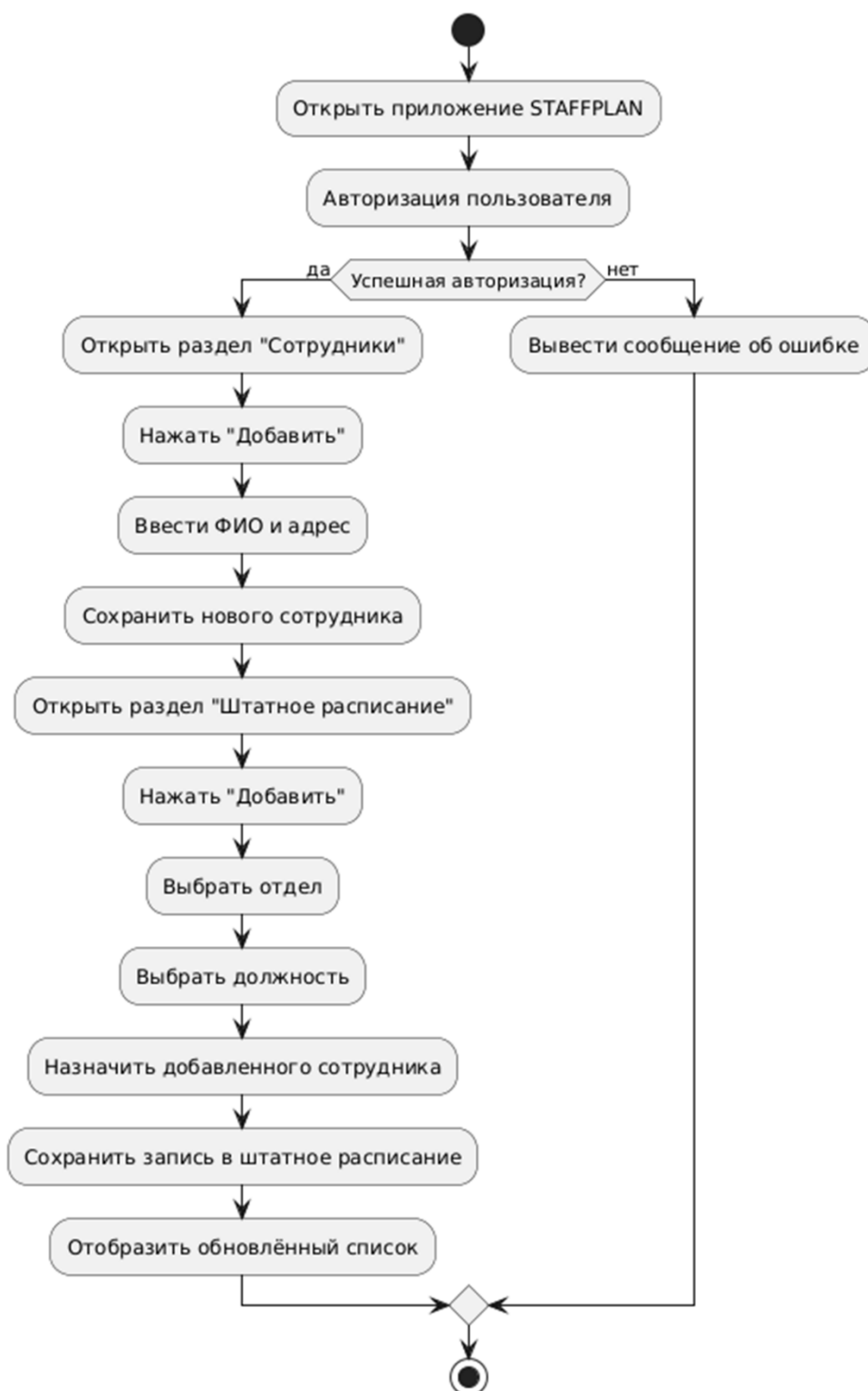
Приложение А.3-Диаграмма компонентов



Приложение А.4-Диаграмма вариантов использования



Приложение А.5-Диаграмма деятельности



					ПГУ 1.090501.06.002 Д17				
Изм.	Лист	№ докум.	Подпись		Дата	StaffPlan: программное обеспечение кадрового учета Департамент деятельности	Лит.	Лист	Лист
Разраб.		Волова В.А.							
Провер.		Евгоров В.Ю.							
И. контр.		Булганов А.С.				ФБТ, каф. ВТ ар. 20В031			
Утверд.									

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.06.001 ПЗ

Лист

66

Приложение Б

(Обязательное)

StaffPlan: программное обеспечение кадрового учета

Результаты функционального тестирования

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

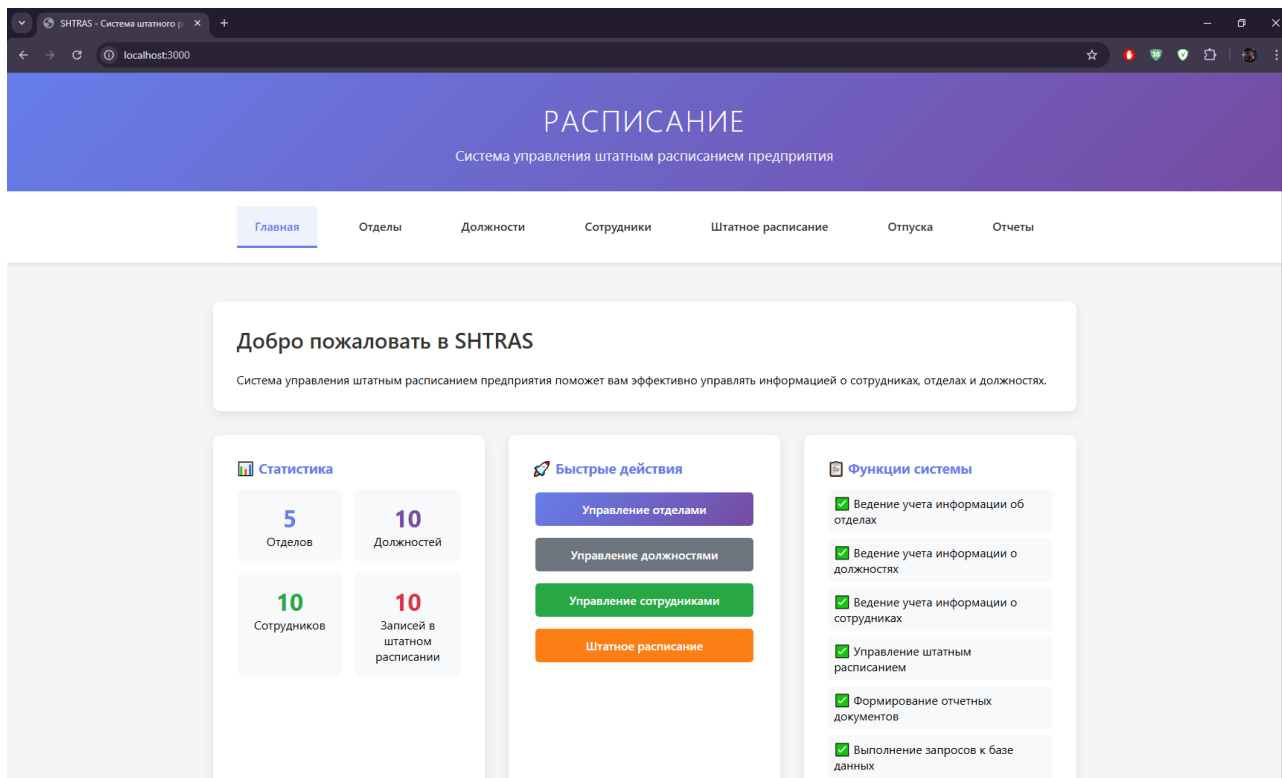


Рисунок Б.1 – Главная страница сайта

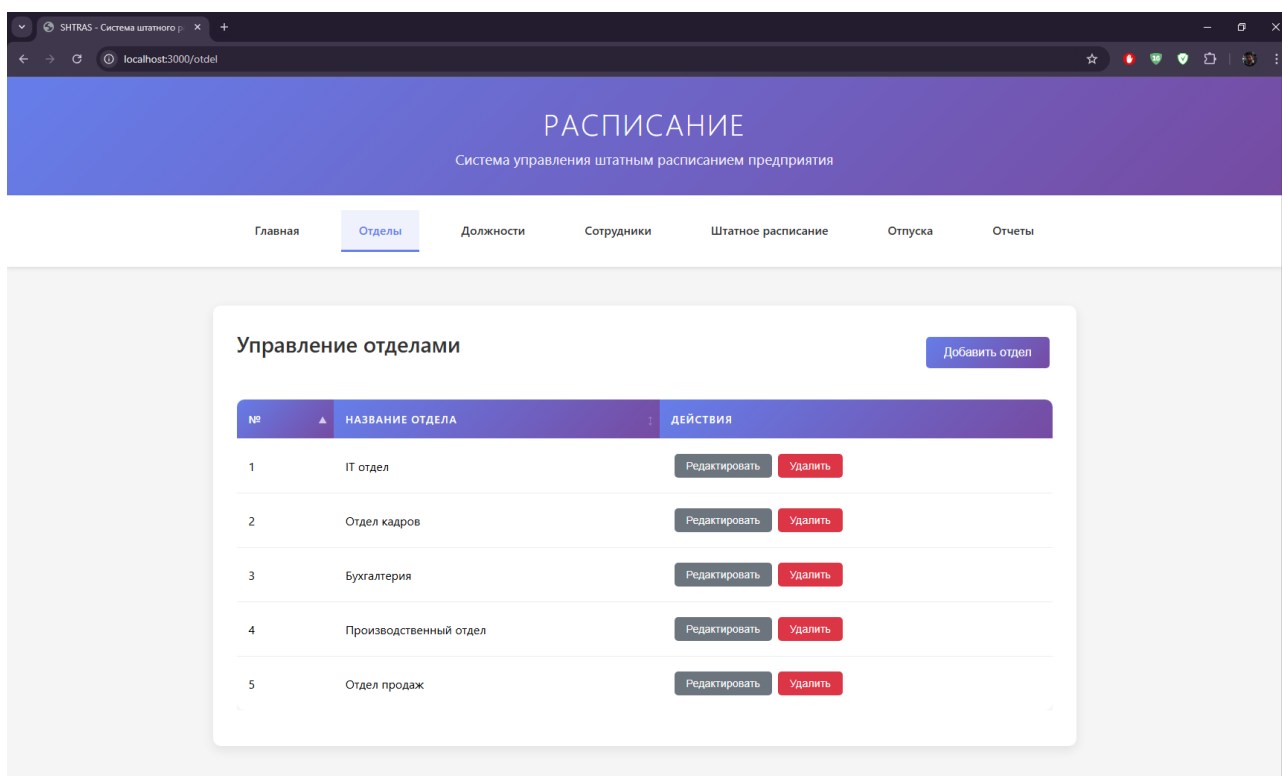


Рисунок Б.2 – Страница вкладки отделы

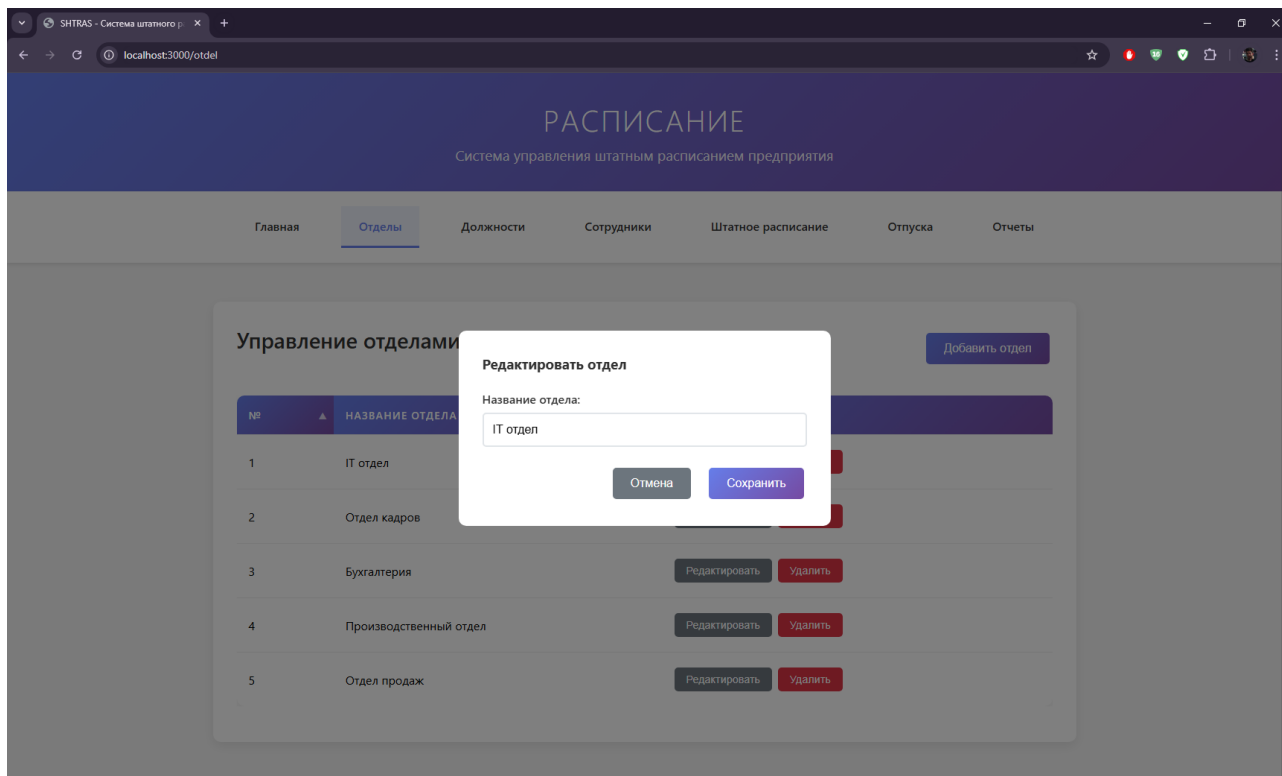


Рисунок Б.3 – Страница добавления и редактирования отдела

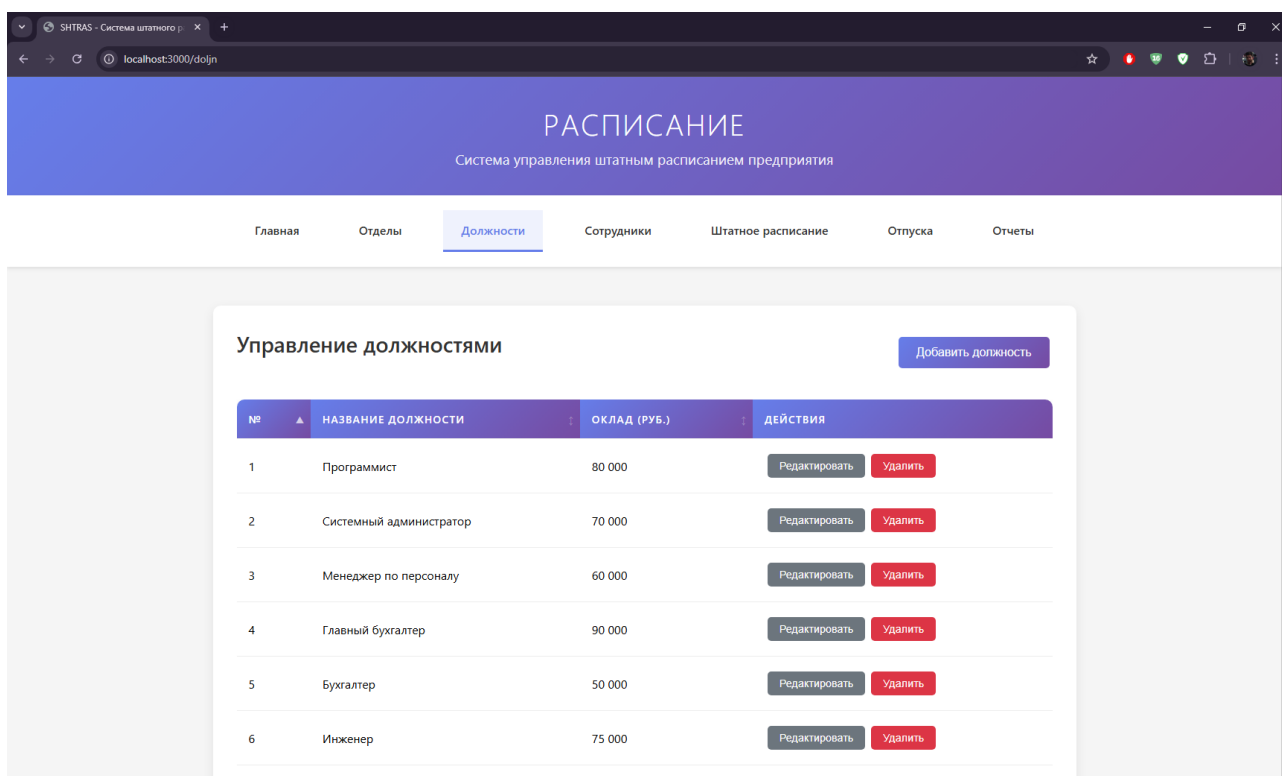


Рисунок Б.4 – Страница вкладки должности

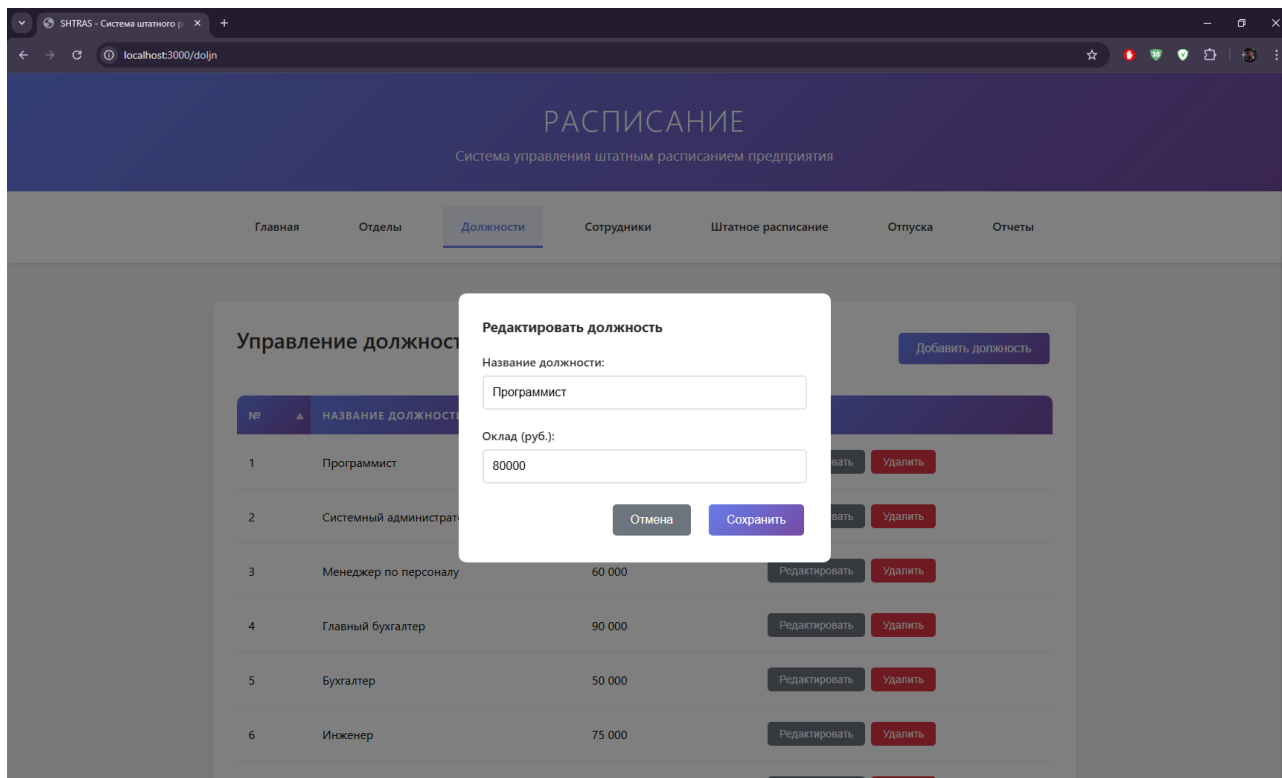


Рисунок Б.5 – Страница добавления и редактирования должности

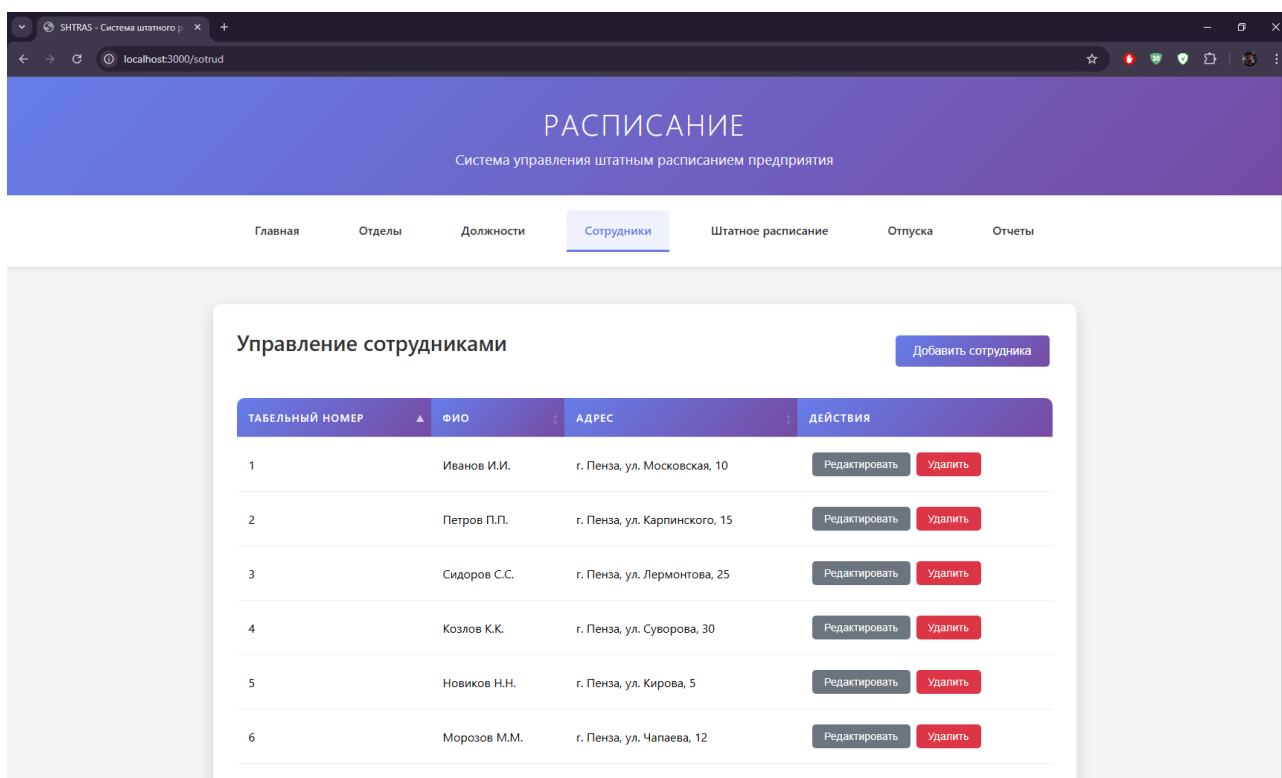


Рисунок Б.6 – Страница вкладки сотрудники

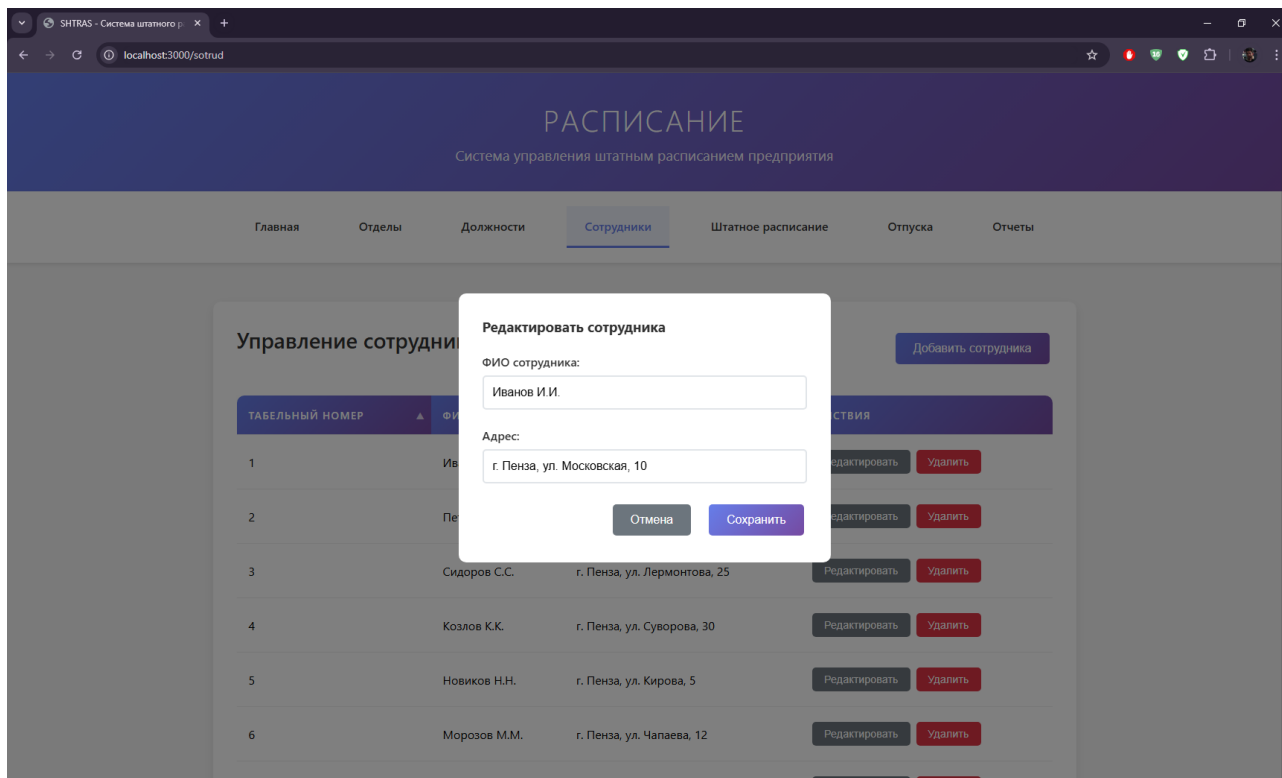


Рисунок Б.7 – Страница добавления и редактирования сотрудников

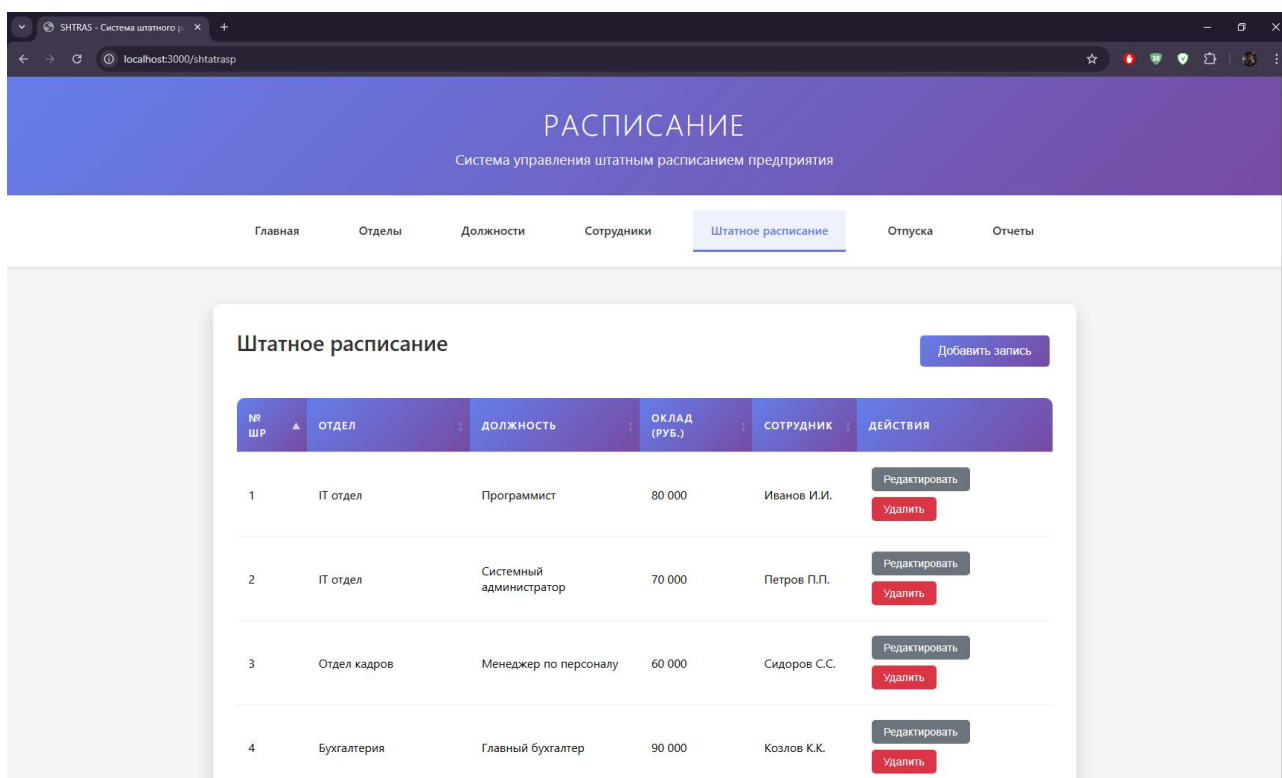


Рисунок Б.8 – Страница вкладки штатное расписание

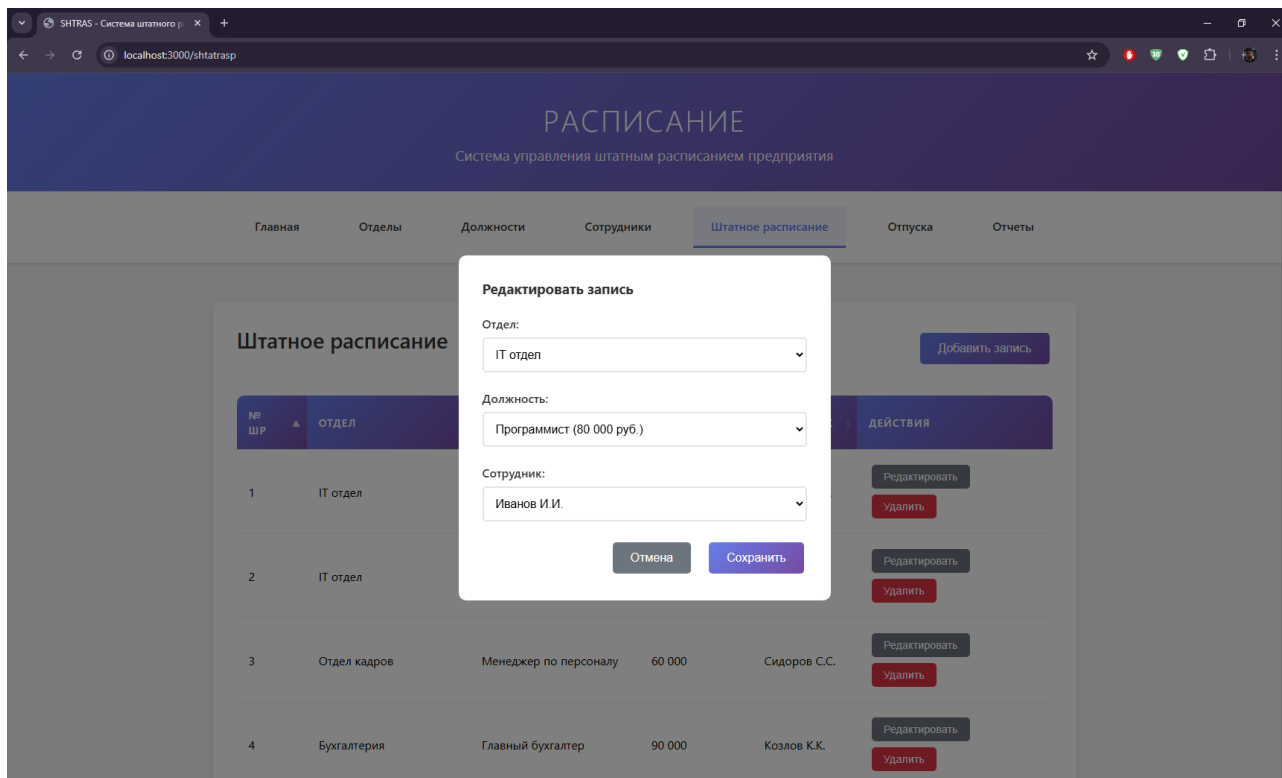


Рисунок Б.9 – Страница добавления и редактирования штатного расписания

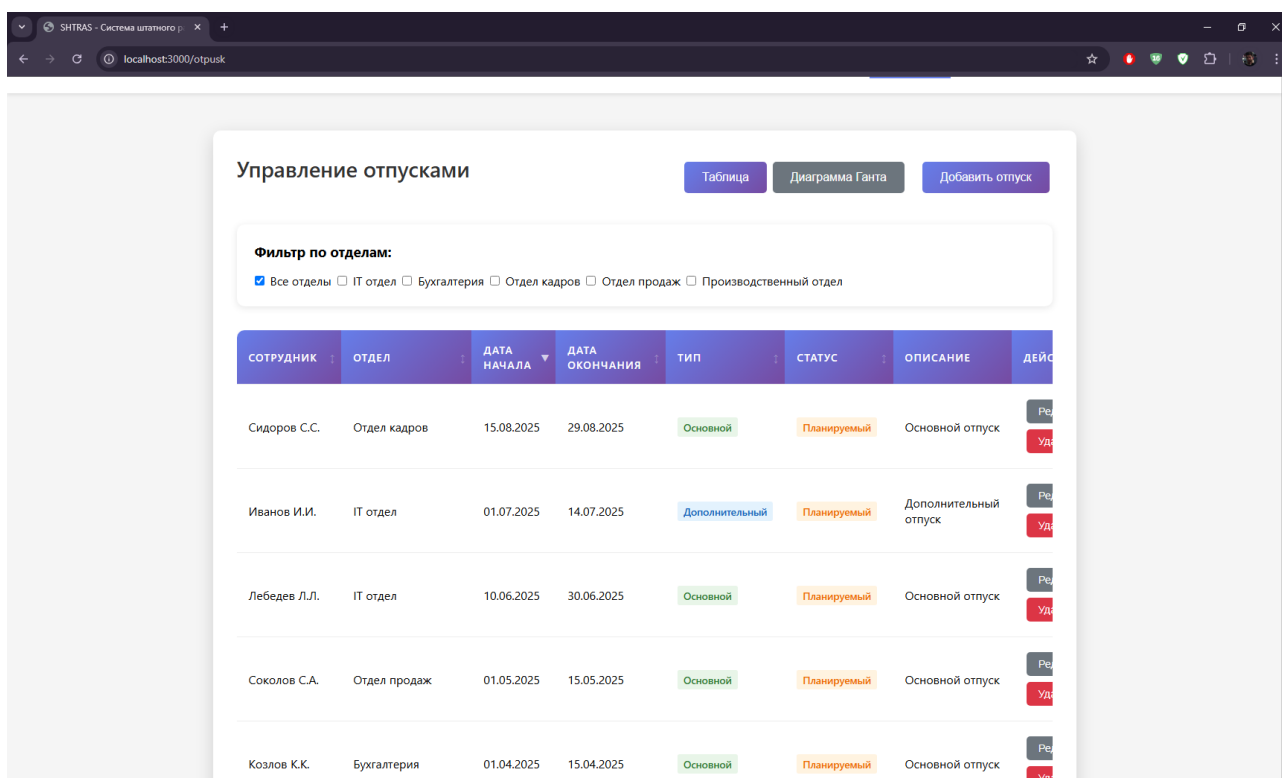


Рисунок Б.10 – Страница вкладки управление отпусками

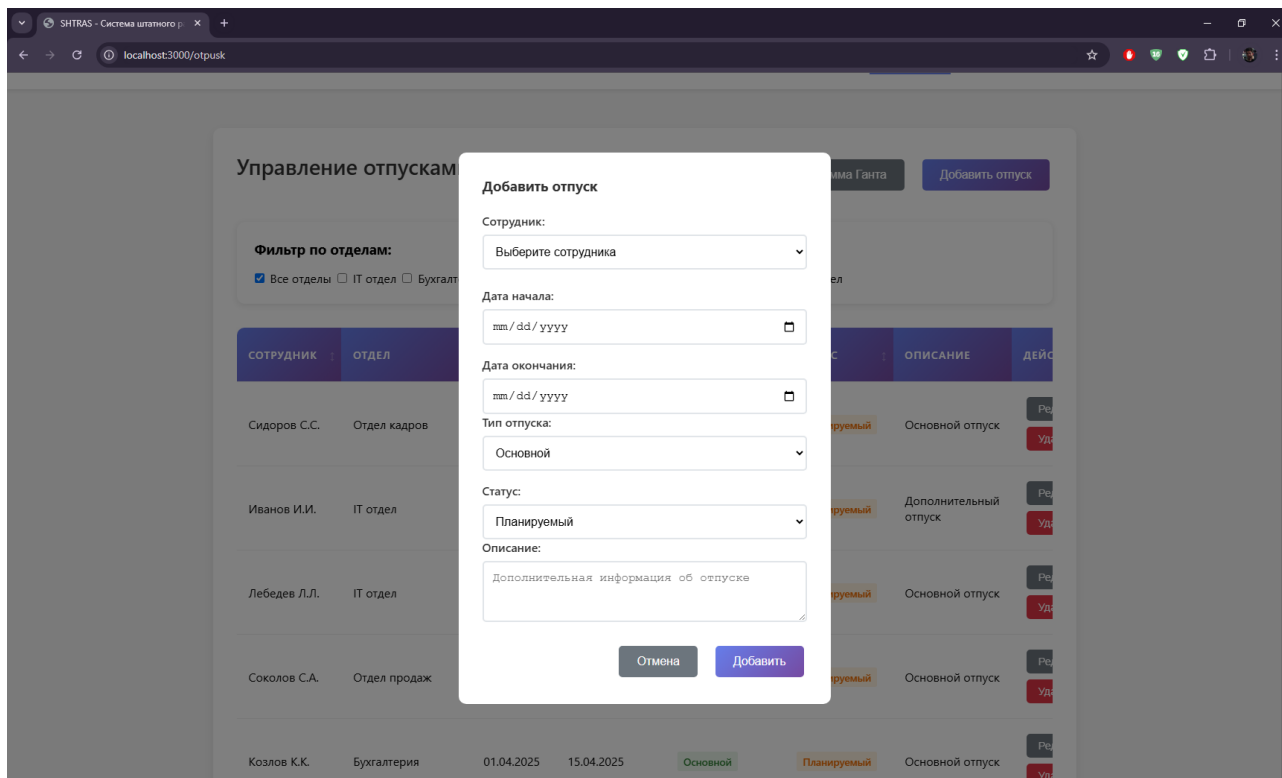


Рисунок Б.11 – Страница вкладки добавления и редактирования отпусков

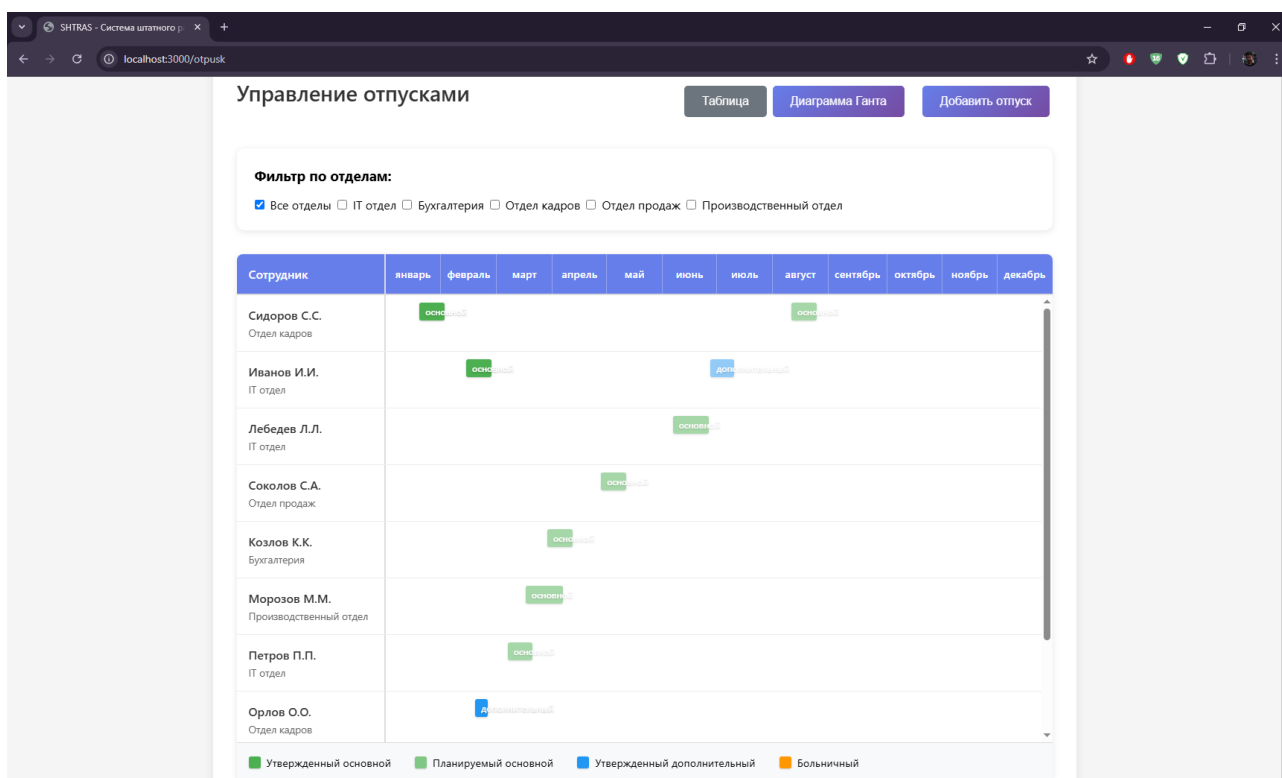


Рисунок Б.12 – Страница вкладки диаграммы Ганта

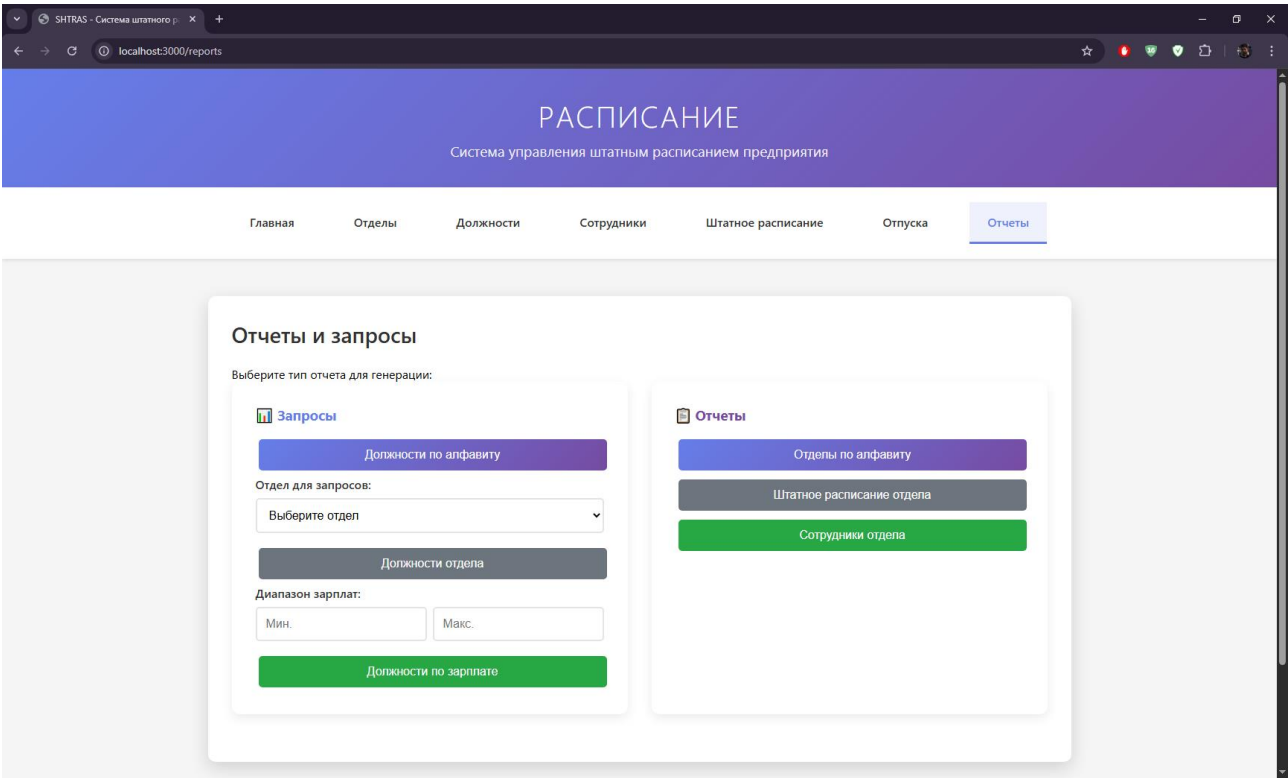


Рисунок Б.13 – Страница вкладки отчеты

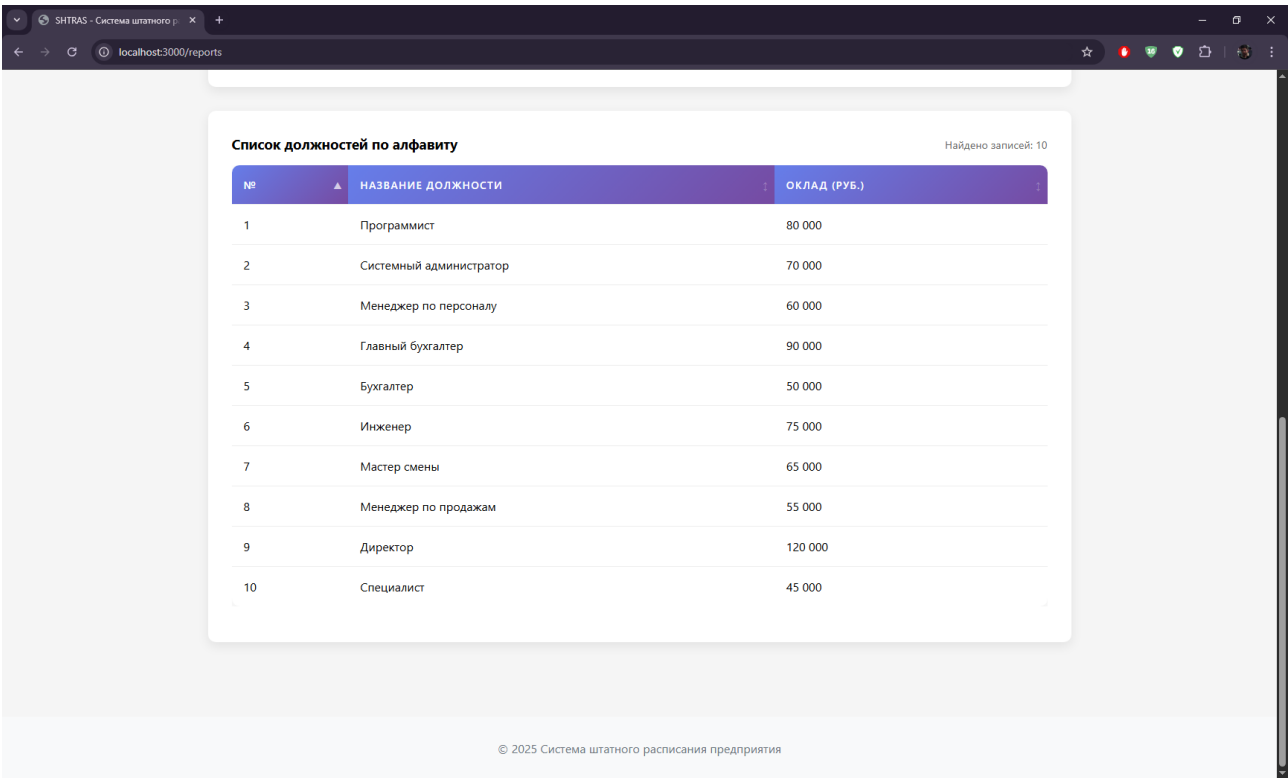


Рисунок Б.14 – Результат работы запроса вывода списка должностей по алфавиту

SHTRAS - Система штатного

localhost:3000/reports

Отчеты и запросы

Выберите тип отчета для генерации:

Запросы

Должности по алфавиту

Отдел для запросов:

IT отдел

Должности отдела

Диапазон зарплат:

Мин.

Макс.

Должности по зарплате

Отчеты

Отделы по алфавиту

Штатное расписание отдела

Сотрудники отдела

Список должностей указанного отдела

Найдено записей: 3

№	НАЗВАНИЕ ДОЛЖНОСТИ	ОКЛАД (РУБ.)	ОТДЕЛ
1	Программист	80 000	IT отдел
2	Системный администратор	70 000	IT отдел
9	Директор	120 000	IT отдел

Рисунок Б.15 – Результат работы запроса вывода списка должностей указанного отдела

SHTRAS - Система штатного

localhost:3000/reports

Отчеты и запросы

Выберите тип отчета для генерации:

Запросы

Должности по алфавиту

Отдел для запросов:

IT отдел

Должности отдела

Диапазон зарплат:

100000

500000

Должности по зарплате

Отчеты

Отделы по алфавиту

Штатное расписание отдела

Сотрудники отдела

Список должностей с окладом в указанном диапазоне

Найдено записей: 1

№	НАЗВАНИЕ ДОЛЖНОСТИ	ОКЛАД (РУБ.)
9	Директор	120 000

Рисунок Б.16 – Результат работы запроса вывода списка должностей указанного отдела с диапазоном зарплат

Запросы

Должности по алфавиту

Отдел для запросов:

IT отдел

Должности отдела

Диапазон зарплат:

100000

500000

Должности по зарплате

Отчеты

Отделы по алфавиту

Штатное расписание отдела

Сотрудники отдела

Список отделов по алфавиту

Найдено записей: 5

№	НАЗВАНИЕ ОТДЕЛА
1	IT отдел
2	Отдел кадров
3	Бухгалтерия
4	Производственный отдел
5	Отдел продаж

Рисунок Б.17 – Результат работы вывода отчета отделов по алфавиту

Запросы

Должности по алфавиту

Отдел для запросов:

IT отдел

Должности отдела

Диапазон зарплат:

100000

500000

Должности по зарплате

Отчеты

Отделы по алфавиту

Штатное расписание отдела

Сотрудники отдела

Штатное расписание указанного отдела

Найдено записей: 3

№ ШР	ОТДЕЛ	ДОЛЖНОСТЬ	ОКЛАД (РУБ.)	СОТРУДНИК
1	IT отдел	Программист	80 000	Иванов И.И.
2	IT отдел	Системный администратор	70 000	Петров П.П.
9	IT отдел	Директор	120 000	Лебедев Л.Л.

Рисунок Б.18 – Результат работы вывода отчета штатного расписания отдела

SHTRAS - Система штатного расписания

localhost:3000/reports

Отчеты и запросы

Выберите тип отчета для генерации:

Запросы

Должности по алфавиту

Отдел для запросов:

ИТ отдел

Должности отдела

Диапазон зарплат:

100000

500000

Должности по зарплате

Отчеты

Отделы по алфавиту

Штатное расписание отдела

Сотрудники отдела

Список сотрудников указанного отдела

Найдено записей: 3

ТАБЕЛЬНЫЙ НОМЕР	ФИО	АДРЕС	ОТДЕЛ
1	Иванов И.И.	г. Пенза, ул. Московская, 10	ИТ отдел
2	Петров П.П.	г. Пенза, ул. Карпинского, 15	ИТ отдел
9	Лебедев Л.Л.	г. Пенза, ул. Маяковского, 18	ИТ отдел

Рисунок Б.19 – Результат работы вывода отчета сотрудников отдела

Приложение В

(обязательное)

StaffPlan: программное обеспечение кадрового учета

Листинг программы

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

Файл index.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const { Pool } = require('pg');

const app = express();
const port = process.env.PORT || 5000;

// Настройка middleware
app.use(cors());
app.use(express.json());

// Настройка подключения к БД
const pool = new Pool({
  host: process.env.DB_HOST || 'localhost',
  port: process.env.DB_PORT || 5432,
  database: process.env.DB_NAME || 'shtras_db',
  user: process.env.DB_USER || 'shtras_user',
  password: process.env.DB_PASSWORD || 'shtras_password',
});

// Проверка подключения к БД
pool.connect((err, client, release) => {
  if (err) {
    console.error('Ошибка подключения к БД:', err);
  } else {
    console.log('Успешное подключение к БД');
    release();
  }
});

// API Routes

// ОТДЕЛЫ
// Получить все отделы
app.get('/api/otdel', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM otdel ORDER BY nazvo');
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении отделов:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Создать отдел
app.post('/api/otdel', async (req, res) => {
  try {
    const { nazvo } = req.body;
    const result = await pool.query(
```

```

    'INSERT INTO otdel (nazvo) VALUES ($1) RETURNING *',
    [nazvo]
  );
  res.status(201).json(result.rows[0]);
} catch (err) {
  console.error('Ошибка при создании отдела:', err);
  res.status(500).json({ error: 'Ошибка сервера' });
}
});

// Обновить отдел
app.put('/api/otdel/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { nazvo } = req.body;
    const result = await pool.query(
      'UPDATE otdel SET nazvo = $1 WHERE no = $2 RETURNING *',
      [nazvo, id]
    );
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Отдел не найден' });
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при обновлении отдела:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Удалить отдел
app.delete('/api/otdel/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const result = await pool.query('DELETE FROM otdel WHERE no = $1 RETURNING *', [id]);
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Отдел не найден' });
    }
    res.json({ message: 'Отдел удален' });
  } catch (err) {
    console.error('Ошибка при удалении отдела:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// ДОЛЖНОСТИ
// Получить все должности
app.get('/api/doljn', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM doljn ORDER BY nazv');
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении должностей:', err);
  }
});

```

```

    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Создать должность
app.post('/api/doljn', async (req, res) => {
  try {
    const { nazv, oklad } = req.body;
    const result = await pool.query(
      'INSERT INTO doljn (nazv, oklad) VALUES ($1, $2) RETURNING *',
      [nazv, oklad]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при создании должности:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Обновить должность
app.put('/api/doljn/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { nazv, oklad } = req.body;
    const result = await pool.query(
      'UPDATE doljn SET nazv = $1, oklad = $2 WHERE nd = $3 RETURNING *',
      [nazv, oklad, id]
    );
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Должность не найдена' });
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при обновлении должности:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Удалить должность
app.delete('/api/doljn/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const result = await pool.query('DELETE FROM doljn WHERE nd = $1 RETURNING *', [id]);
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Должность не найдена' });
    }
    res.json({ message: 'Должность удалена' });
  } catch (err) {
    console.error('Ошибка при удалении должности:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

```

```

// СОТРУДНИКИ
// Получить всех сотрудников
app.get('/api/sotrud', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM sotrud ORDER BY fam');
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении сотрудников:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Создать сотрудника
app.post('/api/sotrud', async (req, res) => {
  try {
    const { fam, sadres } = req.body;
    const result = await pool.query(
      'INSERT INTO sotrud (fam, sadres) VALUES ($1, $2) RETURNING *',
      [fam, sadres]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при создании сотрудника:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Обновить сотрудника
app.put('/api/sotrud/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { fam, sadres } = req.body;
    const result = await pool.query(
      'UPDATE sotrud SET fam = $1, sadres = $2 WHERE tb = $3 RETURNING *',
      [fam, sadres, id]
    );
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Сотрудник не найден' });
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при обновлении сотрудника:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Удалить сотрудника
app.delete('/api/sotrud/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const result = await pool.query('DELETE FROM sotrud WHERE tb = $1 RETURNING *', [id]);
  }
});

```

```

    if (result.rows.length === 0) {
        return res.status(404).json({ error: 'Сотрудник не найден' });
    }
    res.json({ message: 'Сотрудник удален' });
} catch (err) {
    console.error('Ошибка при удалении сотрудника:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
}
});

// ШТАТНОЕ РАСПИСАНИЕ
// Получить штатное расписание
app.get('/api/shtatrasp', async (req, res) => {
    try {
        const result = await pool.query(`
            SELECT
                s.nsh, s.no, s.nd, s.tb,
                o.nazvo as otдел_name,
                d.nazv as dolzn_name, d.oklad,
                st.fam, st.sadres
            FROM shtatrasp s
            LEFT JOIN otдел o ON s.no = o.no
            LEFT JOIN doljn d ON s.nd = d.nd
            LEFT JOIN sotrud st ON s.tb = st.tb
            ORDER BY o.nazvo, d.nazv
        `);
        res.json(result.rows);
    } catch (err) {
        console.error('Ошибка при получении штатного расписания:', err);
        res.status(500).json({ error: 'Ошибка сервера' });
    }
});

// Создать запись штатного расписания
app.post('/api/shtatrasp', async (req, res) => {
    try {
        const { no, nd, tb } = req.body;
        const result = await pool.query(
            'INSERT INTO shtatrasp (no, nd, tb) VALUES ($1, $2, $3) RETURNING *',
            [no, nd, tb]
        );
        res.status(201).json(result.rows[0]);
    } catch (err) {
        console.error('Ошибка при создании записи штатного расписания:', err);
        res.status(500).json({ error: 'Ошибка сервера' });
    }
});

// Обновить запись штатного расписания
app.put('/api/shtatrasp/:nsh/:no', async (req, res) => {
    try {
        const { nsh, no } = req.params;

```

```

const { nd, tb } = req.body;
const result = await pool.query(
  'UPDATE shtatrasp SET nd = $1, tb = $2 WHERE nsh = $3 AND no = $4 RETURNING *',
  [nd, tb, nsh, no]
);
if (result.rows.length === 0) {
  return res.status(404).json({ error: 'Запись штатного расписания не найдена' });
}
res.json(result.rows[0]);
} catch (err) {
  console.error('Ошибка при обновлении записи штатного расписания:', err);
  res.status(500).json({ error: 'Ошибка сервера' });
}
});

```

// Удалить запись штатного расписания

```

app.delete('/api/shtatrasp/:nsh/:no', async (req, res) => {
  try {
    const { nsh, no } = req.params;
    const result = await pool.query(
      'DELETE FROM shtatrasp WHERE nsh = $1 AND no = $2 RETURNING *',
      [nsh, no]
    );
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Запись штатного расписания не найдена' });
    }
    res.json({ message: 'Запись штатного расписания удалена' });
  } catch (err) {
    console.error('Ошибка при удалении записи штатного расписания:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

```

// ОТПУСКА

// Получить все отпуска с информацией о сотрудниках и отделах

```

app.get('/api/otpusk', async (req, res) => {
  try {
    const result = await pool.query(`
      SELECT
        o.id, o.tb, o.start_date, o.end_date, o.type, o.status, o.description,
        s.fam as employee_name,
        ot.nazvo as department_name,
        ot.no as department_id
      FROM otpusk o
      LEFT JOIN sotrud s ON o.tb = s.tb
      LEFT JOIN shtatrasp sr ON s.tb = sr.tb
      LEFT JOIN otdel ot ON sr.no = ot.no
      ORDER BY o.start_date DESC
    `);
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении отпусков:', err);
  }
});

```

```

    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Получить отпуска по отделам
app.get('/api/otpusk/by-departments', async (req, res) => {
  try {
    const { departments } = req.query; // Массив ID отделов
    let query = `
      SELECT
        o.id, o.tb, o.start_date, o.end_date, o.type, o.status, o.description,
        s.fam as employee_name,
        ot.nazvo as department_name,
        ot.no as department_id
      FROM otpusk o
      LEFT JOIN sotrud s ON o.tb = s.tb
      LEFT JOIN shtatrasp sr ON s.tb = sr.tb
      LEFT JOIN otdel ot ON sr.no = ot.no
    `;

    let params = [];
    if (departments && departments !== 'all') {
      const deptIds = departments.split(',').map(id => parseInt(id)).filter(id => !isNaN(id));
      if (deptIds.length > 0) {
        query += ` WHERE ot.no = ANY($1)`;
        params.push(deptIds);
      }
    }

    query += ` ORDER BY o.start_date`;

    const result = await pool.query(query, params);
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении отпусков по отделам:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Создать отпуск
app.post('/api/otpusk', async (req, res) => {
  try {
    const { tb, start_date, end_date, type, status, description } = req.body;
    const result = await pool.query(
      'INSERT INTO otpusk (tb, start_date, end_date, type, status, description) VALUES ($1, $2, $3, $4, $5, $6) RETURNING *',
      [tb, start_date, end_date, type, status, description]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при создании отпуска:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

```

```

    }
  });

// Обновить отпуск
app.put('/api/otpusk/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const { tb, start_date, end_date, type, status, description } = req.body;
    const result = await pool.query(
      'UPDATE otpusk SET tb = $1, start_date = $2, end_date = $3, type = $4, status = $5, description
= $6 WHERE id = $7 RETURNING *',
      [tb, start_date, end_date, type, status, description, id]
    );
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Отпуск не найден' });
    }
    res.json(result.rows[0]);
  } catch (err) {
    console.error('Ошибка при обновлении отпуска:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Удалить отпуск
app.delete('/api/otpusk/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const result = await pool.query('DELETE FROM otpusk WHERE id = $1 RETURNING *', [id]);
    if (result.rows.length === 0) {
      return res.status(404).json({ error: 'Отпуск не найден' });
    }
    res.json({ message: 'Отпуск удален' });
  } catch (err) {
    console.error('Ошибка при удалении отпуска:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// ОТЧЕТЫ И ЗАПРОСЫ
// Список должностей по алфавиту
app.get('/api/reports/positions-alphabetical', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM doljn ORDER BY nazv');
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении отчета должностей по алфавиту:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Список должностей указанного отдела
app.get('/api/reports/positions-by-department/:deptId', async (req, res) => {

```

```

try {
  const { deptId } = req.params;
  const result = await pool.query(`
    SELECT DISTINCT d.nd, d.nazv, d.oklad, o.nazvo as otдел_name
    FROM doljn d
    JOIN shtatrasp s ON d.nd = s.nd
    JOIN otдел o ON s.no = o.no
    WHERE o.no = $1
    ORDER BY d.nazv
  `, [deptId]);
  res.json(result.rows);
} catch (err) {
  console.error('Ошибка при получении должностей отдела:', err);
  res.status(500).json({ error: 'Ошибка сервера' });
}
});

// Список должностей с окладом в указанном диапазоне
app.get('/api/reports/positions-by-salary', async (req, res) => {
  try {
    const { minSalary = 0, maxSalary = 999999 } = req.query;
    const result = await pool.query(
      'SELECT * FROM doljn WHERE oklad BETWEEN $1 AND $2 ORDER BY oklad DESC',
      [minSalary, maxSalary]
    );
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении должностей по зарплате:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Список отделов по алфавиту
app.get('/api/reports/departments-alphabetical', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM otдел ORDER BY nazvo');
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении отчета отделов по алфавиту:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

// Штатное расписание указанного отдела
app.get('/api/reports/staff-schedule/:deptId', async (req, res) => {
  try {
    const { deptId } = req.params;
    const result = await pool.query(`
      SELECT
        s.nsh, o.nazvo as otдел_name,
        d.nazv as dolzn_name, d.oklad,
        st.fam, st.sadres
    `

```

```

FROM shtatrasp s
JOIN otдел o ON s.no = o.no
LEFT JOIN doljn d ON s.nd = d.nd
LEFT JOIN sotrud st ON s.tb = st.tb
WHERE o.no = $1
ORDER BY d.nazv, st.fam
`, [deptId]);
res.json(result.rows);
} catch (err) {
  console.error('Ошибка при получении штатного расписания отдела:', err);
  res.status(500).json({ error: 'Ошибка сервера' });
}
});

```

```

// Список сотрудников указанного отдела
app.get('/api/reports/employees-by-department/:deptId', async (req, res) => {
  try {
    const { deptId } = req.params;
    const result = await pool.query(`
      SELECT DISTINCT st.tb, st.fam, st.sadres, o.nazvo as otдел_name
      FROM sotrud st
      JOIN shtatrasp s ON st.tb = s.tb
      JOIN otдел o ON s.no = o.no
      WHERE o.no = $1
      ORDER BY st.fam
    `, [deptId]);
    res.json(result.rows);
  } catch (err) {
    console.error('Ошибка при получении сотрудников отдела:', err);
    res.status(500).json({ error: 'Ошибка сервера' });
  }
});

```

```

// Запуск сервера
app.listen(port, () => {
  console.log(`Сервер запущен на порту ${port}`);
});

```

Файл app.js

```

import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link, useLocation } from 'react-router-dom';
import Dashboard from './pages/Dashboard';
import Otdel from './pages/Otdel';
import Doljn from './pages/Doljn';
import Sotrud from './pages/Sotrud';
import ShtatRasp from './pages/ShtatRasp';
import Otpusk from './pages/Otpusk';
import Reports from './pages/Reports';

```

```

function Navigation() {
  const location = useLocation();

```

```
const isActive = (path) => location.pathname === path;
```

```
return (
  <nav className="nav">
    <div className="container">
      <ul className="nav-list">
        <li className="nav-item">
          <Link to="/" className={ `nav-link ${isActive('/') ? 'active' : ''} `}>
            Главная
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/otdel" className={ `nav-link ${isActive('/otdel') ? 'active' : ''} `}>
            Отделы
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/doljn" className={ `nav-link ${isActive('/doljn') ? 'active' : ''} `}>
            Должности
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/sotrud" className={ `nav-link ${isActive('/sotrud') ? 'active' : ''} `}>
            Сотрудники
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/shtatrasp" className={ `nav-link ${isActive('/shtatrasp') ? 'active' : ''} `}>
            Штатное расписание
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/otpusk" className={ `nav-link ${isActive('/otpusk') ? 'active' : ''} `}>
            Отпуска
          </Link>
        </li>
        <li className="nav-item">
          <Link to="/reports" className={ `nav-link ${isActive('/reports') ? 'active' : ''} `}>
            Отчеты
          </Link>
        </li>
      </ul>
    </div>
  </nav>
);
}
```

```
function App() {
  return (
    <Router>
      <div className="App">
```

```

<header className="header">
  <div className="container">
    <h1>РАСПИСАНИЕ</h1>
    <p style={{ textAlign: 'center', marginTop: '0.5rem', fontSize: '1.2rem', opacity: 0.9 }}>
      Система управления штатным расписанием предприятия
    </p>
  </div>
</header>

<Navigation />

<main className="container">
  <Routes>
    <Route path="/" element={<Dashboard />} />
    <Route path="/otdel" element={<Otdel />} />
    <Route path="/doljn" element={<Doljn />} />
    <Route path="/sotrud" element={<Sotrud />} />
    <Route path="/shtatrasp" element={<ShtatRasp />} />
    <Route path="/otpusk" element={<Otpusk />} />
    <Route path="/reports" element={<Reports />} />
  </Routes>
</main>

<footer style={{
  textAlign: 'center',
  padding: '2rem',
  background: '#f8f9fa',
  marginTop: '3rem',
  color: '#6c757d'
}}>
  <p>© 2025 Система штатного расписания предприятия</p>
</footer>
</div>
</Router>
);
}

```

export default App;

Файл dashboard.js

```

import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { otdelAPI, doljnAPI, sotrudAPI, shtatraspAPI } from '../services/api';

function Dashboard() {
  const [stats, setStats] = useState({
    otdel: 0,
    doljn: 0,
    sotrud: 0,
    shtatrasp: 0
  });

```

```

const [loading, setLoading] = useState(true);

useEffect(() => {
  const fetchStats = async () => {
    try {
      const [otdelRes, doljnRes, sotrudRes, shtatraspRes] = await Promise.all([
        otdelAPI.getAll(),
        doljnAPI.getAll(),
        sotrudAPI.getAll(),
        shtatraspAPI.getAll()
      ]);

      setStats({
        otdel: otdelRes.data.length,
        doljn: doljnRes.data.length,
        sotrud: sotrudRes.data.length,
        shtatrasp: shtatraspRes.data.length
      });
    } catch (error) {
      console.error('Ошибка при загрузке статистики:', error);
    } finally {
      setLoading(false);
    }
  };

  fetchStats();
}, []);

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка данных...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <h2>Добро пожаловать в SHTRAS</h2>
      <p>Система управления штатным расписанием предприятия поможет вам эффективно управлять информацией о сотрудниках, отделах и должностях.</p>
    </div>

    <div className="grid">
      <div className="card">
        <h3 style={{ color: '#667eea', marginBottom: '1rem' }}><img alt="Bar chart icon" data-bbox="631 823 653 838" style="vertical-align: middle;"/> Статистика</h3>
        <div style={{ display: 'grid', gridTemplateColumns: 'repeat(2, 1fr)', gap: '1rem' }}>
          <div style={{ textAlign: 'center', padding: '1rem', background: '#f8f9fa', borderRadius: '8px' }}>
            <div style={{ fontSize: '2rem', fontWeight: 'bold', color: '#667eea' }}><{stats.otdel}</div>

```

```

    <div>Отделов</div>
  </div>
  <div style={{ textAlign: 'center', padding: '1rem', background: '#f8f9fa', borderRadius: '8px'
}}>
    <div style={{ fontSize: '2rem', fontWeight: 'bold', color: '#764ba2' }}>{stats.doljn}</div>
    <div>Должностей</div>
  </div>
  <div style={{ textAlign: 'center', padding: '1rem', background: '#f8f9fa', borderRadius: '8px'
}}>
    <div style={{ fontSize: '2rem', fontWeight: 'bold', color: '#28a745' }}>{stats.sotrud}</div>
    <div>Сотрудников</div>
  </div>
  <div style={{ textAlign: 'center', padding: '1rem', background: '#f8f9fa', borderRadius: '8px'
}}>
    <div style={{ fontSize: '2rem', fontWeight: 'bold', color: '#dc3545'
}}>{stats.shtatrasp}</div>
    <div>Записей в штатном расписании</div>
  </div>
</div>

<div className="card">
  <h3 style={{ color: '#667eea', marginBottom: '1rem' }}><img alt="rocket icon" data-bbox="628 412 655 430"/> Быстрые действия</h3>
  <div style={{ display: 'flex', flexDirection: 'column', gap: '0.5rem' }}>
    <Link to="/otdel" className="btn btn-primary">Управление отделами</Link>
    <Link to="/doljn" className="btn btn-secondary">Управление должностями</Link>
    <Link to="/sotrud" className="btn btn-success">Управление сотрудниками</Link>
    <Link to="/shtatrasp" className="btn" style={{ background: '#fd7e14', color: 'white'
}}>Штатное расписание</Link>
  </div>
</div>

<div className="card">
  <h3 style={{ color: '#667eea', marginBottom: '1rem' }}><img alt="notepad icon" data-bbox="628 595 655 613"/> Функции системы</h3>
  <ul style={{ listStyle: 'none', padding: 0 }}>
    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      <input checked="" type="checkbox"/> Ведение учета информации об отделах
    </li>
    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      <input checked="" type="checkbox"/> Ведение учета информации о должностях
    </li>
    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      <input checked="" type="checkbox"/> Ведение учета информации о сотрудниках
    </li>
    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      <input checked="" type="checkbox"/> Управление штатным расписанием
    </li>
  </ul>
</div>

```

```

    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      ☒ Формирование отчетных документов
    </li>
    <li style={{ marginBottom: '0.5rem', padding: '0.5rem', background: '#f8f9fa', borderRadius:
'5px' }}>
      ☒ Выполнение запросов к базе данных
    </li>
  </ul>
</div>

<div className="card">
  <h3 style={{ color: '#667eea', marginBottom: '1rem' }}>☒ Отчеты</h3>
  <p style={{ marginBottom: '1rem' }}>Получите подробную аналитику по штатному
расписанию:</p>
  <Link to="/reports" className="btn btn-primary">Перейти к отчетам</Link>
</div>
</div>
);
}

```

export default Dashboard;

Файл doljn.js

```

import React, { useState, useEffect } from 'react';
import { doljnAPI } from '../services/api';

```

```

function DoljnModal({ isOpen, onClose, onSave, doljn }) {
  const [formData, setFormData] = useState({ nazv: "", oklad: "" });

```

```

  useEffect(() => {
    if (doljn) {
      setFormData({ nazv: doljn.nazv, oklad: doljn.oklad });
    } else {
      setFormData({ nazv: "", oklad: "" });
    }
  }, [doljn]);

```

```

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!formData.nazv.trim()) {
      alert('Название должности обязательно для заполнения');
      return;
    }
    if (!formData.oklad || formData.oklad <= 0) {
      alert('Оклад должен быть положительным числом');
      return;
    }
    onSave({ ...formData, oklad: parseInt(formData.oklad) });
  };

```

```

if (!isOpen) return null;

return (
  <div className="modal-overlay" onClick={onClose}>
    <div className="modal" onClick={e => e.stopPropagation()}>
      <h3>{doljn ? 'Редактировать должность' : 'Добавить должность'}</h3>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label className="form-label">Название должности:</label>
          <input
            type="text"
            className="form-input"
            value={formData.nazv}
            onChange={(e) => setFormData({ ...formData, nazv: e.target.value })}
            placeholder="Введите название должности"
            maxLength={30}
          />
        </div>
        <div className="form-group">
          <label className="form-label">Оклад (руб.):</label>
          <input
            type="number"
            className="form-input"
            value={formData.oklad}
            onChange={(e) => setFormData({ ...formData, oklad: e.target.value })}
            placeholder="Введите размер оклада"
            min="1"
          />
        </div>
        <div style={{ display: 'flex', gap: '1rem', justifyContent: 'flex-end' }}>
          <button type="button" className="btn btn-secondary" onClick={onClose}>
            Отмена
          </button>
          <button type="submit" className="btn btn-primary">
            {doljn ? 'Сохранить' : 'Добавить'}
          </button>
        </div>
      </form>
    </div>
  );
}

function Doljn() {
  const [doljns, setDoljns] = useState([]);
  const [loading, setLoading] = useState(true);
  const [modalOpen, setModalOpen] = useState(false);
  const [editingDoljn, setEditingDoljn] = useState(null);
  const [message, setMessage] = useState("");
  // Состояние для сортировки
  const [sortConfig, setSortConfig] = useState({ key: 'nd', direction: 'asc' });

```

```

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

const fetchDoljns = async () => {
  try {
    const response = await doljnAPI.getAll();
    setDoljns(response.data);
  } catch (error) {
    setMessage('Ошибка при загрузке должностей');
    console.error('Ошибка при загрузке должностей:', error);
  } finally {
    setLoading(false);
  }
};

// Функция сортировки
const sortData = (data, key, direction) => {
  return [...data].sort((a, b) => {
    let aVal = a[key];
    let bVal = b[key];

    // Приведение к числу если это возможно
    if (!isNaN(aVal) && !isNaN(bVal)) {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      // Приведение к строке для сравнения
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
  setSortConfig({ key, direction });
};

// Получение отсортированных данных
const sortedDoljns = sortData(doljns, sortConfig.key, sortConfig.direction);

```

```

// Функция для получения индикатора сортировки
const getSortIndicator = (columnKey) => {
  if (sortConfig.key === columnKey) {
    return sortConfig.direction;
  }
  return 'none';
};

const handleAdd = () => {
  setEditingDoljn(null);
  setModalOpen(true);
};

const handleEdit = (doljn) => {
  setEditingDoljn(doljn);
  setModalOpen(true);
};

const handleSave = async (formData) => {
  try {
    if (editingDoljn) {
      await doljnAPI.update(editingDoljn.nd, formData);
      setMessage('Должность успешно обновлена');
    } else {
      await doljnAPI.create(formData);
      setMessage('Должность успешно добавлена');
    }
    setModalOpen(false);
    fetchDoljns();
  } catch (error) {
    setMessage('Ошибка при сохранении должности');
    console.error('Ошибка при сохранении должности:', error);
  }
};

const handleDelete = async (id) => {
  if (window.confirm('Вы уверены, что хотите удалить эту должность?')) {
    try {
      await doljnAPI.delete(id);
      setMessage('Должность успешно удалена');
      fetchDoljns();
    } catch (error) {
      setMessage('Ошибка при удалении должности');
      console.error('Ошибка при удалении должности:', error);
    }
  }
};

// Очистка сообщения через 3 секунды
useEffect(() => {
  if (message) {

```

```

const timer = setTimeout(() => setMessage(""), 3000);
return () => clearTimeout(timer);
}
}, [message]);

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка должностей...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '2rem' }}>
        <h2>Управление должностями</h2>
        <button className="btn btn-primary" onClick={handleAdd}>
          Добавить должность
        </button>
      </div>

      {message && (
        <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-success'} `}>
          {message}
        </div>
      )}

      {sortedDoljns.length === 0 ? (
        <div className="alert alert-info">
          Должности не найдены. Добавьте первую должность, нажав на кнопку "Добавить
должность".
        </div>
      ) : (
        <div className="table-container">
          <table className="table">
            <thead>
              <tr>
                <th className="sortable" onClick={() => handleSort('nd')}>
                  №
                  <span className={`sort-indicator ${getSortIndicator('nd')}`}></span>
                </th>
                <th className="sortable" onClick={() => handleSort('nazv')}>
                  Название должности
                  <span className={`sort-indicator ${getSortIndicator('nazv')}`}></span>
                </th>
                <th className="sortable" onClick={() => handleSort('oklad')}>
                  Оклад (руб.)
                  <span className={`sort-indicator ${getSortIndicator('oklad')}`}></span>
                </th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>1</td>
                <td>Должность</td>
                <td>10000</td>
              </tr>
            </tbody>
          </table>
        </div>
      )}
    </div>
  );
}

```

```

        </th>
        <th>Действия</th>
      </tr>
    </thead>
    <tbody>
      {sortedDoljns.map((doljn) => (
        <tr key={doljn.nd}>
          <td>{doljn.nd}</td>
          <td>{doljn.nazv}</td>
          <td>{doljn.oklad?.toLocaleString('ru-RU') || '-'}</td>
          <td>
            <button
              className="btn btn-secondary btn-small"
              onClick={() => handleEdit(doljn)}
            >
              Редактировать
            </button>
            <button
              className="btn btn-danger btn-small"
              onClick={() => handleDelete(doljn.nd)}
            >
              Удалить
            </button>
          </td>
        </tr>
      ))}
    </tbody>
  </table>
</div>
)}
</div>

```

```

<DoljnModal
  isOpen={modalOpen}
  onClose={() => setModalOpen(false)}
  onSave={handleSave}
  doljn={editingDoljn}
/>
</div>
);
}

```

export default Doljn;

Файл otdel.js

```

import React, { useState, useEffect } from 'react';
import { otdelAPI } from '../services/api';

```

```

function OtdelModal({ isOpen, onClose, onSave, otdel }) {
  const [formData, setFormData] = useState({ nazvo: " " });

```

```

useEffect(() => {
  if (otdel) {
    setFormData({ nazvo: otdel.nazvo });
  } else {
    setFormData({ nazvo: " " });
  }
}, [otdel]);

const handleSubmit = (e) => {
  e.preventDefault();
  if (!formData.nazvo.trim()) {
    alert('Название отдела обязательно для заполнения');
    return;
  }
  onSave(formData);
};

if (!isOpen) return null;

return (
  <div className="modal-overlay" onClick={onClose}>
    <div className="modal" onClick={e => e.stopPropagation()}>
      <h3>{otdel ? 'Редактировать отдел' : 'Добавить отдел'}</h3>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label className="form-label">Название отдела:</label>
          <input
            type="text"
            className="form-input"
            value={formData.nazvo}
            onChange={(e) => setFormData({ ...formData, nazvo: e.target.value })}
            placeholder="Введите название отдела"
            maxLength={30}
          />
        </div>
        <div style={{ display: 'flex', gap: '1rem', justifyContent: 'flex-end' }}>
          <button type="button" className="btn btn-secondary" onClick={onClose}>
            Отмена
          </button>
          <button type="submit" className="btn btn-primary">
            {otdel ? 'Сохранить' : 'Добавить'}
          </button>
        </div>
      </form>
    </div>
  </div>
);
}

function Otdel() {
  const [otdels, setOtdels] = useState([]);
  const [loading, setLoading] = useState(true);

```

```

const [modalOpen, setModalOpen] = useState(false);
const [editingOtdel, setEditingOtdel] = useState(null);
const [message, setMessage] = useState("");
// Состояние для сортировки
const [sortConfig, setSortConfig] = useState({ key: 'no', direction: 'asc' });

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

const fetchOtdels = async () => {
  try {
    const response = await otdelAPI.getAll();
    setOtdels(response.data);
  } catch (error) {
    setMessage('Ошибка при загрузке отделов');
    console.error('Ошибка при загрузке отделов:', error);
  } finally {
    setLoading(false);
  }
};

// Функция сортировки
const sortData = (data, key, direction) => {
  return [...data].sort((a, b) => {
    let aVal = a[key];
    let bVal = b[key];

    // Приведение к числу если это возможно
    if (!isNaN(aVal) && !isNaN(bVal)) {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      // Приведение к строке для сравнения
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
};

```

```

    setSortConfig({ key, direction });
  };

  // Получение отсортированных данных
  const sortedOtdels = sortData(otdels, sortConfig.key, sortConfig.direction);

  // Функция для получения индикатора сортировки
  const getSortIndicator = (columnKey) => {
    if (sortConfig.key === columnKey) {
      return sortConfig.direction;
    }
    return 'none';
  };

  const handleAdd = () => {
    setEditingOtdel(null);
    setModalOpen(true);
  };

  const handleEdit = (otdel) => {
    setEditingOtdel(otdel);
    setModalOpen(true);
  };

  const handleSave = async (formData) => {
    try {
      if (editingOtdel) {
        await otdelAPI.update(editingOtdel.no, formData);
        setMessage('Отдел успешно обновлен');
      } else {
        await otdelAPI.create(formData);
        setMessage('Отдел успешно добавлен');
      }
      setModalOpen(false);
      fetchOtdels();
    } catch (error) {
      setMessage('Ошибка при сохранении отдела');
      console.error('Ошибка при сохранении отдела:', error);
    }
  };

  const handleDelete = async (id) => {
    if (window.confirm('Вы уверены, что хотите удалить этот отдел?')) {
      try {
        await otdelAPI.delete(id);
        setMessage('Отдел успешно удален');
        fetchOtdels();
      } catch (error) {
        setMessage('Ошибка при удалении отдела');
        console.error('Ошибка при удалении отдела:', error);
      }
    }
  }

```

```

};

// Очистка сообщения через 3 секунды
useEffect(() => {
  if (message) {
    const timer = setTimeout(() => setMessage(""), 3000);
    return () => clearTimeout(timer);
  }
}, [message]);

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка отделов...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '2rem' }}>
        <h2>Управление отделами</h2>
        <button className="btn btn-primary" onClick={handleAdd}>
          Добавить отдел
        </button>
      </div>

      {message && (
        <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-success'}`}>
          {message}
        </div>
      )}

      {sortedOtdels.length === 0 ? (
        <div className="alert alert-info">
          Отделы не найдены. Добавьте первый отдел, нажав на кнопку "Добавить отдел".
        </div>
      ) : (
        <div className="table-container">
          <table className="table">
            <thead>
              <tr>
                <th className="sortable" onClick={() => handleSort('no')}>
                  №
                  <span className={`sort-indicator ${getSortIndicator('no')}`}></span>
                </th>
                <th className="sortable" onClick={() => handleSort('nazvo')}>
                  Название отдела
                  <span className={`sort-indicator ${getSortIndicator('nazvo')}`}></span>
                </th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>1</td>
                <td>Отдел 1</td>
              </tr>
              <tr>
                <td>2</td>
                <td>Отдел 2</td>
              </tr>
              <tr>
                <td>3</td>
                <td>Отдел 3</td>
              </tr>
            </tbody>
          </table>
        </div>
      )}
    </div>
  </div>
);

```

```

        </th>
        <th>Действия</th>
      </tr>
    </thead>
    <tbody>
      {sortedOtdels.map((otdel) => (
        <tr key={otdel.no}>
          <td>{otdel.no}</td>
          <td>{otdel.nazvo}</td>
          <td>
            <button
              className="btn btn-secondary btn-small"
              onClick={() => handleEdit(otdel)}
            >
              Редактировать
            </button>
            <button
              className="btn btn-danger btn-small"
              onClick={() => handleDelete(otdel.no)}
            >
              Удалить
            </button>
          </td>
        </tr>
      )})}
    </tbody>
  </table>
</div>
)}
</div>

<OtdelModal
  isOpen={modalOpen}
  onClose={() => setModalOpen(false)}
  onSave={handleSave}
  otdel={editingOtdel}
/>
</div>
);
}

```

export default Otdel;

Файл Otpusk.js

```

import React, { useState, useEffect } from 'react';
import { otpuskAPI, otdelAPI, sotrudAPI } from '../services/api';

```

// Компонент диаграммы Ганта

```

function GanttChart({ otpusks, selectedDepartments, onSelectVacation }) {
  const today = new Date();
  const startOfYear = new Date(today.getFullYear(), 0, 1);
  const endOfYear = new Date(today.getFullYear(), 11, 31);

```

```

// Создаем массив месяцев для заголовка
const months = [];
for (let month = 0; month < 12; month++) {
  const date = new Date(today.getFullYear(), month, 1);
  months.push({
    name: date.toLocaleDateString('ru-RU', { month: 'long' }),
    daysInMonth: new Date(today.getFullYear(), month + 1, 0).getDate()
  });
}

const totalDays = 365; // Приблизительно

// Группируем отпуска по сотрудникам
const employeeVacations = {};
otpusks.forEach(vacation => {
  const empName = vacation.employee_name || 'Не назначен';
  if (!employeeVacations[empName]) {
    employeeVacations[empName] = {
      name: empName,
      department: vacation.department_name || 'Без отдела',
      vacations: []
    };
  }
  employeeVacations[empName].vacations.push(vacation);
});

// Функция для расчета позиции отпуска на шкале времени
const getVacationPosition = (startDate, endDate) => {
  const start = new Date(startDate);
  const end = new Date(endDate);

  const startDay = Math.max(0, Math.floor((start - startOfYear) / (1000 * 60 * 60 * 24)));
  const endDay = Math.min(totalDays, Math.floor((end - startOfYear) / (1000 * 60 * 60 * 24)));

  const left = (startDay / totalDays) * 100;
  const width = ((endDay - startDay) / totalDays) * 100;

  return { left: `${left}%`, width: `${Math.max(width, 0.5)}%` };
};

// Получаем цвет для типа отпуска
const getVacationColor = (type, status) => {
  const colors = {
    'основной': status === 'утвержденный' ? '#4CAF50' : '#81C784',
    'дополнительный': status === 'утвержденный' ? '#2196F3' : '#64B5F6',
    'больничный': status === 'утвержденный' ? '#FF9800' : '#FFB74D',
    'декретный': status === 'утвержденный' ? '#9C27B0' : '#BA68C8'
  };
  return colors[type] || (status === 'утвержденный' ? '#607D8B' : '#90A4AE');
};

```

```

return (
  <div className="gantt-chart">
    <div className="gantt-header">
      <div className="gantt-employee-header">Сотрудник</div>
      <div className="gantt-timeline-header">
        {months.map((month, index) => (
          <div key={index} className="gantt-month" style={{ flex: month.daysInMonth }}>
            {month.name}
          </div>
        ))}
      </div>
    </div>

    <div className="gantt-body">
      {Object.values(employeeVacations).map((employee, index) => (
        <div key={index} className="gantt-row">
          <div className="gantt-employee-cell">
            <div className="employee-name">{employee.name}</div>
            <div className="employee-department">{employee.department}</div>
          </div>
          <div className="gantt-timeline-cell">
            <div className="gantt-timeline">
              {employee.vacations.map((vacation, vIndex) => {
                const position = getVacationPosition(vacation.start_date, vacation.end_date);
                return (
                  <div
                    key={vIndex}
                    className="gantt-vacation-bar"
                    style={{
                      ...position,
                      backgroundColor: getVacationColor(vacation.type, vacation.status),
                      opacity: vacation.status === 'утвержденный' ? 1 : 0.7
                    }}
                    title={` ${vacation.type} (${vacation.status}): ${new
Date(vacation.start_date).toLocaleDateString('ru-RU')} - ${new
Date(vacation.end_date).toLocaleDateString('ru-RU')} }`
                    onClick={() => onSelectVacation && onSelectVacation(vacation)}
                  >
                    <span className="vacation-label">
                      {vacation.type}
                    </span>
                  </div>
                );
              })}
            </div>
          </div>
        </div>
      ))}
    </div>

    <div className="gantt-legend">
      <div className="legend-item">

```

```

    <div className="legend-color" style={{ backgroundColor: '#4CAF50' }}></div>
    <span>Утвержденный основной</span>
  </div>
  <div className="legend-item">
    <div className="legend-color" style={{ backgroundColor: '#81C784' }}></div>
    <span>Планируемый основной</span>
  </div>
  <div className="legend-item">
    <div className="legend-color" style={{ backgroundColor: '#2196F3' }}></div>
    <span>Утвержденный дополнительный</span>
  </div>
  <div className="legend-item">
    <div className="legend-color" style={{ backgroundColor: '#FF9800' }}></div>
    <span>Больничный</span>
  </div>
</div>
</div>
);
}

```

```

// Модальное окно для добавления/редактирования отпуска
function OtpuskModal({ isOpen, onClose, onSave, otpusk, employees }) {
  const [formData, setFormData] = useState({
    tb: "",
    start_date: "",
    end_date: "",
    type: 'основной',
    status: 'планируемый',
    description: ""
  });

  useEffect(() => {
    if (otpusk) {
      setFormData({
        tb: otpusk.tb || "",
        start_date: otpusk.start_date ? otpusk.start_date.split("T")[0] : "",
        end_date: otpusk.end_date ? otpusk.end_date.split("T")[0] : "",
        type: otpusk.type || 'основной',
        status: otpusk.status || 'планируемый',
        description: otpusk.description || ""
      });
    } else {
      setFormData({
        tb: "",
        start_date: "",
        end_date: "",
        type: 'основной',
        status: 'планируемый',
        description: ""
      });
    }
  }, [otpusk]);
}

```

```

const handleSubmit = (e) => {
  e.preventDefault();
  if (!formData.tb || !formData.start_date || !formData.end_date) {
    alert("Заполните все обязательные поля");
    return;
  }
  if (new Date(formData.start_date) >= new Date(formData.end_date)) {
    alert('Дата окончания должна быть позже даты начала');
    return;
  }
  onSave(formData);
};

if (!isOpen) return null;

return (
  <div className="modal-overlay" onClick={onClose}>
    <div className="modal" onClick={e => e.stopPropagation()}>
      <h3>{otpusk ? 'Редактировать отпуск' : 'Добавить отпуск'}</h3>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label className="form-label">Сотрудник:</label>
          <select
            className="form-select"
            value={formData.tb}
            onChange={(e) => setFormData({ ...formData, tb: e.target.value })}
            required
          >
            <option value="">Выберите сотрудника</option>
            {employees.map((emp) => (
              <option key={emp.tb} value={emp.tb}>
                {emp.fam}
              </option>
            ))}
          </select>
        </div>

        <div className="form-row">
          <div className="form-col">
            <label className="form-label">Дата начала:</label>
            <input
              type="date"
              className="form-input"
              value={formData.start_date}
              onChange={(e) => setFormData({ ...formData, start_date: e.target.value })}
              required
            />
          </div>
          <div className="form-col">
            <label className="form-label">Дата окончания:</label>
            <input

```

```

        type="date"
        className="form-input"
        value={formData.end_date}
        onChange={(e) => setFormData({ ...formData, end_date: e.target.value })}
        required
      />
    </div>
  </div>

  <div className="form-row">
    <div className="form-col">
      <label className="form-label">Тип отпуска:</label>
      <select
        className="form-select"
        value={formData.type}
        onChange={(e) => setFormData({ ...formData, type: e.target.value })}
      >
        <option value="основной">Основной</option>
        <option value="дополнительный">Дополнительный</option>
        <option value="больничный">Больничный</option>
        <option value="декретный">Декретный</option>
      </select>
    </div>
    <div className="form-col">
      <label className="form-label">Статус:</label>
      <select
        className="form-select"
        value={formData.status}
        onChange={(e) => setFormData({ ...formData, status: e.target.value })}
      >
        <option value="планируемый">Планируемый</option>
        <option value="утвержденный">Утвержденный</option>
        <option value="отклоненный">Отклоненный</option>
      </select>
    </div>
  </div>

  <div className="form-group">
    <label className="form-label">Описание:</label>
    <textarea
      className="form-input"
      value={formData.description}
      onChange={(e) => setFormData({ ...formData, description: e.target.value })}
      placeholder="Дополнительная информация об отпуске"
      rows="3"
    />
  </div>

  <div style={{ display: 'flex', gap: '1rem', justifyContent: 'flex-end' }}>
    <button type="button" className="btn btn-secondary" onClick={onClose}>
      Отмена
    </button>
  </div>

```

```

        <button type="submit" className="btn btn-primary">
            {otpusk ? 'Сохранить' : 'Добавить'}
        </button>
    </div>
</form>
</div>
</div>
);
}

function Otpusk() {
    const [otpusks, setOtpusks] = useState([]);
    const [departments, setDepartments] = useState([]);
    const [employees, setEmployees] = useState([]);
    const [loading, setLoading] = useState(true);
    const [modalOpen, setModalOpen] = useState(false);
    const [editingOtpusk, setEditingOtpusk] = useState(null);
    const [message, setMessage] = useState("");
    const [selectedDepartments, setSelectedDepartments] = useState(['all']);
    const [viewMode, setViewMode] = useState('table'); // 'table' или 'gantt'
    // Состояние для сортировки
    const [sortConfig, setSortConfig] = useState({ key: 'start_date', direction: 'desc' });

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

    useEffect(() => {
        fetchOtpusks();
    }, [selectedDepartments]);

    const fetchData = async () => {
        try {
            const [otpuskRes, otDelRes, sotrudRes] = await Promise.all([
                otpuskAPI.getAll(),
                otDelAPI.getAll(),
                sotrudAPI.getAll()
            ]);

            setOtpusks(otpuskRes.data);
            setDepartments(otDelRes.data);
            setEmployees(sotrudRes.data);
        } catch (error) {
            setMessage('Ошибка при загрузке данных');
            console.error('Ошибка при загрузке данных:', error);
        } finally {
            setLoading(false);
        }
    };

    const fetchOtpusks = async () => {
        try {

```

```

const deptQuery = selectedDepartments.includes('all') ? 'all' : selectedDepartments.join(',');
const response = await otpuskAPI.getByDepartments(deptQuery);
setOtpusks(response.data);
} catch (error) {
  console.error('Ошибка при загрузке отпусков:', error);
}
};

// Функция сортировки
const sortData = (data, key, direction) => {
  return [...data].sort((a, b) => {
    let aVal = a[key] || "";
    let bVal = b[key] || "";

    // Обработка дат
    if (key.includes('date')) {
      aVal = new Date(aVal);
      bVal = new Date(bVal);
    } else if (!isNaN(aVal) && !isNaN(bVal) && aVal !== "" && bVal !== "") {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
  setSortConfig({ key, direction });
};

// Получение отсортированных данных
const sortedOtpusks = sortData(otpusks, sortConfig.key, sortConfig.direction);

// Функция для получения индикатора сортировки
const getSortIndicator = (columnKey) => {
  if (sortConfig.key === columnKey) {
    return sortConfig.direction;
  }
  return 'none';
};

```

```

};

const handleDepartmentChange = (deptId) => {
  if (deptId === 'all') {
    // Если кликнули на "Все отделы"
    if (selectedDepartments.includes('all')) {
      // Если уже выбрано, ничего не делаем (оставляем выбранным)
      return;
    } else {
      // Если не выбрано, выбираем только "Все отделы"
      setSelectedDepartments(['all']);
    }
  } else {
    // Если кликнули на конкретный отдел
    if (selectedDepartments.includes('all')) {
      // Если выбрано "Все отделы", убираем его и выбираем только этот отдел
      setSelectedDepartments([deptId]);
    } else {
      // Если "Все отделы" не выбрано, работаем с конкретными отделами
      const newSelection = selectedDepartments.includes(deptId)
        ? selectedDepartments.filter(id => id !== deptId) // Убираем отдел из списка
        : [...selectedDepartments, deptId]; // Добавляем отдел в список

      // Если не осталось выбранных отделов, выбираем "Все отделы"
      setSelectedDepartments(newSelection.length === 0 ? ['all'] : newSelection);
    }
  }
};

const handleAdd = () => {
  setEditingOtpusk(null);
  setModalOpen(true);
};

const handleEdit = (otpusk) => {
  setEditingOtpusk(otpusk);
  setModalOpen(true);
};

const handleSave = async (formData) => {
  try {
    if (editingOtpusk) {
      await otpuskAPI.update(editingOtpusk.id, formData);
      setMessage('Отпуск успешно обновлен');
    } else {
      await otpuskAPI.create(formData);
      setMessage('Отпуск успешно добавлен');
    }
    setModalOpen(false);
    fetchOtpusks();
  } catch (error) {
    setMessage('Ошибка при сохранении отпуска');
  }
};

```

```

    console.error('Ошибка при сохранении отпуска:', error);
  }
};

const handleDelete = async (id) => {
  if (window.confirm('Вы уверены, что хотите удалить этот отпуск?')) {
    try {
      await otпускAPI.delete(id);
      setMessage('Отпуск успешно удален');
      fetchОтпусков();
    } catch (error) {
      setMessage('Ошибка при удалении отпуска');
      console.error('Ошибка при удалении отпуска:', error);
    }
  }
};

const handleSelectVacation = (vacation) => {
  handleEdit(vacation);
};

// Очистка сообщения через 3 секунды
useEffect(() => {
  if (message) {
    const timer = setTimeout(() => setMessage(''), 3000);
    return () => clearTimeout(timer);
  }
}, [message]);

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка отпусков...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '2rem' }}>
        <h2>Управление отпусками</h2>
        <div style={{ display: 'flex', gap: '1rem' }}>
          <div>
            <button
              className={`btn ${viewMode === 'table' ? 'btn-primary' : 'btn-secondary'}`}
              onClick={() => setViewMode('table')}
            >
              Таблица
            </button>

```

```

<button
  className={`btn ${viewMode === 'gantt' ? 'btn-primary' : 'btn-secondary'}`}
  onClick={() => setViewMode('gantt')}
>
  Диаграмма Ганта
</button>
</div>
<button className="btn btn-primary" onClick={handleAdd}>
  Добавить отпуск
</button>
</div>
</div>

{/* Фильтр по отделам */}
<div className="search-form">
  <h3>Фильтр по отделам:</h3>
  <div style={{ display: 'flex', flexWrap: 'wrap', gap: '0.5rem', marginTop: '1rem' }}>
    <label style={{ display: 'flex', alignItems: 'center', gap: '0.5rem' }}>
      <input
        type="checkbox"
        checked={selectedDepartments.includes('all')}
        onChange={() => handleDepartmentChange('all')}
      />
      Все отделы
    </label>
    {departments.map((dept) => (
      <label key={dept.no} style={{ display: 'flex', alignItems: 'center', gap: '0.5rem' }}>
        <input
          type="checkbox"
          checked={selectedDepartments.includes(dept.no.toString())}
          onChange={() => handleDepartmentChange(dept.no.toString())}
        />
        {dept.nazvo}
      </label>
    ))}
  </div>
</div>

{message && (
  <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-success'}`}>
    {message}
  </div>
)}

{viewMode === 'gantt' ? (
  <GanttChart
    otpusks={sortedOtpusks}
    selectedDepartments={selectedDepartments}
    onSelectVacation={handleSelectVacation}
  />
) : (
  <>

```

```

{sortedOtpusks.length === 0 ? (
  <div className="alert alert-info">
    Отпуска не найдены. Добавьте первый отпуск, нажав на кнопку "Добавить отпуск".
  </div>
) : (
  <div className="table-container">
    <table className="table">
      <thead>
        <tr>
          <th className="sortable" onClick={() => handleSort('employee_name')}>
            Сотрудник
            <span className={`sort-indicator ${getSortIndicator('employee_name')}`}></span>
          </th>
          <th className="sortable" onClick={() => handleSort('department_name')}>
            Отдел
            <span className={`sort-indicator ${getSortIndicator('department_name')}`}></span>
          </th>
          <th className="sortable" onClick={() => handleSort('start_date')}>
            Дата начала
            <span className={`sort-indicator ${getSortIndicator('start_date')}`}></span>
          </th>
          <th className="sortable" onClick={() => handleSort('end_date')}>
            Дата окончания
            <span className={`sort-indicator ${getSortIndicator('end_date')}`}></span>
          </th>
          <th className="sortable" onClick={() => handleSort('type')}>
            Тип
            <span className={`sort-indicator ${getSortIndicator('type')}`}></span>
          </th>
          <th className="sortable" onClick={() => handleSort('status')}>
            Статус
            <span className={`sort-indicator ${getSortIndicator('status')}`}></span>
          </th>
          <th>Описание</th>
          <th>Действия</th>
        </tr>
      </thead>
      <tbody>
        {sortedOtpusks.map((otpusk) => (
          <tr key={otpusk.id}>
            <td>{otpusk.employee_name || 'Не назначен'}</td>
            <td>{otpusk.department_name || 'Без отдела'}</td>
            <td>{new Date(otpusk.start_date).toLocaleDateString('ru-RU')}</td>
            <td>{new Date(otpusk.end_date).toLocaleDateString('ru-RU')}</td>
            <td>
              <span className={`status-badge ${otpusk.type}`}>
                {otpusk.type}
              </span>
            </td>
            <td>
              <span className={`status-badge ${otpusk.status}`}>

```

```

        {otpusk.status}
      </span>
    </td>
    <td>{otpusk.description || '-'}</td>
  </tr>
  <tr>
    <td>
      <button
        className="btn btn-secondary btn-small"
        onClick={() => handleEdit(otpusk)}
      >
        Редактировать
      </button>
      <button
        className="btn btn-danger btn-small"
        onClick={() => handleDelete(otpusk.id)}
      >
        Удалить
      </button>
    </td>
  </tr>
</tbody>
</table>
</div>
  </div>
</div>

```

```

<OtpuskModal
  isOpen={modalOpen}
  onClose={() => setModalOpen(false)}
  onSave={handleSave}
  otpusk={editingOtpusk}
  employees={employees}
/>
</div>
);
}

```

export default Otpusk;

Файл Reports.js

```

import React, { useState, useEffect } from 'react';
import { reportsAPI, otdelAPI } from '../services/api';

```

```

function Reports() {
  const [activeReport, setActiveReport] = useState("");
  const [reportData, setReportData] = useState([]);
  const [loading, setLoading] = useState(false);
  const [departments, setDepartments] = useState([]);
  const [selectedDepartment, setSelectedDepartment] = useState("");
  const [salaryRange, setSalaryRange] = useState({ min: "", max: "" });

```

```

const [message, setMessage] = useState("");
// Состояние для сортировки
const [sortConfig, setSortConfig] = useState({ key: "", direction: 'asc' });

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

const fetchDepartments = async () => {
  try {
    const response = await otDelAPI.getAll();
    setDepartments(response.data);
  } catch (error) {
    console.error('Ошибка при загрузке отделов:', error);
  }
};

// Функция сортировки
const sortData = (data, key, direction) => {
  if (!key) return data;

  return [...data].sort((a, b) => {
    let aVal = a[key] || "";
    let bVal = b[key] || "";

    // Приведение к числу если это возможно
    if (!isNaN(aVal) && !isNaN(bVal) && aVal !== "" && bVal !== "") {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      // Приведение к строке для сравнения
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
  setSortConfig({ key, direction });
};

```

```

// Функция для получения индикатора сортировки
const getSortIndicator = (columnKey) => {
  if (sortConfig.key === columnKey) {
    return sortConfig.direction;
  }
  return 'none';
};

const generateReport = async (reportType) => {
  setLoading(true);
  setActiveReport(reportType);
  setMessage("");
  // Сброс сортировки при смене отчета
  setSortConfig({ key: "", direction: 'asc' });

  try {
    let response;

    switch (reportType) {
      case 'positions-alphabetical':
        response = await reportsAPI.positionsAlphabetical();
        setSortConfig({ key: 'nd', direction: 'asc' }); // Сортировка по первому столбцу по умолчанию
        break;
      case 'positions-by-department':
        if (!selectedDepartment) {
          setMessage('Выберите отдел');
          setLoading(false);
          return;
        }
        response = await reportsAPI.positionsByDepartment(selectedDepartment);
        setSortConfig({ key: 'nd', direction: 'asc' });
        break;
      case 'positions-by-salary':
        const minSalary = salaryRange.min || 0;
        const maxSalary = salaryRange.max || 999999;
        response = await reportsAPI.positionsBySalary(minSalary, maxSalary);
        setSortConfig({ key: 'nd', direction: 'asc' });
        break;
      case 'departments-alphabetical':
        response = await reportsAPI.departmentsAlphabetical();
        setSortConfig({ key: 'no', direction: 'asc' });
        break;
      case 'staff-schedule':
        if (!selectedDepartment) {
          setMessage('Выберите отдел');
          setLoading(false);
          return;
        }
        response = await reportsAPI.staffSchedule(selectedDepartment);
        setSortConfig({ key: 'nsh', direction: 'asc' });
        break;
    }
  }

```

```

case 'employees-by-department':
  if (!selectedDepartment) {
    setMessage('Выберите отдел');
    setLoading(false);
    return;
  }
  response = await reportsAPI.employeesByDepartment(selectedDepartment);
  setSortConfig({ key: 'tb', direction: 'asc' });
  break;
default:
  setMessage('Неизвестный тип отчета');
  setLoading(false);
  return;
}

setReportData(response.data);
if (response.data.length === 0) {
  setMessage('Нет данных для отображения');
}
} catch (error) {
  setMessage('Ошибка при генерации отчета');
  console.error('Ошибка при генерации отчета:', error);
} finally {
  setLoading(false);
}
};

// Получение отсортированных данных
const sortedReportData = sortData(reportData, sortConfig.key, sortConfig.direction);

const renderReportTable = () => {
  if (sortedReportData.length === 0) return null;

  switch (activeReport) {
    case 'positions-alphabetical':
    case 'positions-by-salary':
      return (
        <table className="table">
          <thead>
            <tr>
              <th className="sortable" onClick={() => handleSort('nd')}>
                №
                <span className={`sort-indicator ${getSortIndicator('nd')}`}></span>
              </th>
              <th className="sortable" onClick={() => handleSort('nazv')}>
                Название должности
                <span className={`sort-indicator ${getSortIndicator('nazv')}`}></span>
              </th>
              <th className="sortable" onClick={() => handleSort('oklad')}>
                Оклад (руб.)
                <span className={`sort-indicator ${getSortIndicator('oklad')}`}></span>
              </th>
            </tr>
          </thead>
        </table>
      );
    }
  }

```

```

</tr>
</thead>
<tbody>
  {sortedReportData.map((item) => (
    <tr key={item.nd}>
      <td>{item.nd}</td>
      <td>{item.nazv}</td>
      <td>{item.oklad?.toLocaleString('ru-RU')}</td>
    </tr>
  ))}
</tbody>
</table>
);

```

case 'positions-by-department':

```

return (
  <table className="table">
    <thead>
      <tr>
        <th className="sortable" onClick={() => handleSort('nd')}>
          №
          <span className={`sort-indicator ${getSortIndicator('nd')}`} `></span>
        </th>
        <th className="sortable" onClick={() => handleSort('nazv')}>
          Название должности
          <span className={`sort-indicator ${getSortIndicator('nazv')}`} `></span>
        </th>
        <th className="sortable" onClick={() => handleSort('oklad')}>
          Оклад (руб.)
          <span className={`sort-indicator ${getSortIndicator('oklad')}`} `></span>
        </th>
        <th className="sortable" onClick={() => handleSort('otdel_name')}>
          Отдел
          <span className={`sort-indicator ${getSortIndicator('otdel_name')}`} `></span>
        </th>
      </tr>
    </thead>
    <tbody>
      {sortedReportData.map((item) => (
        <tr key={item.nd}>
          <td>{item.nd}</td>
          <td>{item.nazv}</td>
          <td>{item.oklad?.toLocaleString('ru-RU')}</td>
          <td>{item.otdel_name}</td>
        </tr>
      ))}
    </tbody>
  </table>
);

```

case 'departments-alphabetical':

```

return (

```

```

<table className="table">
  <thead>
    <tr>
      <th className="sortable" onClick={() => handleSort('no')}>
        №
        <span className={`sort-indicator ${getSortIndicator('no')}}`></span>
      </th>
      <th className="sortable" onClick={() => handleSort('nazvo')}>
        Название отдела
        <span className={`sort-indicator ${getSortIndicator('nazvo')}}`></span>
      </th>
    </tr>
  </thead>
  <tbody>
    {sortedReportData.map((item) => (
      <tr key={item.no}>
        <td>{item.no}</td>
        <td>{item.nazvo}</td>
      </tr>
    ))}
  </tbody>
</table>
);

```

case 'staff-schedule':

```

return (
  <table className="table">
    <thead>
      <tr>
        <th className="sortable" onClick={() => handleSort('nsh')}>
          № ИП
          <span className={`sort-indicator ${getSortIndicator('nsh')}}`></span>
        </th>
        <th className="sortable" onClick={() => handleSort('otdel_name')}>
          Отдел
          <span className={`sort-indicator ${getSortIndicator('otdel_name')}}`></span>
        </th>
        <th className="sortable" onClick={() => handleSort('dolzn_name')}>
          Должность
          <span className={`sort-indicator ${getSortIndicator('dolzn_name')}}`></span>
        </th>
        <th className="sortable" onClick={() => handleSort('oklad')}>
          Оклад (руб.)
          <span className={`sort-indicator ${getSortIndicator('oklad')}}`></span>
        </th>
        <th className="sortable" onClick={() => handleSort('fam')}>
          Сотрудник
          <span className={`sort-indicator ${getSortIndicator('fam')}}`></span>
        </th>
      </tr>
    </thead>
    <tbody>

```

```

        {sortedReportData.map((item) => (
            <tr key={item.nsh}>
                <td>{item.nsh}</td>
                <td>{item.otdel_name}</td>
                <td>{item.dolzn_name || '-'}</td>
                <td>{item.oklad?.toLocaleString('ru-RU') || '-'}</td>
                <td>{item.fam || '-'}</td>
            </tr>
        )))
    </tbody>
</table>
);

case 'employees-by-department':
    return (
        <table className="table">
            <thead>
                <tr>
                    <th className="sortable" onClick={() => handleSort('tb')}>
                        Табельный номер
                        <span className={`sort-indicator ${getSortIndicator('tb')}}`></span>
                    </th>
                    <th className="sortable" onClick={() => handleSort('fam')}>
                        ФИО
                        <span className={`sort-indicator ${getSortIndicator('fam')}}`></span>
                    </th>
                    <th className="sortable" onClick={() => handleSort('sadres')}>
                        Адрес
                        <span className={`sort-indicator ${getSortIndicator('sadres')}}`></span>
                    </th>
                    <th className="sortable" onClick={() => handleSort('otdel_name')}>
                        Отдел
                        <span className={`sort-indicator ${getSortIndicator('otdel_name')}}`></span>
                    </th>
                </tr>
            </thead>
            <tbody>
                {sortedReportData.map((item) => (
                    <tr key={item.tb}>
                        <td>{item.tb}</td>
                        <td>{item.fam}</td>
                        <td>{item.sadres || '-'}</td>
                        <td>{item.otdel_name}</td>
                    </tr>
                )))
            </tbody>
        </table>
    );

default:
    return null;
}

```

```
};
```

```
const getReportTitle = () => {
  switch (activeReport) {
    case 'positions-alphabetical':
      return 'Список должностей по алфавиту';
    case 'positions-by-department':
      return 'Список должностей указанного отдела';
    case 'positions-by-salary':
      return 'Список должностей с окладом в указанном диапазоне';
    case 'departments-alphabetical':
      return 'Список отделов по алфавиту';
    case 'staff-schedule':
      return 'Штатное расписание указанного отдела';
    case 'employees-by-department':
      return 'Список сотрудников указанного отдела';
    default:
      return '';
  }
};
```

```
return (
  <div>
    <div className="card">
      <h2>Отчеты и запросы</h2>
      <p>Выберите тип отчета для генерации:</p>

      <div className="grid" style={{ gridTemplateColumns: 'repeat(auto-fit, minmax(250px, 1fr))'
    }}>
      { /* Запросы */ }
      <div className="card">
        <h3 style={{ color: '#667eea', marginBottom: '1rem' }}><img alt="Flag icon" data-bbox="641 561 661 576"/> Запросы</h3>

        <button
          className="btn btn-primary"
          style={{ width: '100%', marginBottom: '0.5rem' }}
          onClick={() => generateReport('positions-alphabetical')}
        >
          Должности по алфавиту
        </button>

        <div style={{ marginBottom: '1rem' }}>
          <label className="form-label">Отдел для запросов:</label>
          <select
            className="form-select"
            value={selectedDepartment}
            onChange={(e) => setSelectedDepartment(e.target.value)}
          >
            <option value="">Выберите отдел</option>
            {departments.map((dept) => (
              <option key={dept.no} value={dept.no}>{dept.nazvo}</option>
            ))}
          </select>
        </div>
      </div>
    </div>
  </div>
);
```

```

</select>
</div>

<button
  className="btn btn-secondary"
  style={{ width: '100%', marginBottom: '0.5rem' }}
  onClick={() => generateReport('positions-by-department')}
  disabled={!selectedDepartment}
>
  Должности отдела
</button>

<div style={{ marginBottom: '1rem' }}>
  <label className="form-label">Диапазон зарплат:</label>
  <div style={{ display: 'flex', gap: '0.5rem' }}>
    <input
      type="number"
      className="form-input"
      placeholder="Мин."
      value={salaryRange.min}
      onChange={(e) => setSalaryRange({...salaryRange, min: e.target.value})}
    />
    <input
      type="number"
      className="form-input"
      placeholder="Макс."
      value={salaryRange.max}
      onChange={(e) => setSalaryRange({...salaryRange, max: e.target.value})}
    />
  </div>
</div>

<button
  className="btn btn-success"
  style={{ width: '100%' }}
  onClick={() => generateReport('positions-by-salary')}
>
  Должности по зарплате
</button>
</div>

{/* Отчеты */}
<div className="card">
  <h3 style={{ color: '#764ba2', marginBottom: '1rem' }}><img alt="document icon" data-bbox="645 758 665 775"/> Отчеты</h3>

  <button
    className="btn btn-primary"
    style={{ width: '100%', marginBottom: '0.5rem' }}
    onClick={() => generateReport('departments-alphabetical')}
  >
    Отделы по алфавиту
  </button>

```

```

<button
  className="btn btn-secondary"
  style={{ width: '100%', marginBottom: '0.5rem' }}
  onClick={() => generateReport('staff-schedule')}
  disabled={!selectedDepartment}
>
  Штатное расписание отдела
</button>

<button
  className="btn btn-success"
  style={{ width: '100%' }}
  onClick={() => generateReport('employees-by-department')}
  disabled={!selectedDepartment}
>
  Сотрудники отдела
</button>
</div>
</div>
</div>

{/* Результаты отчета */}
{loading && (
  <div className="loading">
    <div className="spinner"></div>
    <p>Генерация отчета...</p>
  </div>
)}

{message && (
  <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-info'}`}>
    {message}
  </div>
)}

{!loading && activeReport && sortedReportData.length > 0 && (
  <div className="card">
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '1rem' }}>
      <h3>{getReportTitle()}</h3>
      <div style={{ fontSize: '0.9rem', color: '#666' }}>
        Найдено записей: {sortedReportData.length}
      </div>
    </div>
    <div className="table-container">
      {renderReportTable()}
    </div>
  </div>
)}
</div>
);

```

```

}

export default Reports;

Файл ShtatRasp

import React, { useState, useEffect } from 'react';
import { shtatraspAPI, otdelAPI, doljnAPI, sotrudAPI } from '../services/api';

function ShtatRaspModal({ isOpen, onClose, onSave, shtatrasp, otrels, doljns, sotruds }) {
  const [formData, setFormData] = useState({ no: "", nd: "", tb: "" });

  useEffect(() => {
    if (shtatrasp) {
      setFormData({
        no: shtatrasp.no || "",
        nd: shtatrasp.nd || "",
        tb: shtatrasp.tb || ""
      });
    } else {
      setFormData({ no: "", nd: "", tb: "" });
    }
  }, [shtatrasp]);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!formData.no) {
      alert('Отдел обязателен для заполнения');
      return;
    }
    onSave({
      no: parseInt(formData.no),
      nd: formData.nd ? parseInt(formData.nd) : null,
      tb: formData.tb ? parseInt(formData.tb) : null
    });
  };

  if (!isOpen) return null;

  return (
    <div className="modal-overlay" onClick={onClose}>
      <div className="modal" onClick={e => e.stopPropagation()}>
        <h3>{shtatrasp ? 'Редактировать запись' : 'Добавить запись'}</h3>
        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label className="form-label">Отдел:</label>
            <select
              className="form-select"
              value={formData.no}
              onChange={(e) => setFormData({ ...formData, no: e.target.value })}
            >
              <option value="">Выберите отдел</option>

```

```

    {otdels.map((otdel) => (
      <option key={otdel.no} value={otdel.no}>
        {otdel.nazvo}
      </option>
    )))}
  </select>
</div>
<div className="form-group">
  <label className="form-label">Должность:</label>
  <select
    className="form-select"
    value={formData.nd}
    onChange={(e) => setFormData({ ...formData, nd: e.target.value })}
  >
    <option value="">Выберите должность</option>
    {doljns.map((doljn) => (
      <option key={doljn.nd} value={doljn.nd}>
        {doljn.nazv} ({doljn.oklad?.toLocaleString('ru-RU')} руб.)
      </option>
    )))}
  </select>
</div>
<div className="form-group">
  <label className="form-label">Сотрудник:</label>
  <select
    className="form-select"
    value={formData.tb}
    onChange={(e) => setFormData({ ...formData, tb: e.target.value })}
  >
    <option value="">Выберите сотрудника</option>
    {sotruds.map((sotrud) => (
      <option key={sotrud.tb} value={sotrud.tb}>
        {sotrud.fam}
      </option>
    )))}
  </select>
</div>
<div style={{ display: 'flex', gap: '1rem', justifyContent: 'flex-end' }}>
  <button type="button" className="btn btn-secondary" onClick={onClose}>
    Отмена
  </button>
  <button type="submit" className="btn btn-primary">
    {shtatrasp ? 'Сохранить' : 'Добавить'}
  </button>
</div>
</form>
</div>
</div>
);
}

function ShtatRasp() {

```

```

const [shtatrasps, setShtatrasps] = useState([]);
const [otdels, setOtdels] = useState([]);
const [doljns, setDoljns] = useState([]);
const [sotruds, setSotruds] = useState([]);
const [loading, setLoading] = useState(true);
const [modalOpen, setModalOpen] = useState(false);
const [editingShtatrasp, setEditingShtatrasp] = useState(null);
const [message, setMessage] = useState("");
// Состояние для сортировки
const [sortConfig, setSortConfig] = useState({ key: 'nsh', direction: 'asc' });

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

const fetchData = async () => {
  try {
    const [shtatraspRes, otdelRes, doljnRes, sotrudRes] = await Promise.all([
      shtatraspAPI.getAll(),
      otdelAPI.getAll(),
      doljnAPI.getAll(),
      sotrudAPI.getAll()
    ]);

    setShtatrasps(shtatraspRes.data);
    setOtdels(otdelRes.data);
    setDoljns(doljnRes.data);
    setSotruds(sotrudRes.data);
  } catch (error) {
    setMessage('Ошибка при загрузке данных');
    console.error('Ошибка при загрузке данных:', error);
  } finally {
    setLoading(false);
  }
};

// Функция сортировки
const sortData = (data, key, direction) => {
  return [...data].sort((a, b) => {
    let aVal = a[key] || "";
    let bVal = b[key] || "";

    // Приведение к числу если это возможно
    if (!isNaN(aVal) && !isNaN(bVal) && aVal !== "" && bVal !== "") {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      // Приведение к строке для сравнения
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }
  });
};

```

```

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
  setSortConfig({ key, direction });
};

// Получение отсортированных данных
const sortedShtatrasps = sortData(shtatrasps, sortConfig.key, sortConfig.direction);

// Функция для получения индикатора сортировки
const getSortIndicator = (columnKey) => {
  if (sortConfig.key === columnKey) {
    return sortConfig.direction;
  }
  return 'none';
};

const handleAdd = () => {
  setEditingShtatrasp(null);
  setModalOpen(true);
};

const handleEdit = (shtatrasp) => {
  setEditingShtatrasp(shtatrasp);
  setModalOpen(true);
};

const handleSave = async (formData) => {
  try {
    if (editingShtatrasp) {
      await shtatraspAPI.update(editingShtatrasp.nsh, editingShtatrasp.no, formData);
      setMessage('Запись успешно обновлена');
    } else {
      await shtatraspAPI.create(formData);
      setMessage('Запись успешно добавлена');
    }
    setModalOpen(false);
    fetchData();
  } catch (error) {
    setMessage('Ошибка при сохранении записи');
    console.error('Ошибка при сохранении записи:', error);
  }
};

```

```

    }
  };

const handleDelete = async (nsh, no) => {
  if (window.confirm('Вы уверены, что хотите удалить эту запись?')) {
    try {
      await shtatraspAPI.delete(nsh, no);
      setMessage('Запись успешно удалена');
      fetchData();
    } catch (error) {
      setMessage('Ошибка при удалении записи');
      console.error('Ошибка при удалении записи:', error);
    }
  }
};

// Очистка сообщения через 3 секунды
useEffect(() => {
  if (message) {
    const timer = setTimeout(() => setMessage(''), 3000);
    return () => clearTimeout(timer);
  }
}, [message]);

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка штатного расписания...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '2rem' }}>
        <h2>Штатное расписание</h2>
        <button className="btn btn-primary" onClick={handleAdd}>
          Добавить запись
        </button>
      </div>

      {message && (
        <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-success'}`}>
          {message}
        </div>
      )}

      {sortedShtatrasps.length === 0 ? (
        <div className="alert alert-info">

```

Записи не найдены. Добавьте первую запись, нажав на кнопку "Добавить запись".

```
) : (  
<div className="table-container">  
  <table className="table">  
    <thead>  
      <tr>  
        <th className="sortable" onClick={() => handleSort('nsh')}>  
          № ИП  
          <span className={`sort-indicator ${getSortIndicator('nsh')}`} `></span>  
        </th>  
        <th className="sortable" onClick={() => handleSort('otdel_name')}>  
          Отдел  
          <span className={`sort-indicator ${getSortIndicator('otdel_name')}`} `></span>  
        </th>  
        <th className="sortable" onClick={() => handleSort('dolzn_name')}>  
          Должность  
          <span className={`sort-indicator ${getSortIndicator('dolzn_name')}`} `></span>  
        </th>  
        <th className="sortable" onClick={() => handleSort('oklad')}>  
          Оклад (руб.)  
          <span className={`sort-indicator ${getSortIndicator('oklad')}`} `></span>  
        </th>  
        <th className="sortable" onClick={() => handleSort('fam')}>  
          Сотрудник  
          <span className={`sort-indicator ${getSortIndicator('fam')}`} `></span>  
        </th>  
        <th>Действия</th>  
      </tr>  
    </thead>  
    <tbody>  
      {sortedShtatrasps.map((shtatrasp) => (  
        <tr key={` ${shtatrasp.nsh}-${shtatrasp.no}`} >  
          <td>{shtatrasp.nsh}</td>  
          <td>{shtatrasp.otdel_name || '-'}</td>  
          <td>{shtatrasp.dolzn_name || '-'}</td>  
          <td>{shtatrasp.oklad?.toLocaleString('ru-RU') || '-'}</td>  
          <td>{shtatrasp.fam || '-'}</td>  
          <td>  
            <button  
              className="btn btn-secondary btn-small"  
              onClick={() => handleEdit(shtatrasp)}  
            >  
              Редактировать  
            </button>  
            <button  
              className="btn btn-danger btn-small"  
              onClick={() => handleDelete(shtatrasp.nsh, shtatrasp.no)}  
            >  
              Удалить  
            </button>  
          </td>  
        </tr>  
      )
```

```

        </tr>
      )})
    </tbody>
  </table>
</div>
})
</div>

<ShtatRaspModal
  isOpen={modalOpen}
  onClose={() => setModalOpen(false)}
  onSave={handleSave}
  shtatrasp={editingShtatrasp}
  otodels={otodels}
  doljns={doljns}
  sotrud={sotrud}
/>
</div>
);
}

```

export default ShtatRasp;

Файл Sotrud

```

import React, { useState, useEffect } from 'react';
import { sotrudAPI } from '../services/api';

function SotrudModal({ isOpen, onClose, onSave, sotrud }) {
  const [formData, setFormData] = useState({ fam: "", sadres: "" });

  useEffect(() => {
    if (sotrud) {
      setFormData({ fam: sotrud.fam || "", sadres: sotrud.sadres || "" });
    } else {
      setFormData({ fam: "", sadres: "" });
    }
  }, [sotrud]);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!formData.fam.trim()) {
      alert('ФИО сотрудника обязательно для заполнения');
      return;
    }
    onSave(formData);
  };

  if (!isOpen) return null;

  return (
    <div className="modal-overlay" onClick={onClose}>

```

```

<div className="modal" onClick={e => e.stopPropagation()}>
  <h3>{sotrud ? 'Редактировать сотрудника' : 'Добавить сотрудника'}</h3>
  <form onSubmit={handleSubmit}>
    <div className="form-group">
      <label className="form-label">ФИО сотрудника:</label>
      <input
        type="text"
        className="form-input"
        value={formData.fam}
        onChange={(e) => setFormData({ ...formData, fam: e.target.value })}
        placeholder="Введите ФИО сотрудника"
        maxLength={45}
      />
    </div>
    <div className="form-group">
      <label className="form-label">Адрес:</label>
      <input
        type="text"
        className="form-input"
        value={formData.sadres}
        onChange={(e) => setFormData({ ...formData, sadres: e.target.value })}
        placeholder="Введите адрес сотрудника"
        maxLength={45}
      />
    </div>
    <div style={{ display: 'flex', gap: '1rem', justifyContent: 'flex-end' }}>
      <button type="button" className="btn btn-secondary" onClick={onClose}>
        Отмена
      </button>
      <button type="submit" className="btn btn-primary">
        {sotrud ? 'Сохранить' : 'Добавить'}
      </button>
    </div>
  </form>
</div>
</div>
);
}

```

```

function Sotrud() {
  const [sotruds, setSotruds] = useState([]);
  const [loading, setLoading] = useState(true);
  const [modalOpen, setModalOpen] = useState(false);
  const [editingSotrud, setEditingSotrud] = useState(null);
  const [message, setMessage] = useState("");
  // Состояние для сортировки
  const [sortConfig, setSortConfig] = useState({ key: 'tb', direction: 'asc' });

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

```

```

const fetchSotruds = async () => {
  try {
    const response = await sotrudAPI.getAll();
    setSotruds(response.data);
  } catch (error) {
    setMessage('Ошибка при загрузке сотрудников');
    console.error('Ошибка при загрузке сотрудников:', error);
  } finally {
    setLoading(false);
  }
};

// Функция сортировки
const sortData = (data, key, direction) => {
  return [...data].sort((a, b) => {
    let aVal = a[key] || '';
    let bVal = b[key] || '';

    // Приведение к числу если это возможно
    if (!isNaN(aVal) && !isNaN(bVal)) {
      aVal = Number(aVal);
      bVal = Number(bVal);
    } else {
      // Приведение к строке для сравнения
      aVal = String(aVal).toLowerCase();
      bVal = String(bVal).toLowerCase();
    }

    if (direction === 'asc') {
      return aVal < bVal ? -1 : aVal > bVal ? 1 : 0;
    } else {
      return aVal > bVal ? -1 : aVal < bVal ? 1 : 0;
    }
  });
};

// Обработчик клика по заголовку для сортировки
const handleSort = (key) => {
  let direction = 'asc';
  if (sortConfig.key === key && sortConfig.direction === 'asc') {
    direction = 'desc';
  }
  setSortConfig({ key, direction });
};

// Получение отсортированных данных
const sortedSotruds = sortData(sotruds, sortConfig.key, sortConfig.direction);

// Функция для получения индикатора сортировки
const getSortIndicator = (columnKey) => {
  if (sortConfig.key === columnKey) {
    return sortConfig.direction;
  }
};

```

```

    }
    return 'none';
  };

const handleAdd = () => {
  setEditingSotrud(null);
  setModalOpen(true);
};

const handleEdit = (sotrud) => {
  setEditingSotrud(sotrud);
  setModalOpen(true);
};

const handleSave = async (formData) => {
  try {
    if (editingSotrud) {
      await sotrudAPI.update(editingSotrud.tb, formData);
      setMessage('Сотрудник успешно обновлен');
    } else {
      await sotrudAPI.create(formData);
      setMessage('Сотрудник успешно добавлен');
    }
    setModalOpen(false);
    fetchSotruds();
  } catch (error) {
    setMessage('Ошибка при сохранении сотрудника');
    console.error('Ошибка при сохранении сотрудника:', error);
  }
};

const handleDelete = async (id) => {
  if (window.confirm('Вы уверены, что хотите удалить этого сотрудника?')) {
    try {
      await sotrudAPI.delete(id);
      setMessage('Сотрудник успешно удален');
      fetchSotruds();
    } catch (error) {
      setMessage('Ошибка при удалении сотрудника');
      console.error('Ошибка при удалении сотрудника:', error);
    }
  }
};

// Очистка сообщения через 3 секунды
useEffect(() => {
  if (message) {
    const timer = setTimeout(() => setMessage(''), 3000);
    return () => clearTimeout(timer);
  }
}, [message]);

```

```

if (loading) {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Загрузка сотрудников...</p>
    </div>
  );
}

return (
  <div>
    <div className="card">
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center',
marginBottom: '2rem' }}>
        <h2>Управление сотрудниками</h2>
        <button className="btn btn-primary" onClick={handleAdd}>
          Добавить сотрудника
        </button>
      </div>

      {message && (
        <div className={`alert ${message.includes('Ошибка') ? 'alert-error' : 'alert-success'}`}>
          {message}
        </div>
      )}

      {sortedSotruds.length === 0 ? (
        <div className="alert alert-info">
          Сотрудники не найдены. Добавьте первого сотрудника, нажав на кнопку "Добавить
сотрудника".
        </div>
      ) : (
        <div className="table-container">
          <table className="table">
            <thead>
              <tr>
                <th className="sortable" onClick={() => handleSort('tb')}>
                  Табельный номер
                  <span className={`sort-indicator ${getSortIndicator('tb')}`}></span>
                </th>
                <th className="sortable" onClick={() => handleSort('fam')}>
                  ФИО
                  <span className={`sort-indicator ${getSortIndicator('fam')}`}></span>
                </th>
                <th className="sortable" onClick={() => handleSort('sadres')}>
                  Адрес
                  <span className={`sort-indicator ${getSortIndicator('sadres')}`}></span>
                </th>
                <th>Действия</th>
              </tr>
            </thead>
            <tbody>

```

```

    {sortedSotrud.s.map((sotrud) => (
      <tr key={sotrud.tb}>
        <td>{sotrud.tb}</td>
        <td>{sotrud.fam}</td>
        <td>{sotrud.sadres || '-'}</td>
        <td>
          <button
            className="btn btn-secondary btn-small"
            onClick={() => handleEdit(sotrud)}
          >
            Редактировать
          </button>
          <button
            className="btn btn-danger btn-small"
            onClick={() => handleDelete(sotrud.tb)}
          >
            Удалить
          </button>
        </td>
      </tr>
    ))}
  </tbody>
</table>
</div>
)}
</div>
<SotrudModal
  isOpen={modalOpen}
  onClose={() => setModalOpen(false)}
  onSave={handleSave}
  sotrud={editingSotrud}
/>
</div>
);
}

export default Sotrud;

Файл api.js

import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:5000';

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

export const otDelAPI = {
  getAll: () => api.get('/api/otdel'),

```

```

create: (data) => api.post('/api/otdel', data),
update: (id, data) => api.put(`/api/otdel/${id}`, data),
delete: (id) => api.delete(`/api/otdel/${id}`),
};

// API для должностей
export const doljnAPI = {
  getAll: () => api.get('/api/doljn'),
  create: (data) => api.post('/api/doljn', data),
  update: (id, data) => api.put(`/api/doljn/${id}`, data),
  delete: (id) => api.delete(`/api/doljn/${id}`),
};

// API для сотрудников
export const sotrudAPI = {
  getAll: () => api.get('/api/sotrud'),
  create: (data) => api.post('/api/sotrud', data),
  update: (id, data) => api.put(`/api/sotrud/${id}`, data),
  delete: (id) => api.delete(`/api/sotrud/${id}`),
};

// API для штатного расписания
export const shtatraspAPI = {
  getAll: () => api.get('/api/shtatrasp'),
  create: (data) => api.post('/api/shtatrasp', data),
  update: (nsh, no, data) => api.put(`/api/shtatrasp/${nsh}/${no}`, data),
  delete: (nsh, no) => api.delete(`/api/shtatrasp/${nsh}/${no}`),
};

// API для отпусков
export const otpuskAPI = {
  getAll: () => api.get('/api/otpusk'),
  getByDepartments: (departments) => api.get(`/api/otpusk/by-departments?departments=${departments}`),
  create: (data) => api.post('/api/otpusk', data),
  update: (id, data) => api.put(`/api/otpusk/${id}`, data),
  delete: (id) => api.delete(`/api/otpusk/${id}`),
};

// API для отчетов
export const reportsAPI = {
  positionsAlphabetical: () => api.get('/api/reports/positions-alphabetical'),
  positionsByDepartment: (deptId) => api.get(`/api/reports/positions-by-department/${deptId}`),
  positionsBySalary: (minSalary, maxSalary) => api.get(`/api/reports/positions-by-salary?minSalary=${minSalary}&maxSalary=${maxSalary}`),
  departmentsAlphabetical: () => api.get('/api/reports/departments-alphabetical'),
  staffSchedule: (deptId) => api.get(`/api/reports/staff-schedule/${deptId}`),
  employeesByDepartment: (deptId) => api.get(`/api/reports/employees-by-department/${deptId}`),
};

export default api;

```

```

{
  "name": "shtras-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.3.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "react-router-dom": "^6.3.0",
    "axios": "^1.4.0",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "proxy": "http://localhost:5000"
}

```

Приложение Г

(Обязательное)

StaffPlan: программное обеспечение кадрового учета

Презентация

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		139

StaffPlan: программное обеспечение кадрового учета

Выполнил:

ст-т гр. 20воЭ1 Волков В.А.

Научный руководитель:

к.т.н., доцент Егоров В.Ю.

Цель и задачи

Цель:

Разработать программное средство для помощи в работе сотрудников отдела кадров по взаимодействию с сотрудниками, должностями и штатным расписанием предприятия

Задачи:

- Выполнить анализ предметной области и определить круг информационных объектов: сотрудники, должности, отделы, штатное расписание;
- Разработать концептуальную, логическую и физическую модели базы данных с соблюдением требований нормализации;
- Реализовать базу данных с использованием SQL-скриптов, генераторов и триггеров в СУБД PostgreSQL;
- Спроектировать архитектуру клиент-серверного приложения и реализовать основные модули интерфейса;
- Провести тестирование приложения с использованием тестовых сценариев и зафиксировать результаты

Актуальность и проблематика



Технологический стек



Логическая и концептуальная модели базы данных

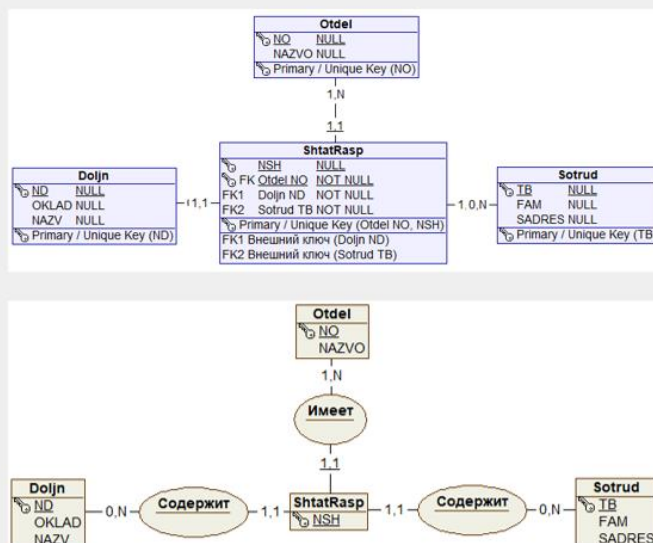
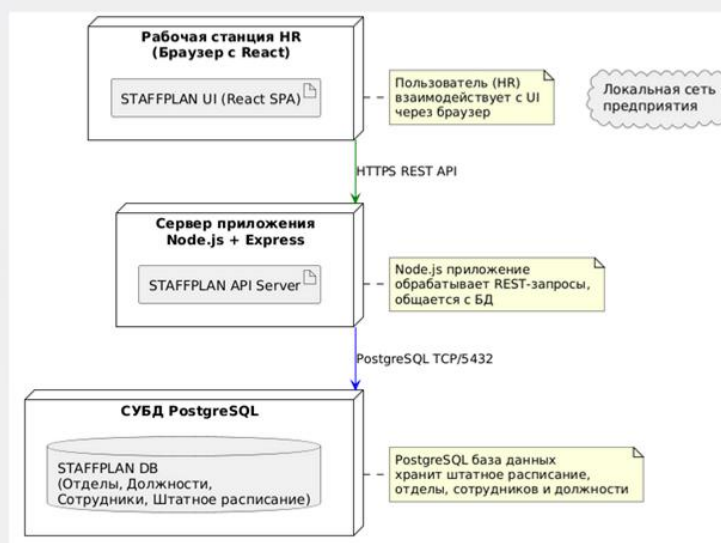
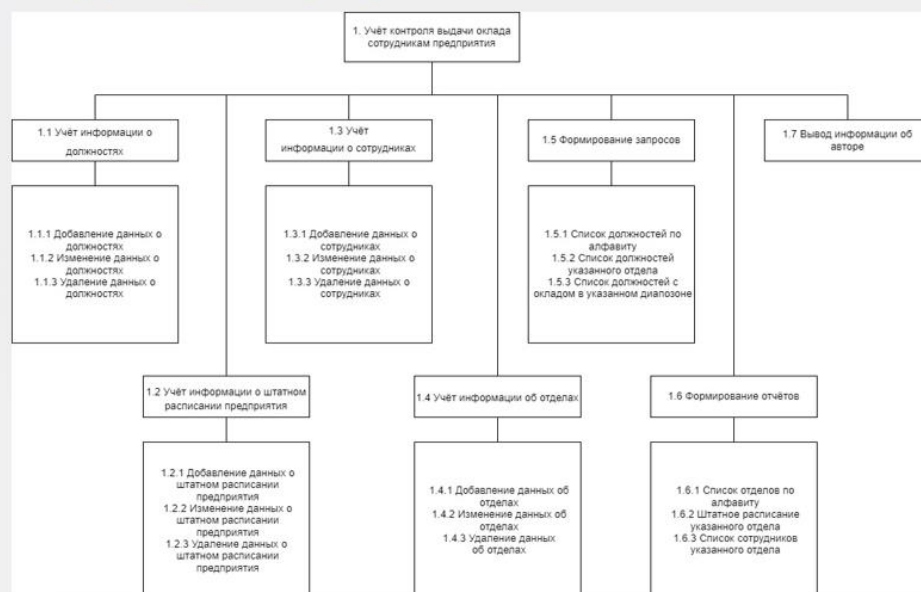


Диаграмма разворачивания системы



Иерархия функций программного обеспечения



Варианты использования приложения

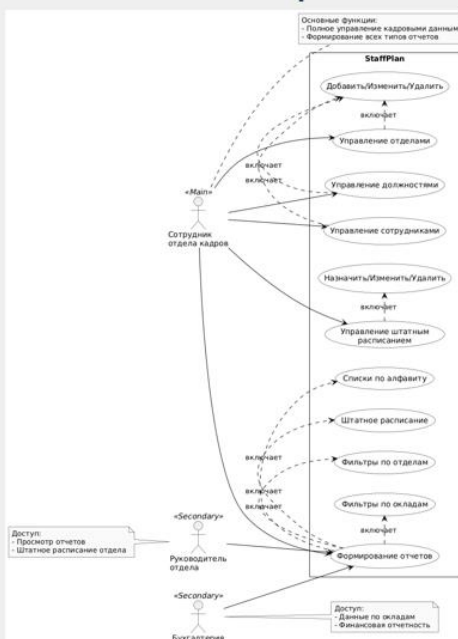
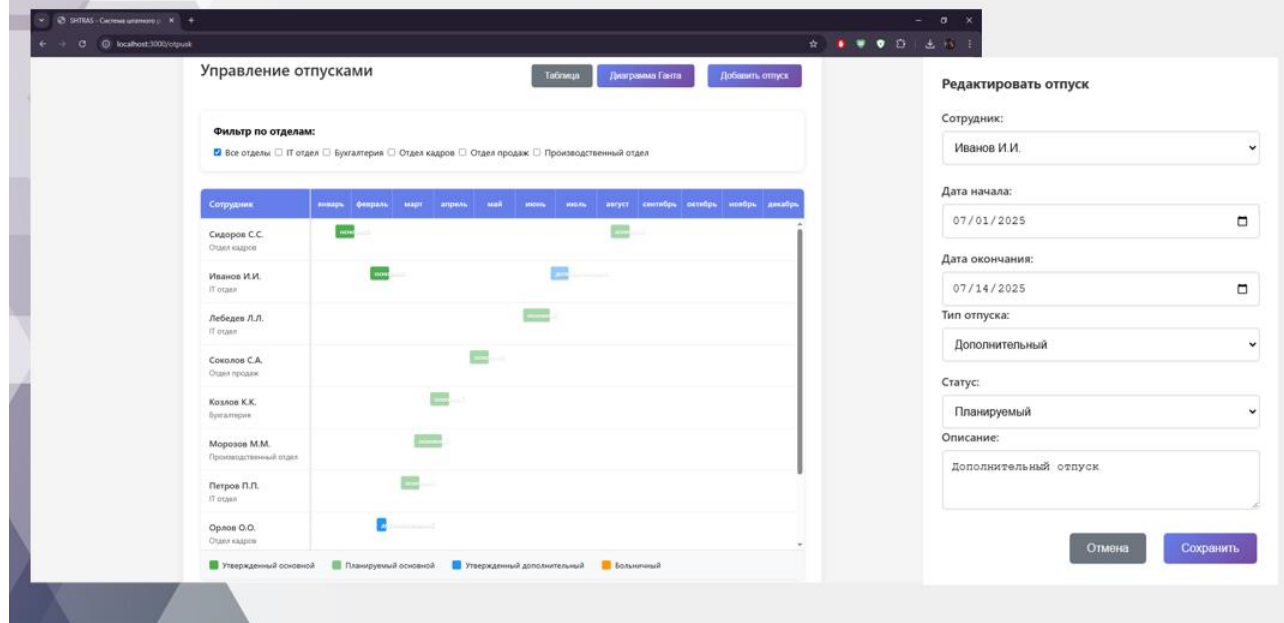


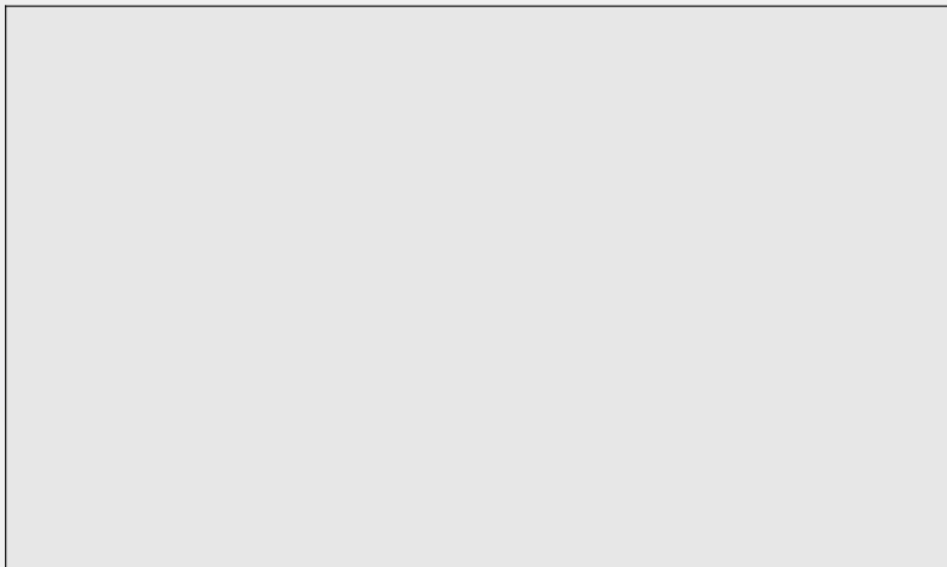
Диаграмма Ганта в программном обеспечении



Заключение

- Разработанное программное средство эффективно автоматизирует взаимодействие сотрудников отдела кадров с персоналом, должностями и штатным расписанием предприятия.
- Проведённый анализ предметной области, проектирование базы данных с соблюдением нормализации, а также реализация в PostgreSQL обеспечили надёжное хранение и обработку данных.
- Архитектура клиент-серверного приложения и удобный интерфейс способствуют повышению продуктивности работы кадрового отдела.
- Тестирование подтвердило стабильность и корректность функционирования системы.
- В дальнейшем планируется развитие программного обеспечения за счёт интеграции искусственного интеллекта для повышения возможностей и удобства взаимодействия пользователя с системой.

Демонстрация работы программного обеспечения



					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		145