Частное учреждение образования

«Колледж бизнеса и права»

ОТЧЕТ

по производственной преддипломной ПРАКТИКЕ

ОП Т.218001

Руководитель практики

от предприятия (Д. Ю. Некрасов)

М.П.

Руководитель практики

от колледжа (С. В. Банцевич )

Обучающаяся (В. А. Белисова )

2025

**СОДЕРЖАНИЕ**

Изм.

Изм.

Изм.

Изм.

Лист

Лист

Лист

Лист

№ докум.

№ докум.

№ докум.

№ докум.

Подпись

Подпись

Подпись

Подпись

Дата

Дата

Дата

Дата

Лист

Лист

Лист

Лист

2

3

3

3

ОП Т.218001

КП Т.716007.401 ПЗ

КП Т.716009.401 ПЗ

КП Т.716007.401 ПЗ

Разраб.

Разраб.

Разраб.

Разраб.

*Белисова В. А.*

*Голубев И.В.*

*Гринь А.М.*

*Голубев И.В.*

Провер.

Провер.

Провер.

Провер.

Банцевич С. В.

Банцевич. С.В

Банцевич. С.В

Банцевич. С.В

Отчёт по преддипломной практике

Лит.

Листов

Листов

Листов

Листов

54

50

55

50

КБП

КБП

КБП

КБП

[Введение 3](#_Toc196348483)

[1 Описание задачи 5](#_Toc196348484)

[1.1 Анализ предметной области 5](#_Toc196348485)

[1.2 Постановка задачи 6](#_Toc196348486)

[2 Проектирование веб-приложения 8](#_Toc196348487)

[2.1 Проектирование модели 8](#_Toc196348488)

[2.2 Требования к веб-приложению 10](#_Toc196348489)

[2.3 Структура веб-приложения 11](#_Toc196348490)

[2.4 Проектирование макета веб-приложения 12](#_Toc196348491)

[2.5 Программно-технические средства, необходимые для разработки приложения 15](#_Toc196348492)

[2.6 Защита и сохранность данных 16](#_Toc196348493)

[2.7 Организация и введение информационной базы (модели) 17](#_Toc196348494)

[3 Реализация веб-приложения 22](#_Toc196348495)

[3.1 Описание разделов веб-приложения 22](#_Toc196348496)

[3.2 Разработка административной части приложения 23](#_Toc196348497)

[3.3 Разработка клиентской части приложения 23](#_Toc196348498)

[3.4 Описание используемых функций и процедур 24](#_Toc196348499)

[3.5 Функциональное тестирование 25](#_Toc196348500)

[4 Применение 29](#_Toc196348501)

[4.1 Назначение веб-приложения 29](#_Toc196348502)

[4.2 Функциональное тестирование 31](#_Toc196348503)

[Заключение 32](#_Toc196348504)

[Список использованных источников 33](#_Toc196348505)

[ПРИЛОЖЕНИЕ А Текст модулей приложения 34](#_Toc196348506)

[ПРИЛОЖЕНИЕ Б Результаты работы приложения 49](#_Toc196348507)

У

3

3

3

УУ

Лит.

У

Лит.

У

Лит.

У

Введение

В современных условиях ведения бизнеса растущая сложность задач и увеличение объемов работы предъявляют повышенные требования к организации труда сотрудников компании. Эффективность распределения задач, контроль их выполнения и своевременное реагирование на изменения играют ключевую роль в успешной деятельности компании. Внедрение специализированного веб-приложения позволит автоматизировать и оптимизировать процессы планирования, распределения и контроля задач, тем самым существенно повысив производительность и оперативность работы сотрудников.

Целью дипломного проекта является разработка веб-приложения для распределения задач между сотрудниками компании, которое обеспечит следующие функциональные возможности:

* регистрацию и авторизацию пользователей с учётом их ролей (менеджер, сотрудник);
* ведение журнала задач в разрезе проектов и заказов;
* планирование и распределение задач по сотрудникам с учётом регламентов;
* визуализацию задач в виде Kanban-доски с изменением статуса путём перетаскивания карточек;
* контроль сроков и учёт времени выполнения задач;
* расширенные возможности просмотра, фильтрации, поиска и сортировки;
* операции добавления, удаления и обновления информации;
* формирование отчётности в заданных разрезах и экспорт отчётов;
* адаптивный дизайн и корректное отображение интерфейса в актуальных версиях браузеров.

Для достижения поставленной цели необходимо решить следующие задачи:

* исследовать предметную область и определить требования к системе;
* спроектировать базу данных, удовлетворяющую третьей нормальной форме, с таблицами, атрибутами и связями;
* разработать архитектуру веб-приложения с учетом многоуровневой структуры (клиентская часть, серверная часть, база данных);
* создать интуитивно понятный интерфейс и удобную систему навигации;
* реализовать функционал авторизации и регистрации с управлением ролями;
* разработать механизм планирования и распределения задач между сотрудниками с контролем исполнения;
* внедрить систему мониторинга выполнения задач и расчета рабочего времени сотрудников;
* обеспечить гибкую систему отчетности с возможностью экспорта данных;
* провести функциональное и интеграционное тестирование веб-приложения на основе разработанных тест-кейсов;
* описать условия и примеры применения веб-приложения в реальных рабочих ситуациях.

Структура пояснительной записки будет состоять из следующих разделов:

Первый раздел «Описание задачи» включает в себя анализ предметной области, описание существующих бизнес-процессов и задач, анализ аналогов и их недостатков, обоснование необходимости внедрения разрабатываемого веб-приложения.

Второй раздел «Проектирование веб-приложения» описывает архитектуру системы, проектирование базы данных с таблицами, связями и атрибутами, концептуальную модель и структуру пользовательского интерфейса.

Третий раздел «Реализация веб-приложения» детально раскрывает процесс разработки клиентской и серверной частей, описывает физическую и логическую структуру, компоненты и модули, их взаимодействие, а также результаты проведенных тестирований.

Четвертый раздел «Применение» содержит описание возможных сценариев использования веб-приложения, примеры его работы в реальных условиях и рекомендации по его внедрению.

В заключении проекта подводятся итоги проделанной работы, формулируются основные результаты и выводы, полученные в ходе анализа и разработки веб-приложения для управления проектами. Обсуждаются возможные направления дальнейшего развития системы и рекомендации по её совершенствованию.

Заключение подводит итоги работы, анализирует степень достижения целей, оценивает качество и эффективность разработанного веб-приложения.

В приложение А содержит исходные тексты модулей веб-приложения.

В приложении Б представлены результаты работы веб-приложения.

1. Описание задачи
   1. Анализ предметной области

Предметной областью решаемой задачи является подразделение фронтэнд-разработки компании ООО «ФП Трэйд», занимающейся разработкой заказного программного. Внутри подразделения осуществляется распределение и контроль выполнения задач между сотрудниками проектной команды, работающей над созданием интерфейсных решений для финансовых и страховых систем. Основной целью подразделения является разработка удобных, адаптивных и эффективных пользовательских интерфейсов с учетом современных требований к пользовательскому опыту и техническим характеристикам. [1]

Работа сотрудников подразделения начинается с получения задач, которые формируются на основе требований заказчиков и спецификаций от бизнес-аналитиков. Задачи включают разработку новых функциональных модулей, оптимизацию существующего кода, устранение выявленных ошибок, адаптацию интерфейсов под различные устройства и браузеры. Тимлид осуществляет распределение задач между старшими, средними и младшими разработчиками, контролирует сроки и качество исполнения. При необходимости задачи могут быть перенаправлены или уточнены бизнес-аналитиком, DevOps-инженером или QA-инженером для детальной проработки или тестирования.

Для организации эффективного управления и контроля задач подразделение использует методологию Agile с Align-досками, которые визуализируют текущие задачи, стадии их выполнения и ответственными исполнителями. Align-доска представляет собой инструмент, на котором задачи отображаются в виде карточек, перемещаемых между различными статусами («Новая задача», «В работе», «На проверке», «Завершено»). Это позволяет всем членам команды оперативно видеть текущий статус выполнения задач, выявлять узкие места, своевременно реагировать на изменения и приоритезировать работу.



Рисунок 1.1 – Пример Align-доски

Использование веб-приложения для автоматизации распределения и управления задачами обеспечит:

* централизовать хранение и доступ к данным по задачам и проектам;
* автоматизировать планирование и распределение работ;
* визуализировать жизненный цикл задач и заказов;
* обеспечить контроль сроков, включая интеграцию норм времени;
* упростить работу менеджеров за счёт встроенных фильтров, поиска, отчётности;
* повысить прозрачность и подотчётность сотрудников через визуализацию задач;
* экспортировать отчёты в форматы, пригодные для внутренней или внешней отчётности.
  1. Постановка задачи

В современных компаниях процесс планирования и распределения задач между сотрудниками зачастую организован неэффективно. Это приводит к потере времени, снижению производительности, а также усложнению управления проектами и контроля выполнения задач. На практике распределение задач часто осуществляется вручную – через устные указания, электронную почту или мессенджеры, что делает процесс непрозрачным и подверженным ошибкам. Такие методы не позволяют полноценно контролировать сроки, равномерность загрузки сотрудников и качество исполнения.

В некоторых организациях задачи ведутся с помощью табличных редакторов (например, Microsoft Excel или Google Таблицы), однако данный подход не обеспечивает должного уровня автоматизации и сильно ограничивает возможности анализа, фильтрации и адаптации под регламенты. Отсутствие централизованной системы, обеспечивающей управление проектами с учётом норм времени и типовых перечней задач, приводит к дублированию работы, потере данных, затруднениям в планировании и невозможности построения статистических отчётов по результатам деятельности.

Ключевыми проблемами существующих подходов являются:

* низкая прозрачность процессов исполнения задач;
* отсутствие механизма учета и контроля норм времени;
* неравномерное распределение задач между сотрудниками;
* высокая трудоёмкость при подготовке отчетов;
* сложности с анализом эффективности выполнения задач и загруженности сотрудников;
* невозможность быстрой адаптации системы под конкретные бизнес-процессы.

Многие популярные решения, такие как Trello, Jira, Asana, частично решают проблему, однако они не адаптированы под узкоспециализированные потребности. Например, Trello предоставляет визуальное представление задач, но не реализует автоматическое распределение с учётом норм времени и регламентов. Jira ориентирована на масштабные ИТ-проекты и требует сложной конфигурации, что делает её неудобной и избыточной для небольших команд. Asana, несмотря на развитые механизмы трекинга, также не всегда предоставляет нужный уровень адаптивности для индивидуальных бизнес-процессов, включая управление временем и отчётность по нормативам.

Разрабатываемое в рамках дипломного проекта веб-приложение призвано устранить перечисленные недостатки и автоматизировать процессы управления проектами с учётом специфики реальных производственных условий. Система будет выполнять следующие задачи:

* вести централизованный журнал задач по каждому проекту с привязкой к регламентированному перечню работ;
* учитывать нормы времени на выполнение задач по каждому типу проекта;
* обеспечивать возможность назначения задач на сотрудников с учётом их роли, занятости и квалификации;
* реализовать визуализацию задач в виде Kanban-доски со статусами: «Новая», «В работе», «Завершена», «Выполнена»;
* обеспечить возможность изменения статуса задач через механизм drag-and-drop;
* автоматизировать контроль сроков исполнения заказов и задач;
* учитывать фактическое и плановое время выполнения для анализа продуктивности;
* реализовать систему фильтрации, поиска и сортировки информации по проектам, задачам, статусам, сотрудникам;
* формировать отчёты по задачам в разрезе проектов, сотрудников, периодов, с возможностью экспорта в XLSX/PDF;
* внедрить ролевую модель с разделением прав доступа: менеджеры получают доступ ко всем проектам, сотрудники – только к своим задачам.

Таким образом, система позволит не только улучшить распределение задач, но и обеспечит высокий уровень управляемости, прозрачности и отчётности, что особенно важно при работе по проектной модели с жестко заданными сроками и нормами. Благодаря централизованному хранению данных, визуализации процессов и автоматизированной отчётности, приложение станет надёжным инструментом для оптимизации производственной деятельности и принятия управленческих решений.

1. Проектирование веб-приложения
   1. Проектирование модели

Проектирование модели веб-приложения является ключевым этапом разработки, на котором формируется абстрактное представление системы, ее функциональности, архитектуры и взаимодействия компонентов. Этот процесс включает анализ требований, определение ролей пользователей, моделирование логики взаимодействия и структуры данных. Грамотное проектирование минимизирует ошибки на этапе реализации, повышает удобство использования и обеспечивает масштабируемость системы.

Проектирование веб-приложения для управления проектами и задачами в компании включает комплексное моделирование с использованием унифицированного языка моделирования (UML). Модель охватывает функциональные возможности, такие как управление пользователями, заказами, задачами, командами, а также отслеживание статуса выполнения и генерация отчетов. Основные роли пользователей:

* менеджер (Администратор): управляет пользователями, командами, заказами, задачами, назначает сотрудников, контролирует выполнение и генерирует отчеты;
* сотрудник: просматривает назначенные задачи, обновляет их статус, фиксирует время выполнения и добавляет комментарии.
* система: отвечает за автоматические уведомления, расчет времени выполнения и контроль сроков.

Диаграмма вариантов использования предоставлена на Рисунке 1 в графической части, она описывает взаимодействие актёров с системой. Основные кейсы включают:

* менеджер может входить в систему и регистрировать новых пользователей, создавать, изменять или удалять заказы и задачи, назначать задачи сотрудникам и управлять командами. Также менеджер просматривает и экспортирует отчеты, например, через представление OrdersWithProjectTypes, следит за статусом задач и контролирует соблюдение сроков;
* сотрудник входит в систему, просматривает свои задачи через представление TasksByStatus, обновляет их статус, фиксирует время выполнения в таблице Execution и добавляет комментарии к задачам;
* система отправляет уведомления о дедлайнах, рассчитывает время выполнения задач через представление EmployeeTaskExecution и обеспечивает целостность данных.

Каждый кейс включает подзадачи и расширения. Например, кейс «Просмотр задач» включает функции «Обновление статуса» и «Добавление комментариев». Кейсы «Регистрация» и «Авторизация» являются обязательными для доступа к функционалу.

Концептуальная модель «Сущность-связь» предоставлена на Рисунке 2 в графической части, она описывает основные сущности и их связи:

* Roles (ID\_Role, Role\_Name): роли пользователей (например, Менеджер, Сотрудник).
* ProjectTypes (ID\_ProjectType, Type\_Name, Description): типы проектов.
* Users (ID\_User, First\_Name, Last\_Name, Phone, Email, Password, ID\_Role): пользователи системы.
* Statuses (ID\_Status, Status\_Name): статусы задач (Новая, В работе, Завершена).
* Teams (ID\_Team, Team\_Name): команды сотрудников.
* Orders (ID\_Order, ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager): заказы, связанные с проектами.
* Tasks (ID\_Task, ID\_Order, Task\_Name, Description, Time\_Norm, ID\_Status): задачи, привязанные к заказам.
* Assignment (ID\_Assignment, ID\_Task, ID\_Employee, Assignment\_Date): назначение задач сотрудникам.
* Execution (ID\_Execution, ID\_Task, ID\_Employee, Start\_Date, End\_Date): выполнение задач.
* TeamMembers (ID\_Member, ID\_Team, ID\_User, Role): участники команд.

**Связи**:

* один пользователь может иметь одну роль (Users → Roles);
* один заказ связан с одним типом проекта и одной командой (Orders → ProjectTypes, Teams);
* один заказ содержит множество задач (Orders → Tasks).
* одна задача может быть назначена одному сотруднику (Tasks → Assignment → Users);
* одна задача имеет записи о выполнении (Tasks → Execution);
* один пользователь может быть участником нескольких команд (Users → TeamMembers → Teams).

Логическая модель предоставлена на Рисунке 3 в графической части соответствует третьей нормальной форме (3NF) и реализована в виде таблиц, описанных в SQL-скрипте:

* Roles: ID\_Role (PK), Role\_Name (UNIQUE);
* ProjectTypes: ID\_ProjectType (PK), Type\_Name, Description;
* Users: ID\_User (PK), First\_Name, Last\_Name, Phone, Email (UNIQUE), Password, ID\_Role (FK → Roles);
* Statuses: ID\_Status (PK), Status\_Name (UNIQUE);
* Teams: ID\_Team (PK), Team\_Name;
* Orders: ID\_Order (PK), ID\_ProjectType (FK → ProjectTypes), ID\_Team (FK → Teams), Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager (FK → Users);
* Tasks: ID\_Task (PK), ID\_Order (FK → Orders), Task\_Name, Description, Time\_Norm, ID\_Status (FK → Statuses);
* Assignment: ID\_Assignment (PK), ID\_Task (FK → Tasks), ID\_Employee (FK → Users), Assignment\_Date;
* Execution: ID\_Execution (PK), ID\_Task (FK → Tasks), ID\_Employee (FK → Users), Start\_Date, End\_Date;
* TeamMembers: ID\_Member (PK), ID\_Team (FK → Teams), ID\_User (FK → Users), Role.

Внешние ключи обеспечивают целостность данных. Представления (TasksByStatus, OrdersWithProjectTypes, EmployeeTaskExecution) упрощают доступ к данным для отчетов и интерфейсов.

Диаграмма классов предоставлена на Рисунке 4 в графической части, она описывает объектно-ориентированную структуру системы. Классы соответствуют сущностям базы данных, а их методы отражают функционал:

* User: методы authorize(), register(), viewTasks(), updateTaskStatus(), addComment().
* Order: методы create(), update(), delete(), getTasks();
* Task: методы assign(), updateStatus(), trackExecution();
* Team: методы addMember(), removeMember(), getOrders();
* Execution: методы start(), complete(), calculateHours().

Ассоциации между классами соответствуют связям в концептуальной модели. Например, класс User имеет ассоциацию «один-ко-многим» с Assignment и TeamMembers.

Диаграмма развертывания предоставлена на Рисунке 5 в графической части, она описывает физическую архитектуру системы:

* клиентский уровень: веб-браузер (React, Type Script) для доступа к интерфейсу;
* сервер приложений: сервер (Node.js) с REST API для обработки запросов;
* сервер базы данных: СУБД (Microsoft SQL Server) с базой ProjectManagementDB;
* внешние сервисы: SMTP-сервер для уведомлений, облачное хранилище для отчетов.

Компоненты взаимодействуют через HTTP/HTTPS (клиент-сервер) и JDBC/ODBC (сервер-БД). Масштабируемость обеспечивается возможностью добавления серверов приложений и репликации базы данных.

* 1. Требования к веб-приложению

Для обеспечения удобства и эффективности использования веб-приложения, предназначенного для распределения задач между сотрудниками компании, определены следующие требования:

Требования к стилистическому оформлению: веб-приложение будет выполнено в современном минималистичном стиле, чтобы пользователи могли легко и быстро взаимодействовать с системой без отвлекающих факторов.

Требования к графическому дизайну: графический дизайн веб-приложения будет ориентирован на минимализм и функциональность. Использование иконок и графических элементов будет лаконичным и понятным, что позволит сократить время на восприятие информации.

Требования к шрифтовому оформлению: основным шрифтом выбран шрифт «Roboto» благодаря его высокой читаемости и универсальности, хорошо подходящий для интерфейсов веб-приложений. Основные размеры шрифтов будут:

* заголовки: 22 px;
* подзаголовки: 18 px;
* основной текст: 14 px.

Требования к средствам просмотра: приложение должно корректно отображаться и стабильно работать в современных браузерах, таких как Google Chrome, Mozilla Firefox, Safari и Microsoft Edge (актуальные версии и версии последних двух лет). Также должна быть обеспечена адаптивность и корректное отображение на устройствах с различными разрешениями экранов (мобильные телефоны, планшеты, настольные компьютеры).

Требования к контенту (наполнению): контент страниц будет структурированным, лаконичным и простым для восприятия, тексты написаны ясным и доступным языком без использования сложной профессиональной терминологии.

Требования к компоновке страниц:

* страницы выполнены в едином стиле для всех разделов веб-приложения;
* элементы интерфейса должны быть выровнены по единой сетке;
* основные блоки информации на страницах четко структурированы и легко различимы;
* навигация должна быть простой и интуитивно понятной;
* кнопки и другие интерактивные элементы должны иметь четкие границы, легко различаться и быть выделенными визуально для удобства пользователей.
  1. Структура веб-приложения

Структура веб-приложения представляет собой организацию его компонентов и модулей, обеспечивающую удобное взаимодействие пользователей с системой. Четкая структура упрощает навигацию и позволяет быстро находить нужные функции и разделы.

Информационная структура веб-приложения для управления проектами включает главную страницу с авторизацией и регистрацией, личный кабинет менеджера и личный кабинет сотрудника. Личный кабинет менеджера предоставляет функции управления проектами и задачами (создание, редактирование и удаление заказов и задач), распределения задач между сотрудниками с учетом норм времени, ведения журнала задач, мониторинга жизненного цикла заказов и контроля сроков выполнения. Также менеджер может формировать и экспортировать отчеты, включая статистические отчеты по количеству задач за период или по типу проекта, отчеты по задачам сотрудника и отчеты о выполнении заказов. Личный кабинет сотрудника позволяет просматривать назначенные задачи на Kanban-доске с разбивкой по статусам («Новая», «В работе», «Завершена», «Выполнена»), изменять статусы задач перетаскиванием, фиксировать время выполнения и добавлять комментарии. Общие компоненты приложения включают систему навигации, поиск и фильтрацию данных, уведомления о дедлайнах и автоматическое формирование отчетов.

Структура веб-приложения отображена на ментальной карте на рисунке 2.2.



Рисунок 2.2 – Ментальная карта структуры веб-приложения

* 1. Проектирование макета веб-приложения

Проектирование веб-приложения для распределения задач между сотрудниками компании начинается с определения визуального стиля и логической структуры интерфейса. Основной целью является создание удобного, интуитивно понятного и функционального дизайна, который удовлетворяет современным требованиям usability. Выбранный стиль минималистичен и функционален, с чистым и простым интерфейсом, избегая избыточных графических элементов для ускорения восприятия и удобства работы пользователей. Простая и понятная навигация помогает быстро получить доступ к ключевым функциям системы как для сотрудников, так и для администраторов.

Для визуализации структуры веб-приложения были разработаны макеты страниц и форм с использованием онлайн-сервиса Figma, который предназначен для проектирования интерфейсов и прототипирования. Figma позволяет совместно работать над дизайном в реальном времени, создавать интерактивные макеты веб-приложений, а также тестировать маршруты пользователей и взаимодействие элементов.

Основные экраны включают страницу авторизации и регистрации с возможностью создания учетных записей и назначения ролей (администратор, сотрудник).

Макет страницы авторизации предоставлена на рисунке 2.1.

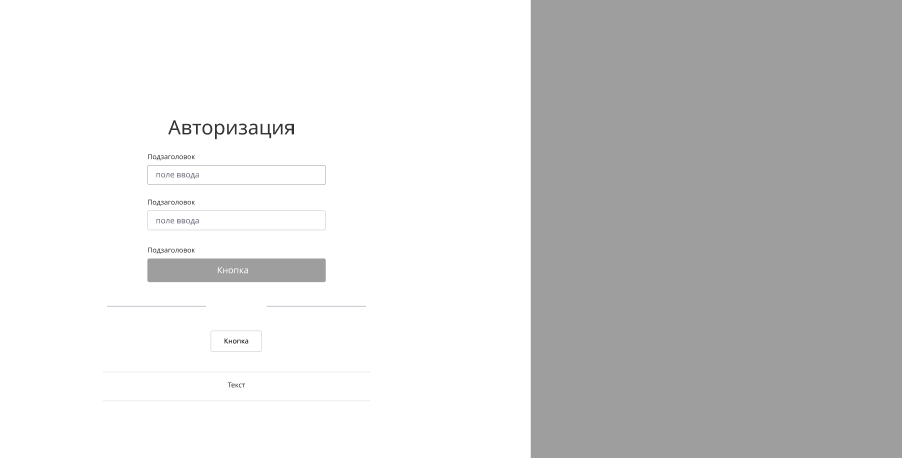


Рисунок 2.1 – Макет страницы авторизации

Панель сотрудника содержит страницы для просмотра задач, изменения статуса задач, добавления комментариев и ведения учета рабочего времени.

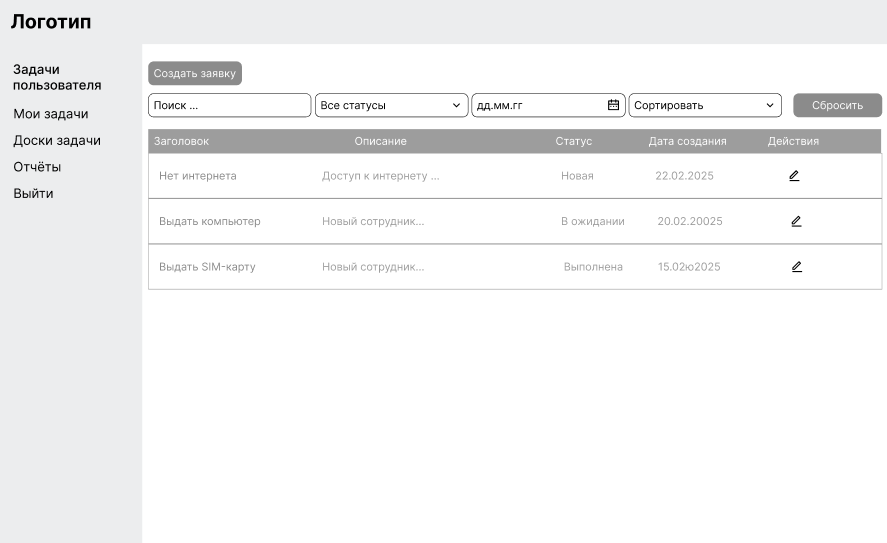


Рисунок 2.2 – Макет страницы «Задачи пользователя»

Панель администратора обеспечивает управление учетными записями, задачами и контроль выполнения заданий сотрудниками, а также формирование и экспорт отчетов.

Макет страницы добавления задачи для сотрудника со стороны администратора показано на рисунке 2.3.

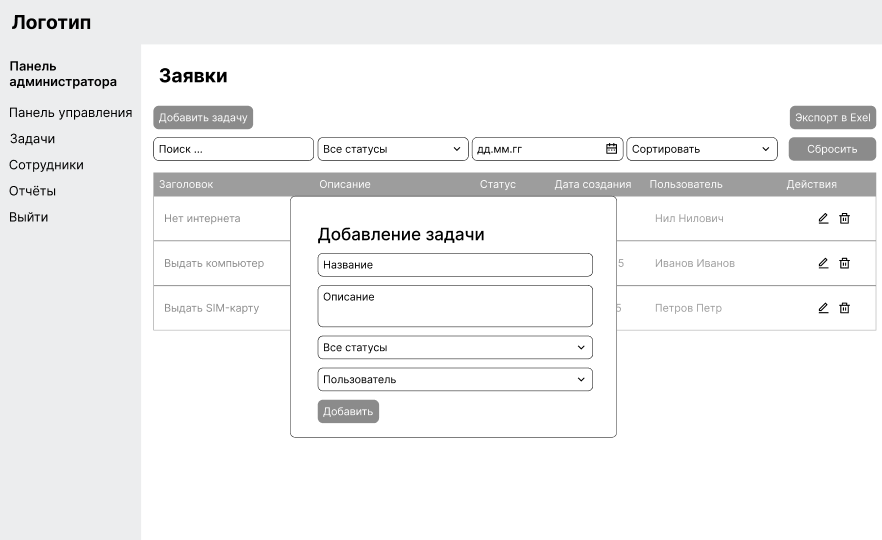


Рисунок 2.3 – Макет страницы «Задачи пользователя»

Страница с отчетами для администратора включает две таблицы с подробными данными о работе сотрудников и три диаграммы (гистограмма, график и круговая диаграмма) для визуализации данных. Таблицы отчетов предусматривают экспорт в Word.

Макет страницы отчётов предоставлен на рисунке 2.4.

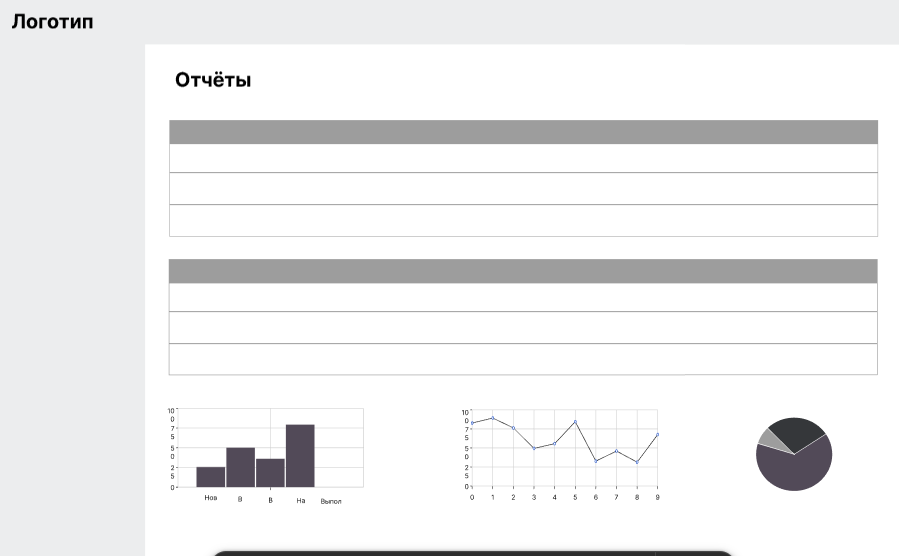


Рисунок 2.4 – Макет страницы «Отчёты»

Макеты, представленные в приложении, четко демонстрируют структуру интерфейса, логику навигации и способы взаимодействия пользователей с элементами системы.

* 1. Программно-технические средства, необходимые для разработки приложения

Разработка веб-приложения требует использования современных технологий и инструментов, обеспечивающих высокую производительность, безопасность, масштабируемость и удобство работы. Выбор программно-технических средств обоснован требованиями проекта, включая поддержку многопользовательского взаимодействия, защиту данных, гибкость в управлении задачами и удобство использования интерфейса.

Для реализации веб-приложения выбраны **современные стековые технологии**, позволяющие создать удобный пользовательский интерфейс и надежную серверную часть. Основные критерии выбора технологий включают:

* совместимость с различными устройствами и кроссбраузерность, что позволяет корректно отображать веб-приложение в актуальных версиях браузеров;
* гибкость и масштабируемость, обеспечивающие возможность дальнейшего развития и модернизации приложения;
* простота интеграции с внешними сервисами, такими как базы данных, системы аутентификации и API.

Разработка веб-приложения осуществляется на основе **клиент-серверной архитектуры**, где фронтенд отвечает за отображение пользовательского интерфейса, а бэкенд – за обработку данных и бизнес-логику.

Для разработки пользовательского интерфейса используются HTML, CSS, JavaScript, React.js и TypeScript. Данные технологии обеспечивают:

* структурированную разметку страниц с помощью HTML;
* гибкую стилизацию и адаптивный дизайн с использованием CSS;
* динамическое обновление интерфейса и интерактивность с помощью JavaScript;
* компонентную архитектуру и управление состоянием с React.js и Redux [3];
* повышенную безопасность и удобство разработки с использованием TypeScript.

Серверная часть реализуется с использованием Node.js и Express.js, что позволяет:

* эффективно обрабатывать HTTP-запросы и управлять маршрутами;
* реализовать REST API для взаимодействия между клиентом и сервером;
* обеспечить масштабируемость за счет асинхронной обработки запросов [2].

Для хранения информации о пользователях, задачах и заказах используется **SSMS**, что обеспечивает:

* соответствие требованиям третьей нормальной формы (3НФ);
* гибкость работы с реляционными данными;
* высокую скорость обработки запросов благодаря индексированию и оптимизированным операциям.

**Контроль версий и тестирование:**

* Git и GitHub используются для управления кодом и ведения истории изменений;
* Postman применяются для автоматизированного тестирования функционала и проверки работы API.

Таким образом, выбранный стек технологий позволяет создать **интерактивное, надежное и масштабируемое веб-приложение**, соответствующее современным требованиям разработки и удобству использования.

* 1. Защита и сохранность данных

Разработка веб-приложения для распределения задач между сотрудниками компании требует особого внимания к вопросам безопасности данных. Важнейшими аспектами являются контроль доступа к информации, защита от несанкционированного использования и обеспечение надежного хранения данных.

Для предотвращения несанкционированного доступа реализована **многоуровневая система авторизации и аутентификации** пользователей. Доступ к данным зависит от роли пользователя:

* администратор имеет полный доступ к управлению пользователями, задачами, заказами и отчетами;
* сотрудник может просматривать и изменять только назначенные ему задачи, а также управлять своим рабочим временем.

Дополнительно реализованы механизмы **контроля сессий**, предотвращающие доступ к системе с неавторизованных устройств. В случае длительного бездействия пользователя система автоматически завершает сеанс.

Для обеспечения целостности и сохранности информации реализована **система резервного копирования**, включающая:

* автоматическое ежедневное создание резервных копий базы данных, которые хранятся в защищенном облачном хранилище;
* разделение прав доступа к резервным копиям, что позволяет только администраторам восстанавливать данные;
* мониторинг целостности данных, который включает контроль за изменениями в базе данных и оперативное восстановление в случае сбоев.

Кроме того, предусмотрено **разграничение уровней доступа на уровне базы данных**, что исключает возможность удаления или изменения критически важных данных без соответствующих привилегий.

* 1. Организация и введение информационной базы (модели)

Для управления проектами и задачами в веб-приложении используется реляционная база данных ProjectManagementDB, реализованная в Microsoft SQL Server. База данных включает следующие таблицы:

* Statuses – содержит перечень возможных статусов задач, таких как «Новая», «В работе», «Завершена», «Выполнена».
* Roles – хранит перечень ролей пользователей, таких как менеджер, сотрудник, разработчик и другие.
* ProjectTypes – содержит информацию о типах проектов, например, разработка ПО или дизайн.
* Teams – хранит информацию о командах, участвующих в проектах.
* Orders – содержит данные о заказах (проектах), включая их тип, команду, сроки и статус.
* Tasks – хранит информацию о задачах, привязанных к заказам, с указанием норм времени и статуса.
* Assignment – фиксирует назначение задач сотрудникам.
* Execution – хранит данные о выполнении задач, включая даты начала и завершения.
* TeamMembers – содержит информацию о составе команд и ролях участников.

Таблица «Users» хранит информацию о сотрудниках компании. Структура таблицы представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы «Users»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_User | INT | 4 | Уникальный идентификатор пользователя |
| First\_Name | NVARCHAR(50) | 100 | Имя пользователя |
| Last\_Name | NVARCHAR(50) | 100 | Фамилия пользователя |
| Phone | NVARCHAR(20) | 40 | Контактный телефон |
| Email | NVARCHAR(100) | 200 | Адрес электронной почты (уникальный) |
| Password | NVARCHAR(255) | 510 | Хэш пароля |
| ID\_Role | INT | 4 | Ссылка на роль (FK → Roles) |

Таблица «Orders» хранит информацию о задачах и заказах. Структура таблицы представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы «Orders»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Order | INT | 4 | Уникальный идентификатор заказа |
| ID\_ProjectType | INT | 4 | Ссылка на тип проекта (FK → ProjectTypes) |
| ID\_Team | INT | 4 | Ссылка на команду (FK → Teams) |
| Order\_Name | NVARCHAR(100) | 200 | Название заказа |
| Creation\_Date | DATE | 3 | Дата создания заказа |
| End\_Date | DATE | 3 | Дата завершения заказа |
| Status | NVARCHAR(50) | 100 | Статус заказа |
| ID\_Manager | INT | 4 | Ссылка на менеджера (FK → Users) |

Таблица «Statuses» хранит возможные статусы задач. Структура таблицы представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «Statuses»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_User | INT | 4 | Уникальный идентификатор пользователя |
| First\_Name | NVARCHAR(50) | 100 | Имя пользователя |
| Last\_Name | NVARCHAR(50) | 100 | Фамилия пользователя |
| Phone | NVARCHAR(20) | 40 | Контактный телефон |
| Email | NVARCHAR(100) | 200 | Адрес электронной почты (уникальный) |
| Password | NVARCHAR(255) | 510 | Хэш пароля |
| ID\_Role | INT | 4 | Ссылка на роль (FK → Roles) |

Таблица «ProjectTypes» хранит типы проектов. Структура таблицы представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы «ProjectTypes»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_ProjectType | INT | 4 | Уникальный идентификатор типа. |
| Type\_Name | NVARCHAR(100) | 200 | Название типа проекта. |
| Description | NVARCHAR(255) | 510 | Описание типа проекта. |

Таблица «Roles» хранит перечень ролей пользователей. Структура таблицы представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы «Roles»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Role | INT | 4 | Уникальный идентификатор роли |
| Role\_Name | NVARCHAR(50) | 100 | Название роли (уникальное) |

Таблица «Teams» хранит информацию о командах. Структура таблицы представлена в 2.6.

Таблица 2.6 – Структура таблицы «Teams»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Team | INT | 4 | Уникальный идентификатор команды |
| Team\_Name | NVARCHAR(100) | 200 | Название команды |

Таблица «Tasks» хранит информацию о задачах. Структура таблицы представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы «Task»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Task | INT | 4 | Уникальный идентификатор задачи |
| ID\_Order | INT | 4 | Ссылка на заказ (FK → Orders) |
| Task\_Name | NVARCHAR(100) | 200 | Название задачи |
| Description | NVARCHAR(255) | 510 | Описание задачи |
| Time\_Norm | INT | 4 | Норма времени на задачу (часы) |
| ID\_Status | INT | 4 | Ссылка на статус (FK → Statuses) |

Таблица «Assignment» фиксирует назначение задач сотрудникам. Структура таблицы представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы «Assignment»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Assignment | INT | 4 | Уникальный идентификатор назначения |
| ID\_Task | INT | 4 | Ссылка на задачу (FK → Tasks) |

Продолжение таблицы 2.8 – Структура таблицы «Assignment»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Employee | INT | 4 | Ссылка на сотрудника (FK → Users) |
| Assignment\_Date | DATE | 3 | Дата назначения |

Таблица «Execution» хранит данные о выполнении задач. Структура таблицы представлена в таблице 2.9.

Таблица 2.9 – Структура таблицы «Execution»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Execution | INT | 4 | Уникальный идентификатор выполнения |
| ID\_Task | INT | 4 | Ссылка на задачу (FK → Tasks) |
| ID\_Employee | INT | 4 | Ссылка на сотрудника (FK → Users) |
| Start\_Date | DATETIME | 8 | Дата и время начала выполнения |
| End\_Date | DATETIME | 8 | Дата и время завершения выполнения |

Таблица «TeamMembers» хранит информацию о составе команд. Структура таблицы представлена в 2.10.

Таблица 2.10 – Структура таблицы «Teams»

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя поля** | **Тип поля** | **Размер поля, байт** | **Описание поля** |
| ID\_Member | INT | 4 | Уникальный идентификатор участника |
| ID\_Team | INT | 4 | Ссылка на команду (FK → Teams) |
| ID\_User | INT | 4 | Ссылка на пользователя (FK → Users) |
| Role | NVARCHAR(100) | 200 | Роль участника в команде |

Модели данных делятся на логические и физические. Логическая модель определяет структуру данных независимо от их физического хранения, включая сущности, атрибуты и отношения.

Физическая модель базы данных разрабатываемого веб-приложения представлена на рисунке 2.5

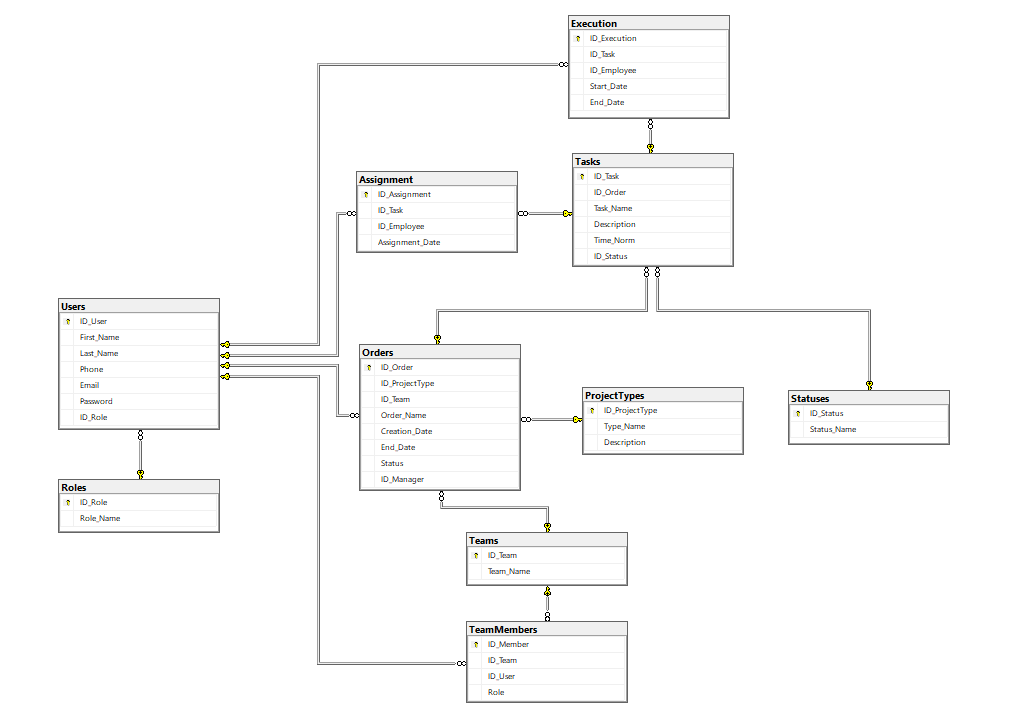


Рисунок 2.6 – Физическая схема базы данных

Код базы данных предоставлен в Приложении А.

1. Реализация веб-приложения

3.1 Описание разделов веб-приложения

Главная страница веб-приложения представлена в двух частях: сторона администратора и сторона сотрудника, каждая с собственным интерфейсом и навигационным меню, расположенным слева.

Перед входом осуществляется авторизация и регистрация пользователя для определения администратор он или обычный пользователь.

Сторона администратора включает в себе таблицу задач с колонками «Заголовок», «Описание», «Статус», «Дата создания», «Пользователь» и «Действия». В таблице отображаются задачи с данными, такими как название, описание, статус («На проверке», «В процессе», «Решена», «Просрочена»), дата создания, ответственный пользователь, а также кнопки для редактирования и удаления. Над таблицей расположены кнопки «Создать задачу» для добавления новой задачи, «Экспорт в Excel» для выгрузки данных в Microsoft Excel, фильтры по статусу и дате, а также строка поиска. Навигационное меню содержит четыре пункта:

* «Задачи»: открывает основную страницу с таблицей всех заявок, где администратор может просматривать детали, редактировать (изменять статус или описание, переназначить на другого пользователя) или удалять задачи, а также использовать фильтры и поиск для навигации;
* «Сотрудники»: открывает страницу с таблицей всех пользователей, где администратор может просматривать, редактировать или удалять информацию про них;
* «Отчеты»: открывает аналитический раздел со статистикой по задачам, представленной в виде графиков;
* «Выйти»: завершает сессию администратора, перенаправляя на страницу авторизации для повторного входа.

Сторона сотрудника представлена в виде досок задач с колонками «Новая», «В процессе», «На проверке», «В ожидании» и «Решена». Сотрудник может перетаскивать карточки между колонками, просматривать детали задачи при клике или создавать новую задачу через кнопку «Добавить задачу» над доской. Навигационное меню включает следующие пункты:

* «Мои задачи»: отображает основную страницу с задачами;
* «Доски задач»: отображает страницу с досками задач;
* «Отчеты»: открывает раздел, где отображаются данные в виде таблиц графиков;
* «Выйти»: завершает сессию, перенаправляя на страницу входа, где требуется повторно ввести логин и пароль.

3.2 Разработка административной части приложения

Логическая структура административной части веб-приложения для управления проектами представляет собой иерархическую систему, обеспечивающую управление проектами, задачами, сотрудниками и аналитикой. Основные компоненты включают модуль авторизации, который позволяет менеджерам и сотрудникам входить в систему, предоставляя доступ к функциям в зависимости от их роли; модуль управления проектами и задачами, обеспечивающий создание, редактирование, удаление заказов и задач, распределение задач между сотрудниками с учетом норм времени, а также поиск и фильтрацию по различным параметрам с интеграцией с базой данных; модуль управления сотрудниками, позволяющий менеджерам редактировать данные учетных записей и назначать роли; модуль отчетности, который формирует статистические отчеты по количеству задач за период, по типу проекта или по сотрудникам, с возможностью визуализации данных и экспорта в Excel или PDF; модуль выхода, завершающий сессию пользователя и перенаправляющий на страницу авторизации для обеспечения безопасности.

Графическое изображение в виде организационной диаграммы изображено на рисунке 3.1.

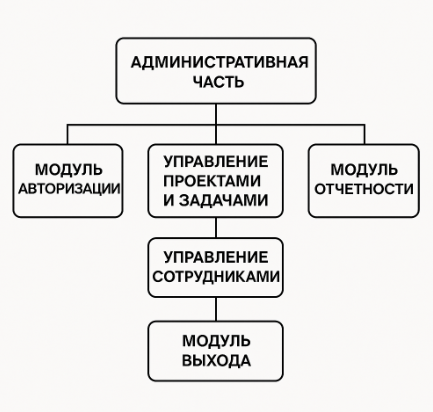


Рисунок 3.1 – Логическая структура административной части приложения

3.3 Разработка клиентской части приложения

Логическая структура клиентской части веб-приложения для управления проектами обеспечивает удобное взаимодействие сотрудников с системой для выполнения и отслеживания задач. Основные компоненты включают модуль авторизации, который позволяет сотрудникам входить в личный кабинет через страницу логина, проверяя учетные данные и предоставляя доступ к функционалу в зависимости от роли. Модуль управления задачами отображает задачи на Kanban-доске с разбивкой по статусам («Новая», «В работе», «Завершена», «Выполнена»), позволяет изменять статусы перетаскиванием, просматривать детали задач, добавлять комментарии и фиксировать время выполнения, а также поддерживает поиск, сортировку и фильтрацию задач. Модуль «Мои задачи» предоставляет сотруднику доступ к списку только его задач с возможностью просмотра деталей, изменения статуса и учета времени выполнения. Модуль отчетности формирует персонализированные статистические данные о задачах сотрудника, включая их количество, статусы и время выполнения, с визуализацией в виде таблиц и графиков для анализа производительности. Модуль выхода завершает сессию, очищает данные и перенаправляет на страницу авторизации, обеспечивая безопасность.

3.4 Описание используемых функций и процедур

Данный раздел описывает разработанные функции, связанные с элементами управления веб-приложения, а также программный код основных модулей с комментариями и указанием на элементы управления, которые инициируют выполнение данных функций.

**Элементы управления страницы регистрации:**

* поле ввода для имени;
* поле ввода для электронной почты;
* поле ввода для пароля;
* кнопка «Зарегистрироваться».

Функция для обработки регистрации пользователя:

function handleRegistration() {

const name = document.getElementById('nameInput').value;

const email = document.getElementById('emailInput').value;

const password = document.getElementById('passwordInput').value;

Проверка на пустые поля:

if (!name || !email || !password) {alert('Пожалуйста, заполните все поля.'); return;}

Отправка данных на сервер для регистрации:

fetch('/api/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ name, email, password }),

})

.then(response => {

if (response.ok) {

alert('Регистрация успешна! Вы можете войти в систему.');

window.location.href = '/login';

} else {alert('Ошибка регистрации. Пожалуйста, попробуйте еще раз.');}}) catch(error => {console.error('Ошибка:', error);});}

Привязка функции к кнопке «Зарегистрироваться»: document.getElementById('registerButton').addEventListener('click', handleRegistration);

Элементы управления страницы авторизации:

* поле ввода для электронной почты;
* поле ввода для пароля;
* кнопка «Продолжить».

Функция для обработки авторизации пользователя

function handleLogin() {

const email = document.getElementById('emailInput').value;

const password = document.getElementById('passwordInput').value;

Проверка на пустые поля:

if (!email || !password) {alert('Пожалуйста, заполните все поля.'); return;}

Отправка данных на сервер для авторизации:

fetch('/api/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ email, password }),})

.then(response => {

if (response.ok) {

alert('Вы успешно вошли в систему!');

window.location.href = '/dashboard';

} else {

alert('Ошибка входа. Проверьте ваши учетные данные.');}}) catch(error => {console.error('Ошибка:', error);});}

Привязка функции к кнопке «Продолжить»: document.getElementById('continueButton').addEventListener('click', handleLogin).

**Функция для перетаскивания карточек**

interface TaskCardProps {

task: Task;}

const TaskCard: React.FC<TaskCardProps> = ({ task }) => {

const [{ isDragging }, drag] = useDrag({

type: "task",

item: { id: task.id },

collect: (monitor) => ({

isDragging: monitor.isDragging(),

}),

});

**Полный код программы предоставлен в приложении А.**

3.5 Функциональное тестирование

Функциональное тестирование – это процесс проверки функций приложения на соответствие установленным требованиям, который выполняется для обнаружения ошибок и недочетов перед его запуском в эксплуатацию. Тестирование будет осуществляться последовательно, переходя от одной части программы к другой.

Предварительное условие входа для пользователя запустить программу и находиться в форме регистрации. Форма регистрации представлена в приложении Б на рисунке Б.1.

Таблица 3.1 – Функционал сотрудника.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / функция | Шаги воспроизведения | Результат |
| 1 | Регистрация нового аккаунта | Заполнить поля:   * «Имя» – Василий; * «Фамилия» – Волковец; * «Телефон» –« +375293130956»; * «Электронная почта» – vlk12345@gmail.com; * «Пароль» – 1234567vv; * «Проверка пароля» – 1234567vv.   Нажать на кнопку «Зарегистрироваться». | Ожидаемый результат. Успешная регистрация пользователя. Выдаётся модальное окно «Успех! Вы успешно зарегистрировались». |
| Фактический результат соответствует ожидаемому и представлен на рисунке Б1 в Приложении Б. |
| 2 | Авторизация | 1. Нажать на кнопку «Вход» на главном меню.  2. Заполнить поля данными:  – «Электронная почта» – vlk12345@gmail.com;  – «Пароль» – 1234567vv.  3. Нажать на кнопку «Продолжить». | Ожидаемый результат. Успешная авторизация пользователя. Выдаётся сообщение «Вы успешно вошли в систему». |
| Фактический результат соответствует ожидаемому и представлен в приложении Б на рисунке Б.2 в Приложении Б. |
| 43 | Изменение статуса задачи | 1. Перейти в раздел «Главная» через навигационное меню.  2. Найти задачу «Проблема с доступом к системе» в колонке «Новая».  3. Перетащить курсором карточку задачу в колонку «В процессе». | Ожидаемый результат. Задача перемещена в колонку «В процессе» на доске задач. |
| Фактический результат  Форма с перемещенной карточкой представлена на рисунке Б.3 в Приложении Б. |
| 4 | Просмотр задач | 1. Перейти в раздел «Мои задачи».  2. Просмотреть список задач.  3. Использовать фильтры по статусу «Новая», «В процессе», «Завершена». | Ожидаемый результат.  Задачи отображаются корректно в зависимости от статуса и фильтров. |
| Фактический результат  Форма с перемещенной карточкой представлена на рисунке Б.4 в Приложении Б. |
| 5 | Поиск задачи по названию | 1. Нажать фильтр «Поиск», расположенный над таблицей.  2. Нажать фильтр «Проблема с доступом». | Ожидаемый результат.  В системе появляется в результатах поиска только задача «Проблема с доступом». |

Продолжение таблицы 3.1 – Функционал сотрудника.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Фактический результат  Поиск по фильтрам представлен на рисунке Б.5 в Приложении Б. |
| 6 | Просмотр Kanban-доски задач | 1. Перейти в раздел «Доска задач».  2. Использовать фильтры по статусу задачи (например, «В процессе»). | Ожидаемый результат. Задача перемещена в колонку «В процессе» на доске задач. |
| Фактический результат  Форма с перемещенной карточкой представлена на рисунке Б.6 в Приложении Б. |

Предварительное условие: запустить программу со стороны менеджера (администратора). Личный кабинет со стороны менеджера представлена в приложении Б на рисунке Б.7.

Таблица 3.2 – функционал администратора.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / функция | Шаги воспроизведения | Результат |
| 7 | Создание нового проекта | 1. Перейти в раздел «Управление проектами».  2. Нажать на кнопку «Создать проект».  3. Заполнить поля:  - «Название проекта» – «Разработка ПО»;  - «Выбрать команду» – Команда1.  4. Нажать «Создать». | Ожидаемый результат. Проект успешно создан, появляется в списке проектов. |
| Фактический результат  Форма с перемещенной карточкой представлена на рисунке Б.8 в Приложении Б. |
| 8 | Распределение задач среди сотрудников | 1. Перейти в раздел «Распределение задач».  2. Выбрать задачу: «Проблема с доступом к системе» и нажать «Назначить сотруднику».  3. Выбрать сотрудника из списка: «Василий Волковец».  4. Нажать «Сохранить». | Ожидаемый результат. Задача успешно назначена сотруднику. |
| Фактический результат.  Успешно назначенная задача сотруднику предоставлена представлена рисунке Б.9 в Приложении Б. |
| 9 | Изменение статуса проекта | 1. Перейти в раздел «Проекты».  2. Выбрать проект, например, «Проблема с доступом к системе».  3. Изменить статус на «Завершен». | Ожидаемый результат. Статус успешно изменён. |
| Фактический результат  Отображение статуса представлено на рисунке Б.10 в Приложении Б. |
| 10 | Отчёт по задачам на проект | 1. Перейти в личный кабинет.  2. Выбрать вкладку «Мои отчёты». | Ожидаемый результат. Отчёты успешно отображаются. |
| Фактический результат  Отображение отчётов представлено на рисунке Б.11 в Приложении Б. |

Тест-кейсы для функционала администрирования и отчетности предоставлены в Таблице 3.2. Предварительное условие: зайти в личный кабинет со стороны менеджера (администратора) или со стороны сотрудника. Страница личного кабинета предоставлена в Приложении Б на рисунке Б.12.

Таблица 3.2 – Функционал экспорта.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль / функция | Шаги воспроизведения | Результат |
| 11 | Экспорт отчёта в Excel | 1. Перейти в раздел «Отчёты» сотрудника/менеджера.  2. Выбрать период для отчёта.  3. Нажать «Экспорт в Excel». | Ожидаемый результат. Отчёт успешно экспортирован в Excel. Файл скачивается на компьютер пользователя. |
| Фактический результат  Успешный экспорт предоставлен на рисунке Б.13 в Приложении Б. |
| 12 | Экспорт отчёта в PDF | 1. Перейти в раздел «Отчёты».  2. Выбрать нужный отчет по проекту.  3. Нажать «Экспорт в PDF». | Ожидаемый результат. Отчёт успешно экспортирован в PDF. Файл скачивается на компьютер пользователя. |
| Фактический результат.  Успешный экспорт предоставлен на рисунке Б.14 в Приложении Б. |

1. Применение

4.1 Назначение веб-приложения

Веб-приложение для управления проектами предназначено для автоматизации процессов планирования, распределения и контроля задач в компании. Оно обеспечивает эффективное управление проектами, включая ведение жизненного цикла заказов, назначение задач сотрудникам, учет рабочего времени и контроль сроков выполнения. Приложение поддерживает работу менеджеров и сотрудников, предоставляя персонализированные интерфейсы для управления проектами и задачами. Основные функции включают создание и распределение задач, отслеживание их статуса на Kanban-доске, формирование статистических отчетов и экспорт данных в форматы Word. Приложение повышает прозрачность рабочих процессов, улучшает коммуникацию между сотрудниками и командами, а также способствует повышению общей производительности компании. Целью разработки является создание удобного и надежного инструмента для управления проектами, который позволит автоматизировать рутинные процессы, такие как распределение задач и контроль сроков, а также предоставить менеджерам и сотрудникам доступ к актуальной информации о проектах. Приложение направлено на упрощение координации работы, минимизацию ошибок в планировании и обеспечение прозрачного мониторинга выполнения задач.

Веб-приложение имеет следующую структуру бэкенда проекта:

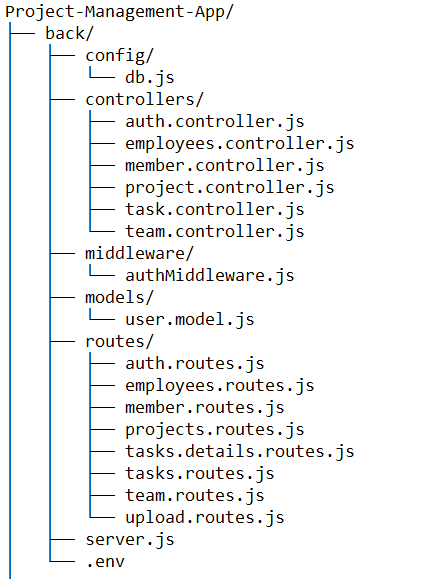


Рисунок 4.1 – Структура бэкенда приложения

Веб-приложение имеет следующую структуру фронтенда проекта:

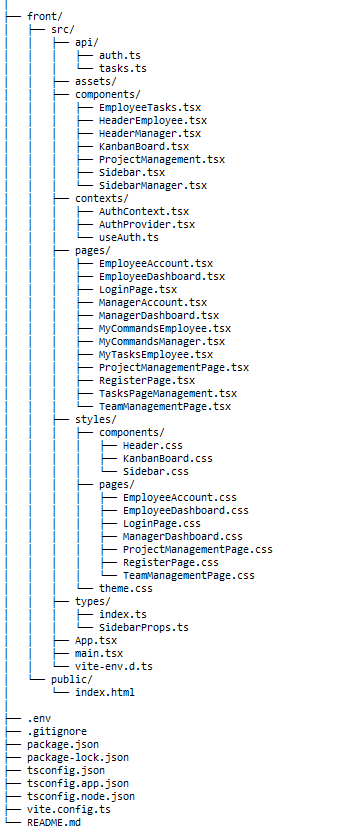


Рисунок 4.2 – Структура фронтенда приложения

Для установки веб-приложения нужно клонировать репозиторий с помощью команды git clone, установить зависимости серверной части командой npm install в директории back/ и клиентской части командой npm install в директории front/. Затем настроить переменные окружения в файлах .env, указав параметры подключения к базе данных и порт сервера, а также настроить базу данных ProjectManagementDB в Microsoft SQL Server с помощью SQL-скрипта.

Для запуска приложения запустить сервер из директории back/ командой npm run dev и клиентскую часть из директории front/ командой npm run dev для разработки или npm run build для развертывания. Приложение открывается в браузере по адресу, указанному в консоли, обычно <http://localhost:5173>.

Веб-приложение используется в компаниях для управления проектами и задачами в сферах IT, дизайна, маркетинга и других. Оно автоматизирует распределение задач с учетом норм времени, управляет жизненным циклом проектов, контролирует сроки, отображает задачи на Kanban-доске с перетаскиванием, создает статистические отчеты и упрощает коммуникацию через уведомления.

Для работы нужен стабильный интернет и современные браузеры (Chrome, Firefox, Edge, Safari). Приложение подходит для компаний с четкими ролями и проектами, разделенными на задачи. Требуется сервер с Node.js и Microsoft SQL Server. Для крупных компаний может потребоваться оптимизация базы данных и инфраструктуры.

4.2 Функциональное тестирование

Требования к серверу:

* Операционная система: Windows Server, Linux (Ubuntu, CentOS) или любая ОС, поддерживающая Node.js.
* Процессор: Минимум 2-ядерный процессор с частотой 2 ГГц (рекомендуется 4 ядра для высокой нагрузки).
* Оперативная память: Минимум 4 ГБ (рекомендуется 8 ГБ для стабильной работы).
* Дисковое пространство: Минимум 20 ГБ для базы данных и файлов приложения.

Программное обеспечение:

* VS Code – для открытия проекта;
* Node.js (версия 16 или выше) для серверной части;
* Microsoft SQL Server (2019 или новее) для хранения данных;
* npm для управления зависимостями;
* сеть. Статический IP-адрес и порт (например, 3000 для сервера) для доступа.

Требования к клиенту:

* устройства. Компьютеры, планшеты или смартфоны с поддержкой веб-браузеров;
* браузеры: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari (последние версии).
* операционная система: Windows 10/11, macOS, Linux, iOS, Android.
* оперативная память. Минимум 2 ГБ для комфортной работы браузера.
* интернет. Стабильное соединение со скоростью не менее 1 Мбит/с.
* разрешение экрана. Минимум 1024x768 для корректного отображения интерфейса (адаптивная верстка поддерживает меньшие разрешения).

Серверные требования обеспечивают стабильную работу Node.js и SQL Server, которые обрабатывают запросы и хранят данные. Высокая производительность процессора и достаточный объем оперативной памяти необходимы для обработки запросов от нескольких пользователей одновременно. Клиентские требования минимальны, так как приложение работает через браузер, а адаптивная верстка и оптимизированный фронтенд (React) гарантируют корректное отображение на различных устройствах. Современные браузеры обеспечивают поддержку JavaScript (ES6+) и CSS3, необходимых для работы приложения.

Заключение

В рамках дипломного проекта была выполнена разработка веб-приложения для управления проектами, соответствующего поставленным задачам. Программа практики полностью реализована: проведен анализ требований, спроектирована база данных, разработаны серверная и клиентская части, выполнен графический дизайн и адаптивная верстка. Приложение протестировано на стабильность и корректность работы в современных браузерах.

Задача разработки веб-приложения для управления проектами выполнена в полном объеме. Реализованы личные кабинеты для менеджеров и сотрудников с авторизацией, Kanban-доска с drag-and-drop функционалом, управление задачами и проектами, а также формирование и экспорт отчетов. База данных ProjectManagementDB, соответствующая третьей нормальной форме, обеспечивает надежное хранение данных о пользователях, задачах, заказах и командах. Функционал поиска, фильтрации и сортировки упрощает работу с данными, а уведомления о дедлайнах повышают контроль сроков.

Проектные решения полностью соответствуют требованиям задания. Реализованы все ключевые функции: управление жизненным циклом заказов, распределение задач, Kanban-доска, статистические отчеты и экспорт данных. Использование React [5] для фронтенда, Node.js с Express [8] для бэкенда и Microsoft SQL Server [9] для базы данных обеспечивает высокую производительность и масштабируемость. Адаптивная верстка гарантирует корректное отображение на разных устройствах, а тестирование подтвердило отсутствие критических ошибок.

Несоответствий требованиям задания не выявлено. Все функции, указанные в исходных данных, реализованы в полном объеме. В дальнейшем можно рассмотреть добавление функций, таких как интеграция с внешними сервисами (например, календарями или системами учета времени), но это выходит за рамки текущего задания.

В результате разработано веб-приложение, которое автоматизирует управление проектами, упрощает контроль задач и повышает эффективность работы компании. Приложение готово к внедрению и может быть масштабировано при необходимости.

# **Список использованных источников**

1. ООО «ФП Трэйд»: лидеры в разработке корпоративного ПО –Источник: Журнал «IT-Бизнес» №4. – Режим доступа: https://completesoft.ru. – Дата доступа: 2023 г.
2. Node.js учебное пособие по Visual Studio Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/docs/nodejs/nodejs-tutorial>. – Дата доступа: 17.03.2025.
3. Установка React [Электронный ресурс]. – Режим доступа: https://react.dev/learn/installation. – Дата доступа: 17.03.2025.
4. Онлайн-руководство по MongoDB [Электронный ресурс]. – Режим доступа: https://metanit.com/nosql/mongodb/. – Дата доступа: 03.01.2025.
5. React.js [Электронный ресурс]. – React, 2025. – Режим доступа: <https://react.dev/learn>. – Дата доступа: 01.03.2025.
6. TypeScript [Электронный ресурс]. – TypeScript, 2024. – Режим доступа: <https://www.typescriptlang.org/docs/>. – Дата доступа: 01.03.2025.
7. Node.js [Электронный ресурс]. – Node.js, 2024. – Режим доступа: <https://nodejs.org/en/docs>. – Дата доступа: 01.03.2025.
8. Express.js [Электронный ресурс]. – Express, 2024. – Режим доступа: <https://expressjs.com/en/guide/routing.html>. – Дата доступа: 01.03.2025.
9. Microsoft SQL Server [Электронный ресурс]. – Microsoft, 2025. – Режим доступа: <https://learn.microsoft.com/en-us/sql/sql-server>. – Дата доступа: 01.03.2025.

**ПРИЛОЖЕНИЕ А**

**(справочное)**

**Результаты работы приложения**

Код базы данных:

-- Установка формата даты для корректной интерпретации

SET DATEFORMAT ymd;

GO

-- Проверка текущих настроек даты

DBCC USEROPTIONS;

GO

-- Создание базы данных

IF NOT EXISTS (SELECT name FROM sys.databases WHERE name = 'ProjectManagementDB')

BEGIN

CREATE DATABASE ProjectManagementDB;

PRINT 'База данных ProjectManagementDB создана';

END

ELSE

BEGIN

PRINT 'База данных ProjectManagementDB уже существует';

END

GO

-- Переключение на базу данных

USE ProjectManagementDB;

GO

-- Создание таблиц

-- Таблица ролей

CREATE TABLE Roles (

ID\_Role INT PRIMARY KEY IDENTITY(1,1),

Role\_Name NVARCHAR(50) NOT NULL UNIQUE

);

GO

-- Таблица типов проектов

CREATE TABLE ProjectTypes (

ID\_ProjectType INT PRIMARY KEY IDENTITY(1,1),

Type\_Name NVARCHAR(100) NOT NULL,

Description NVARCHAR(255)

);

GO

-- Таблица пользователей

CREATE TABLE Users (

ID\_User INT PRIMARY KEY IDENTITY(1,1),

First\_Name NVARCHAR(50) NOT NULL,

Last\_Name NVARCHAR(50) NOT NULL,

Phone NVARCHAR(20) NOT NULL,

Email NVARCHAR(100) NOT NULL UNIQUE,

Password NVARCHAR(255) NOT NULL,

ID\_Role INT NOT NULL,

CONSTRAINT CHK\_Email CHECK (Email LIKE '%@gmail.com'),

CONSTRAINT FK\_Users\_Roles FOREIGN KEY (ID\_Role) REFERENCES Roles(ID\_Role)

);

GO

-- Таблица статусов

CREATE TABLE Statuses (

ID\_Status INT PRIMARY KEY IDENTITY(1,1),

Status\_Name NVARCHAR(50) NOT NULL UNIQUE

);

GO

-- Таблица команд

CREATE TABLE Teams (

ID\_Team INT PRIMARY KEY IDENTITY(1,1),

Team\_Name NVARCHAR(100) NOT NULL

);

GO

-- Таблица заказов

CREATE TABLE Orders (

ID\_Order INT PRIMARY KEY IDENTITY(1,1),

ID\_ProjectType INT NOT NULL,

ID\_Team INT NOT NULL,

Order\_Name NVARCHAR(100) NOT NULL,

Creation\_Date DATE NOT NULL,

End\_Date DATE,

Status NVARCHAR(50),

ID\_Manager INT,

FOREIGN KEY (ID\_ProjectType) REFERENCES ProjectTypes(ID\_ProjectType),

FOREIGN KEY (ID\_Team) REFERENCES Teams(ID\_Team),

FOREIGN KEY (ID\_Manager) REFERENCES Users(ID\_User)

);

GO

-- Таблица задач

CREATE TABLE Tasks (

ID\_Task INT PRIMARY KEY IDENTITY(1,1),

ID\_Order INT,

Task\_Name NVARCHAR(100) NOT NULL,

Description NVARCHAR(255),

Time\_Norm INT NOT NULL,

ID\_Status INT,

FOREIGN KEY (ID\_Order) REFERENCES Orders(ID\_Order),

FOREIGN KEY (ID\_Status) REFERENCES Statuses(ID\_Status)

);

GO

-- ПРЕДСТАВЛЕНИЕ ДЛЯ ЗАДАЧ

CREATE OR ALTER VIEW TasksFullDetails AS

SELECT

t.ID\_Task,

t.Task\_Name,

t.Description,

t.Time\_Norm,

s.Status\_Name,

o.Order\_Name,

tm.Team\_Name,

u.First\_Name + ' ' + u.Last\_Name AS Employee\_Name

FROM Tasks t

INNER JOIN Statuses s ON t.ID\_Status = s.ID\_Status

INNER JOIN Orders o ON t.ID\_Order = o.ID\_Order

INNER JOIN Teams tm ON o.ID\_Team = tm.ID\_Team

LEFT JOIN Assignment a ON t.ID\_Task = a.ID\_Task

LEFT JOIN Users u ON a.ID\_Employee = u.ID\_User;

GO

-- Таблица закрепления

CREATE TABLE Assignment (

ID\_Assignment INT PRIMARY KEY IDENTITY(1,1),

ID\_Task INT,

ID\_Employee INT,

Assignment\_Date DATE NOT NULL,

FOREIGN KEY (ID\_Task) REFERENCES Tasks(ID\_Task),

FOREIGN KEY (ID\_Employee) REFERENCES Users(ID\_User)

);

GO

-- Таблица выполнения

CREATE TABLE Execution (

ID\_Execution INT PRIMARY KEY IDENTITY(1,1),

ID\_Task INT,

ID\_Employee INT,

Start\_Date DATETIME,

End\_Date DATETIME,

FOREIGN KEY (ID\_Task) REFERENCES Tasks(ID\_Task),

FOREIGN KEY (ID\_Employee) REFERENCES Users(ID\_User)

);

GO

-- Таблица участников команд

CREATE TABLE TeamMembers (

ID\_Member INT PRIMARY KEY IDENTITY(1,1),

ID\_Team INT NOT NULL,

ID\_User INT NOT NULL,

Role NVARCHAR(100),

FOREIGN KEY (ID\_Team) REFERENCES Teams(ID\_Team),

FOREIGN KEY (ID\_User) REFERENCES Users(ID\_User)

);

GO

-- Вставка начальных данных

-- Роли

INSERT INTO Roles (Role\_Name) VALUES

('Менеджер'), ('Сотрудник'), ('Scrum Master'), ('Product Owner'), ('Разработчик'),

('Тестировщик'), ('Дизайнер UX/UI'), ('Аналитик'), ('DevOps-инженер'), ('Технический писатель'),

('Менеджер (Администратор)'), ('Тимлид'), ('Бизнес-аналитик'), ('Архитектор ПО'),

('Frontend-разработчик'), ('Backend-разработчик'), ('Fullstack-разработчик'),

('Системный администратор'), ('Специалист по безопасности'), ('Маркетолог'),

('HR-менеджер'), ('Координатор проектов');

GO

-- Типы проектов

INSERT INTO ProjectTypes (Type\_Name, Description) VALUES

('Разработка ПО', 'Создание программного обеспечения'),

('Дизайн', 'Разработка интерфейсов и графики');

GO

-- Пользователи

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'ivan\_manager@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Иван', 'Иванов', '+375291234567', 'ivan\_manager@gmail.com', '$2b$10$yx0X0uJd4gYPKsrYQM/KGOXEuYciDRhxEUOy.6UNbt1jqFMABHubC', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Менеджер'));

ELSE

UPDATE Users SET Phone = '+375291234567' WHERE Email = 'ivan\_manager@gmail.com';

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'petr\_employee@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Петр', 'Петров', '+375331234567', 'petr\_employee@gmail.com', '$2b$10$R4TkUYKRuNvwUsP6AMiqnepeo2uPnHduK1ZpuklBpKqpQ6uVleYaO', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Сотрудник'));

ELSE

UPDATE Users SET Phone = '+375331234567' WHERE Email = 'petr\_employee@gmail.com';

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'alex\_teamlead@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Алексей', 'Смирнов', '+375441234567', 'alex\_teamlead@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Тимлид'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'anna\_scrum@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Анна', 'Ковалёва', '+375291234568', 'anna\_scrum@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Scrum Master'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'dmitry\_po@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Дмитрий', 'Лебедев', '+375331234568', 'dmitry\_po@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Product Owner'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'olga\_dev@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Ольга', 'Морозова', '+375441234568', 'olga\_dev@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Разработчик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'vlad\_qa@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Владислав', 'Павлов', '+375251234567', 'vlad\_qa@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Тестировщик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'marina\_ui@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Марина', 'Соколова', '+375291234569', 'marina\_ui@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Дизайнер UX/UI'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'sergey\_analyst@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Сергей', 'Волков', '+375331234569', 'sergey\_analyst@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Аналитик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'ekaterina\_devops@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Екатерина', 'Бондарева', '+375441234569', 'ekaterina\_devops@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'DevOps-инженер'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'nikita\_writer@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Никита', 'Зайцев', '+375251234568', 'nikita\_writer@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Технический писатель'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'yulia\_admin@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Юлия', 'Козлова', '+375291234570', 'yulia\_admin@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Менеджер (Администратор)'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'pavel\_ba@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Павел', 'Григорьев', '+375331234570', 'pavel\_ba@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Бизнес-аналитик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'igor\_architect@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Игорь', 'Фомин', '+375441234570', 'igor\_architect@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Архитектор ПО'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'alena\_front@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Алёна', 'Лазарева', '+375251234569', 'alena\_front@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Frontend-разработчик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'roman\_back@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Роман', 'Титов', '+375291234571', 'roman\_back@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Backend-разработчик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'vika\_fullstack@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Виктория', 'Шевцова', '+375331234571', 'vika\_fullstack@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Fullstack-разработчик'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'anton\_sysadmin@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Антон', 'Михайлов', '+375441234571', 'anton\_sysadmin@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Системный администратор'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'kristina\_security@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Кристина', 'Егорова', '+375251234570', 'kristina\_security@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Специалист по безопасности'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'maria\_marketing@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Мария', 'Романова', '+375291234572', 'maria\_marketing@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Маркетолог'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'elena\_hr@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Елена', 'Васильева', '+375331234572', 'elena\_hr@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'HR-менеджер'));

IF NOT EXISTS (SELECT 1 FROM Users WHERE Email = 'andrey\_coord@gmail.com')

INSERT INTO Users (First\_Name, Last\_Name, Phone, Email, Password, ID\_Role)

VALUES ('Андрей', 'Белов', '+375441234572', 'andrey\_coord@gmail.com', '$2b$10$exampleHash', (SELECT ID\_Role FROM Roles WHERE Role\_Name = 'Координатор проектов'));

GO

-- Статусы

INSERT INTO Statuses (Status\_Name) VALUES

('Новая'), ('В работе'), ('Завершена'), ('Выполнена');

GO

-- Команды

INSERT INTO Teams (Team\_Name) VALUES

('Alpha'), ('Beta'), ('Gamma');

GO

-- Создание заказов для разных команд

-- Команда Alpha

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Alpha'), 'Веб-приложение для управления проектами', '2025-04-01', '2025-06-30', 'В работе', ID\_User

FROM Users WHERE Email = 'ivan\_manager@gmail.com';

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Alpha'), 'Мобильное приложение для управления задачами', '2025-05-01', '2025-07-15', 'В работе', ID\_User

FROM Users WHERE Email = 'ivan\_manager@gmail.com';

GO

-- Команда Beta

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Beta'), 'Веб-приложение для электронной коммерции', '2025-06-01', '2025-12-01', 'В работе', ID\_User

FROM Users WHERE Email = 'yulia\_admin@gmail.com';

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Beta'), 'Система управления складом', '2025-07-01', '2025-11-30', 'В работе', ID\_User

FROM Users WHERE Email = 'yulia\_admin@gmail.com';

GO

-- Команда Gamma

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Gamma'), 'Система автоматизации бизнеса', '2025-08-01', '2026-01-31', 'В работе', ID\_User

FROM Users WHERE Email = 'pavel\_ba@gmail.com';

INSERT INTO Orders (ID\_ProjectType, ID\_Team, Order\_Name, Creation\_Date, End\_Date, Status, ID\_Manager)

SELECT 1, (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Gamma'), 'Разработка аналитической платформы', '2025-09-01', '2026-03-31', 'В работе', ID\_User

FROM Users WHERE Email = 'pavel\_ba@gmail.com';

GO

-- Задачи для каждого проекта

-- Для проекта "Веб-приложение для управления проектами"

INSERT INTO Tasks (ID\_Order, Task\_Name, Description, Time\_Norm, ID\_Status)

SELECT ID\_Order, 'Проектирование базы данных', 'Проектирование структуры базы данных для веб-приложения', 40, ID\_Status

FROM Orders, Statuses

WHERE Order\_Name = 'Веб-приложение для управления проектами' AND Status\_Name = 'В работе';

INSERT INTO Tasks (ID\_Order, Task\_Name, Description, Time\_Norm, ID\_Status)

SELECT ID\_Order, 'Разработка интерфейса', 'Дизайн интерфейса веб-приложения', 30, ID\_Status

FROM Orders, Statuses

WHERE Order\_Name = 'Веб-приложение для управления проектами' AND Status\_Name = 'В работе';

GO

-- Для проекта "Мобильное приложение для управления задачами"

INSERT INTO Tasks (ID\_Order, Task\_Name, Description, Time\_Norm, ID\_Status)

SELECT ID\_Order, 'Проектирование базы данных для мобильного приложения', 'Проектирование структуры базы данных для мобильного приложения', 40, ID\_Status

FROM Orders, Statuses

WHERE Order\_Name = 'Мобильное приложение для управления задачами' AND Status\_Name = 'В работе';

INSERT INTO Tasks (ID\_Order, Task\_Name, Description, Time\_Norm, ID\_Status)

SELECT ID\_Order, 'Разработка интерфейса для мобильного приложения', 'Дизайн интерфейса мобильного приложения', 35, ID\_Status

FROM Orders, Statuses

WHERE Order\_Name = 'Мобильное приложение для управления задачами' AND Status\_Name = 'В работе';

GO

-- Закрепление задач за сотрудниками

INSERT INTO Assignment (ID\_Task, ID\_Employee, Assignment\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-01'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Проектирование базы данных' AND u.Email = 'olga\_dev@gmail.com';

INSERT INTO Assignment (ID\_Task, ID\_Employee, Assignment\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-01'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Разработка интерфейса' AND u.Email = 'marina\_ui@gmail.com';

GO

-- Назначение сотрудника Петра на две последние задачи (мобильное приложение)

INSERT INTO Assignment (ID\_Task, ID\_Employee, Assignment\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-10'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Проектирование базы данных для мобильного приложения'

AND u.Email = 'petr\_employee@gmail.com';

INSERT INTO Assignment (ID\_Task, ID\_Employee, Assignment\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-10'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Разработка интерфейса для мобильного приложения'

AND u.Email = 'petr\_employee@gmail.com';

-- Выполнение задач

INSERT INTO Execution (ID\_Task, ID\_Employee, Start\_Date, End\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-01 09:00:00', '2025-04-05 17:00:00'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Проектирование базы данных' AND u.Email = 'olga\_dev@gmail.com';

INSERT INTO Execution (ID\_Task, ID\_Employee, Start\_Date, End\_Date)

SELECT t.ID\_Task, u.ID\_User, '2025-04-01 09:00:00', '2025-04-04 17:00:00'

FROM Tasks t, Users u

WHERE t.Task\_Name = 'Разработка интерфейса' AND u.Email = 'marina\_ui@gmail.com';

GO

-- Назначение сотрудников в команды

-- Команда Alpha

INSERT INTO TeamMembers (ID\_Team, ID\_User, Role)

SELECT (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Alpha'), u.ID\_User, r.Role\_Name

FROM Users u JOIN Roles r ON u.ID\_Role = r.ID\_Role

WHERE u.Email IN ('ivan\_manager@gmail.com', 'alex\_teamlead@gmail.com', 'anna\_scrum@gmail.com', 'dmitry\_po@gmail.com', 'olga\_dev@gmail.com', 'vlad\_qa@gmail.com', 'marina\_ui@gmail.com');

-- Команда Beta

INSERT INTO TeamMembers (ID\_Team, ID\_User, Role)

SELECT (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Beta'), u.ID\_User, r.Role\_Name

FROM Users u JOIN Roles r ON u.ID\_Role = r.ID\_Role

WHERE u.Email IN ('yulia\_admin@gmail.com', 'alex\_teamlead@gmail.com', 'sergey\_analyst@gmail.com', 'ekaterina\_devops@gmail.com', 'nikita\_writer@gmail.com', 'alena\_front@gmail.com', 'roman\_back@gmail.com');

-- Команда Gamma

INSERT INTO TeamMembers (ID\_Team, ID\_User, Role)

SELECT (SELECT ID\_Team FROM Teams WHERE Team\_Name = 'Gamma'), u.ID\_User, r.Role\_Name

FROM Users u JOIN Roles r ON u.ID\_Role = r.ID\_Role

WHERE u.Email IN ('pavel\_ba@gmail.com', 'igor\_architect@gmail.com', 'vika\_fullstack@gmail.com', 'anton\_sysadmin@gmail.com', 'kristina\_security@gmail.com', 'maria\_marketing@gmail.com', 'elena\_hr@gmail.com', 'andrey\_coord@gmail.com', 'olga\_dev@gmail.com');

GO

-- Создание представлений

-- Представление для Kanban-доски сотрудников (TasksByStatus)

CREATE OR ALTER VIEW TasksByStatus AS

SELECT

u.ID\_User,

u.First\_Name + ' ' + u.Last\_Name AS Employee\_Name,

t.ID\_Task,

t.Task\_Name,

t.Description,

s.Status\_Name,

o.Order\_Name,

pt.Type\_Name AS Project\_Type,

t.Time\_Norm,

o.End\_Date AS Order\_End\_Date,

CASE

WHEN o.End\_Date IS NOT NULL AND o.End\_Date < GETDATE() AND s.Status\_Name NOT IN ('Завершена', 'Выполнена')

THEN 'Просрочена'

ELSE 'В срок'

END AS Deadline\_Status

FROM Users u

INNER JOIN Assignment a ON u.ID\_User = a.ID\_Employee

INNER JOIN Tasks t ON a.ID\_Task = t.ID\_Task

INNER JOIN Statuses s ON t.ID\_Status = s.ID\_Status

INNER JOIN Orders o ON t.ID\_Order = o.ID\_Order

INNER JOIN ProjectTypes pt ON o.ID\_ProjectType = pt.ID\_ProjectType;

GO

-- Представление для управления заказами и задачами менеджеров (OrdersWithProjectTypes)

CREATE OR ALTER VIEW OrdersWithProjectTypes AS

SELECT

o.ID\_Order,

o.Order\_Name,

pt.Type\_Name AS Project\_Type,

t.Team\_Name,

o.Creation\_Date,

o.End\_Date,

o.Status AS Order\_Status,

u.First\_Name + ' ' + u.Last\_Name AS Manager\_Name,

COUNT(tsk.ID\_Task) AS Total\_Tasks,

SUM(CASE WHEN s.Status\_Name = 'Выполнена' THEN 1 ELSE 0 END) AS Completed\_Tasks,

SUM(CASE WHEN s.Status\_Name IN ('Новая', 'В работе') THEN 1 ELSE 0 END) AS Active\_Tasks,

COALESCE(SUM(tsk.Time\_Norm), 0) AS Total\_Time\_Norm

FROM Orders o

INNER JOIN ProjectTypes pt ON o.ID\_ProjectType = pt.ID\_ProjectType

INNER JOIN Teams t ON o.ID\_Team = t.ID\_Team

INNER JOIN Users u ON o.ID\_Manager = u.ID\_User

LEFT JOIN Tasks tsk ON o.ID\_Order = tsk.ID\_Order

LEFT JOIN Statuses s ON tsk.ID\_Status = s.ID\_Status

GROUP BY

o.ID\_Order, o.Order\_Name, pt.Type\_Name, t.Team\_Name,

o.Creation\_Date, o.End\_Date, o.Status, u.First\_Name, u.Last\_Name;

GO

-- Представление для статистических отчетов по задачам (EmployeeTaskExecution)

CREATE OR ALTER VIEW EmployeeTaskExecution AS

SELECT

pt.Type\_Name AS Project\_Type,

o.Order\_Name,

t.ID\_Task,

t.Task\_Name,

s.Status\_Name,

u.First\_Name + ' ' + u.Last\_Name AS Employee\_Name,

t.Time\_Norm,

e.Start\_Date,

e.End\_Date,

DATEDIFF(HOUR, e.Start\_Date, ISNULL(e.End\_Date, GETDATE())) AS Hours\_Spent,

YEAR(o.Creation\_Date) AS Year,

MONTH(o.Creation\_Date) AS Month

FROM Tasks t

INNER JOIN Orders o ON t.ID\_Order = o.ID\_Order

INNER JOIN ProjectTypes pt ON o.ID\_ProjectType = pt.ID\_ProjectType

INNER JOIN Statuses s ON t.ID\_Status = s.ID\_Status

LEFT JOIN Assignment a ON t.ID\_Task = a.ID\_Task

LEFT JOIN Users u ON a.ID\_Employee = u.ID\_User

LEFT JOIN Execution e ON t.ID\_Task = e.ID\_Task AND u.ID\_User = e.ID\_Employee;

GO

-- Проверка содержимого таблиц и представлений

SELECT 'Roles' AS Table\_Name, \* FROM Roles;

SELECT 'ProjectTypes' AS Table\_Name, \* FROM ProjectTypes;

SELECT 'Users' AS Table\_Name, \* FROM Users;

SELECT 'Statuses' AS Table\_Name, \* FROM Statuses;

SELECT 'Teams' AS Table\_Name, \* FROM Teams;

SELECT 'Orders' AS Table\_Name, o.\*, t.Team\_Name FROM Orders o JOIN Teams t ON o.ID\_Team = t.ID\_Team;

SELECT 'Tasks' AS Table\_Name, \* FROM Tasks;

SELECT 'Assignment' AS Table\_Name, \* FROM Assignment;

SELECT 'Execution' AS Table\_Name, \* FROM Execution;

SELECT 'TeamMembers' AS Table\_Name, tm.\*, u.First\_Name + ' ' + u.Last\_Name AS User\_Name, t.Team\_Name

FROM TeamMembers tm JOIN Users u ON tm.ID\_User = u.ID\_User JOIN Teams t ON tm.ID\_Team = t.ID\_Team;

SELECT 'TasksByStatus' AS View\_Name, \* FROM TasksByStatus;

SELECT 'OrdersWithProjectTypes' AS View\_Name, \* FROM OrdersWithProjectTypes;

SELECT 'EmployeeTaskExecution' AS View\_Name, \* FROM EmployeeTaskExecution;

GO

PRINT 'База данных ProjectManagementDB успешно создана и заполнена';

GO

Код программы:

PORT=3002

DB\_USER=ProssLibrann

DB\_PASSWORD=123456789

DB\_HOST=localhost

DB\_PORT=1433

DB\_NAME=ProjectManagementDB

JWT\_SECRET=mysecretkey123

import sql from 'mssql';

import dotenv from 'dotenv';

// Загружаем переменные окружения

dotenv.config();

// Проверяем, что все необходимые переменные окружения заданы

const requiredEnvVars = ['DB\_USER', 'DB\_PASSWORD', 'DB\_HOST', 'DB\_PORT', 'DB\_NAME'];

const missingEnvVars = requiredEnvVars.filter((envVar) => !process.env[envVar]);

if (missingEnvVars.length > 0) {

console.error('Missing required environment variables:', missingEnvVars.join(', '));

process.exit(1);

}

const dbConfig: sql.config = {

user: process.env.DB\_USER || 'ProssLibrann',

password: process.env.DB\_PASSWORD || '123456789',

server: process.env.DB\_HOST || 'localhost',

port: parseInt(process.env.DB\_PORT || '1433', 10),

database: process.env.DB\_NAME || 'DB\_PROJECTORY',

options: {

encrypt: false, // Для локального SQL Server обычно не требуется шифрование

trustServerCertificate: true, // Для локального сервера

},

};

// Логируем конфигурацию для отладки

console.log('Database configuration:', {

user: dbConfig.user,

server: dbConfig.server,

port: dbConfig.port,

database: dbConfig.database,

});

const poolPromise = new sql.ConnectionPool(dbConfig)

.connect()

.then(pool => {

console.log('Successfully connected to the database:', dbConfig.database);

return pool;

})

.catch(err => {

console.error('Database connection failed:', err.message);

console.error('Error details:', err);

throw err;

});

export default poolPromise;

// backend/src/index.ts

import express from 'express';

import cors from 'cors';

import authRoutes from './routes/authRoutes';

import reportsRoutes from './routes/reports';

import tasksRoutes from './routes/tasks';

import ordersRoutes from './routes/orders';

import commentsRoutes from './routes/comments';

import poolPromise from './config/db';

import dotenv from 'dotenv';

dotenv.config();

const app = express();

const PORT = process.env.PORT || 3002;

app.use(cors({

origin: 'http://localhost:3000',

methods: ['GET', 'POST', 'PUT', 'DELETE'],

allowedHeaders: ['Content-Type', 'Authorization'],

}));

app.use(express.json());

app.use('/api/auth', authRoutes);

app.use('/api/reports', reportsRoutes);

app.use('/api/tasks', tasksRoutes);

app.use('/api/orders', ordersRoutes);

app.use('/api/comments', commentsRoutes);

app.listen(PORT, async () => {

try {

const pool = await poolPromise;

console.log(`Server running on port ${PORT}`);

console.log('Connected to SQL Server');

const result = await pool.request().query('SELECT 1 AS test');

console.log('Database connection test successful:', result);

} catch (err) {

console.error('Database connection failed:', err);

}

});

import axios from "axios";

const API\_URL = process.env.REACT\_APP\_API\_URL;

export const login = async (email: string, password: string) => {

const response = await axios.post(`${API\_URL}/auth/login`, { email, password });

return response.data;

};

export const register = async (email: string, password: string, fullName: string, phone: string) => {

const response = await axios.post(`${API\_URL}/auth/register`, { email, password, fullName, phone });

return response.data;

};

// frontend/src/api/axiosInstance.ts

import axios from 'axios';

const axiosInstance = axios.create({

baseURL: 'http://localhost:3002/api',

headers: {

'Content-Type': 'application/json',

},

});

axiosInstance.interceptors.request.use(

(config) => {

const token = localStorage.getItem('token');

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

},

(error) => {

return Promise.reject(error);

}

);

export default axiosInstance;

import axios from "axios";

import { Task, Comment } from "../types/Task";

const API\_URL = process.env.REACT\_APP\_API\_URL;

export const getTaskById = async (taskId: number): Promise<Task> => {

const token = localStorage.getItem("token");

const response = await axios.get(`${API\_URL}/tasks/${taskId}`, {

headers: { Authorization: `Bearer ${token}` },

});

return response.data;

};

export const updateTask = async (taskId: number, taskData: Partial<Task>): Promise<Task> => {

const token = localStorage.getItem("token");

const response = await axios.put(`${API\_URL}/tasks/${taskId}`, taskData, {

headers: { Authorization: `Bearer ${token}` },

});

return response.data;

};

export const addComment = async (taskId: number, commentText: string): Promise<Comment> => {

const token = localStorage.getItem("token");

const response = await axios.post(

`${API\_URL}/tasks/${taskId}/comments`,

{ text: commentText },

{ headers: { Authorization: `Bearer ${token}` } }

);

return response.data;

};

// backend/src/controllers/authController.ts

import { Request, Response, NextFunction } from 'express';

import bcrypt from 'bcryptjs';

import jwt from 'jsonwebtoken';

import poolPromise from '../config/db';

import { body, validationResult } from 'express-validator';

const JWT\_SECRET = process.env.JWT\_SECRET || 'mysecretkey123';

export const registerValidation = [

body('email').isEmail().withMessage('Неверный формат email'),

body('password').isLength({ min: 6 }).withMessage('Пароль должен быть минимум 6 символов'),

body('phone').isMobilePhone('any').withMessage('Неверный формат телефона'),

body('positionId').isInt().withMessage('PositionId должен быть числом'),

body('fullName').notEmpty().withMessage('Полное имя обязательно'),

];

export const loginValidation = [

body('email').isEmail().withMessage('Неверный формат email'),

body('password').notEmpty().withMessage('Пароль обязателен'),

];

export const register = async (req: Request, res: Response, next: NextFunction): Promise<void> => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

res.status(400).json({ message: 'Ошибка валидации', errors: errors.array() });

return;

}

const { email, password, phone, positionId, fullName } = req.body;

try {

const pool = await poolPromise;

const hashedPassword = await bcrypt.hash(password, 12);

console.log('authController.ts: Сгенерирован хеш пароля:', hashedPassword);

console.log('authController.ts: Попытка регистрации:', { email, fullName, phone, positionId });

const userExists = await pool

.request()

.input('email', email)

.query('SELECT \* FROM Users WHERE Email = @email');

if (userExists.recordset.length > 0) {

console.log('authController.ts: Пользователь с таким email уже существует:', email);

res.status(400).json({ message: 'Пользователь с таким email уже существует' });

return;

}

await pool

.request()

.input('fullName', fullName)

.input('email', email)

.input('password', hashedPassword)

.input('phone', phone)

.input('position', positionId)

.input('role', 'Сотрудник')

.input('isAdmin', 0)

.query(

'INSERT INTO Users (FullName, Email, Password, Phone, PositionID, Role, IsAdmin) VALUES (@fullName, @email, @password, @phone, @position, @role, @isAdmin)'

);

console.log('authController.ts: Пользователь успешно добавлен в таблицу Users:', email);

const newUser = await pool

.request()

.input('email', email)

.query('SELECT UserID, Role FROM Users WHERE Email = @email');

const user = newUser.recordset[0];

if (!user) {

console.log('authController.ts: Не удалось найти нового пользователя:', email);

res.status(500).json({ message: 'Ошибка при создании пользователя' });

return;

}

const token = jwt.sign({ userId: user.UserID, role: user.Role }, JWT\_SECRET, { expiresIn: '1h' });

console.log('authController.ts: Токен сгенерирован для нового пользователя:', { userId: user.UserID, role: user.Role });

res.status(201).json({ token, role: user.Role });

} catch (error) {

console.error('authController.ts: Ошибка регистрации:', error);

res.status(500).json({ message: 'Ошибка сервера при регистрации' });

next(error);

}

};

export const login = async (req: Request, res: Response, next: NextFunction): Promise<void> => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

res.status(400).json({ message: 'Ошибка валидации', errors: errors.array() });

return;

}

const { email, password } = req.body;

try {

const pool = await poolPromise;

console.log('authController.ts: Попытка авторизации:', { email, password });

const result = await pool

.request()

.input('email', email)

.query('SELECT \* FROM Users WHERE Email = @email');

const user = result.recordset[0];

if (!user) {

console.log('authController.ts: Пользователь не найден:', email);

res.status(400).json({ message: 'Неверный email' });

return;

}

console.log('authController.ts: Найденный пользователь:', { userId: user.UserID, email: user.Email, storedHash: user.PlainPassword });

// const isMatch = await bcrypt.compare(password, user.Password);

const isMatch = password === user.PlainPassword

console.log('authController.ts: Результат сравнения пароля:', isMatch);

if (!isMatch) {

console.log('authController.ts: Пароль не совпадает для пользователя:', email);

res.status(400).json({ message: 'Неверный пароль' });

return;

}

const token = jwt.sign({ userId: user.UserID, role: user.Role }, JWT\_SECRET, { expiresIn: '1h' });

console.log('authController.ts: Токен сгенерирован для пользователя:', { userId: user.UserID, role: user.Role });

res.json({ token, role: user.Role });

} catch (error) {

console.error('authController.ts: Ошибка авторизации:', error);

res.status(500).json({ message: 'Ошибка сервера при авторизации' });

next(error);

}

};

**ПРИЛОЖЕНИЕ Б**

**(справочное)**

**Результаты работы программы**

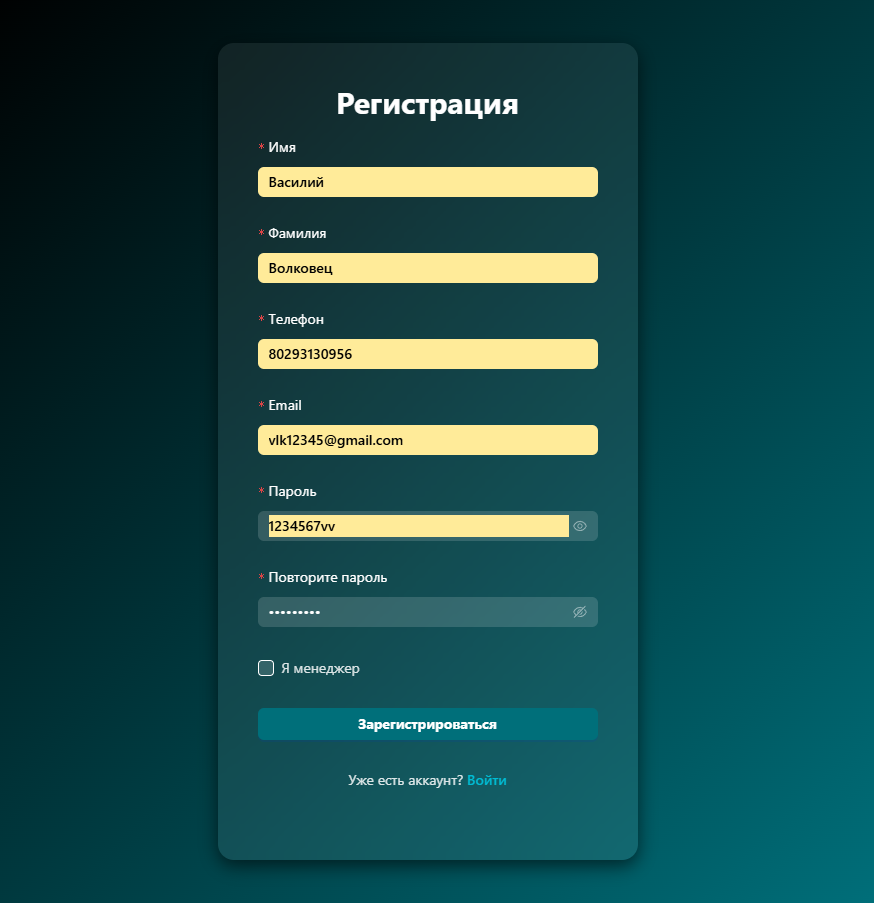
****

Рисунок Б.1 – Страница регистрации

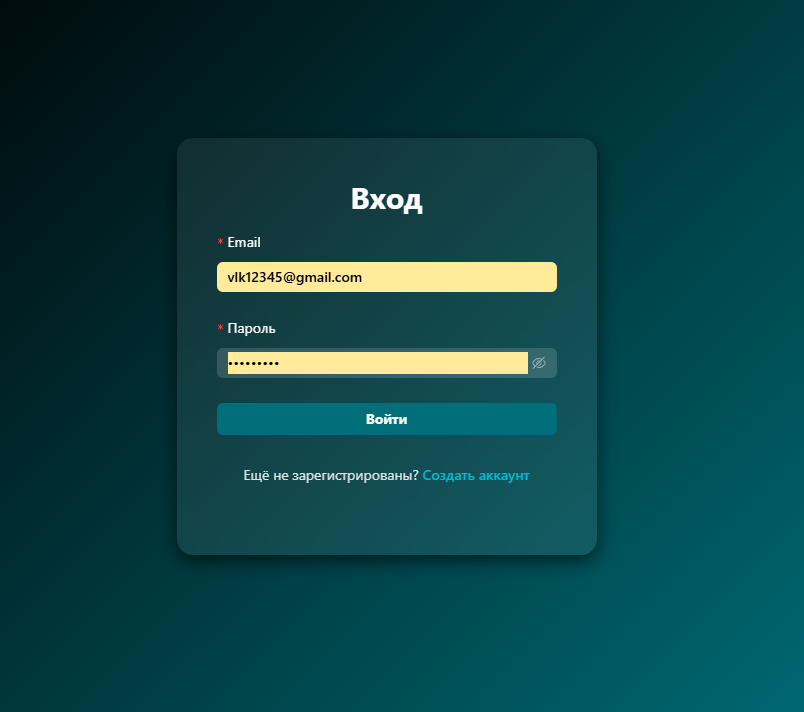


Рисунок Б.2 – Страница авторизации

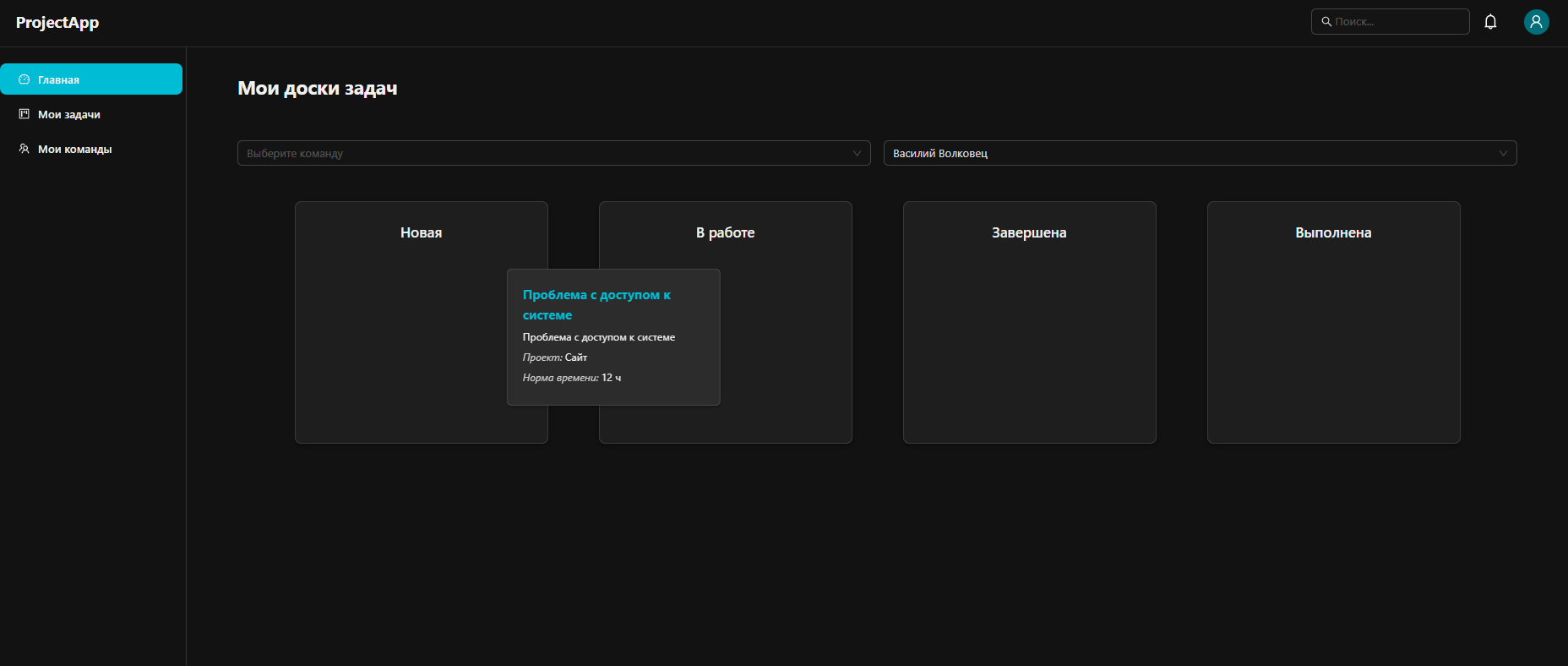


Рисунок Б.3 – Функционал перемещения карточки задачи

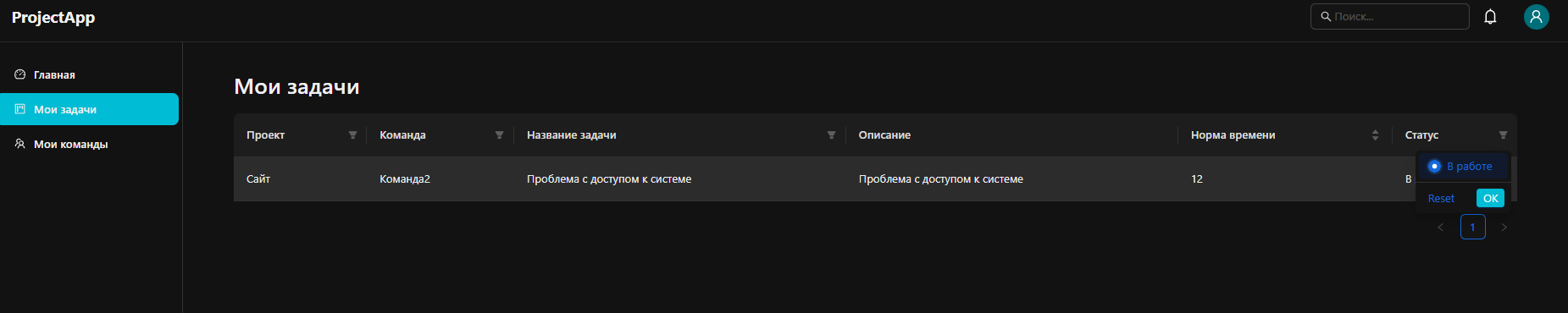


Рисунок Б.4 – Функционал фильтрации статуса задачи

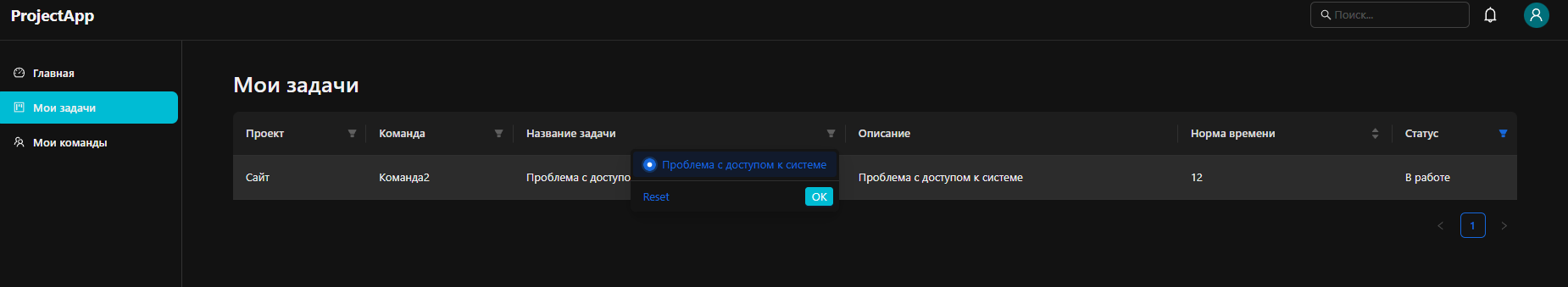


Рисунок Б.5– Результат фильтрации задачи по названию

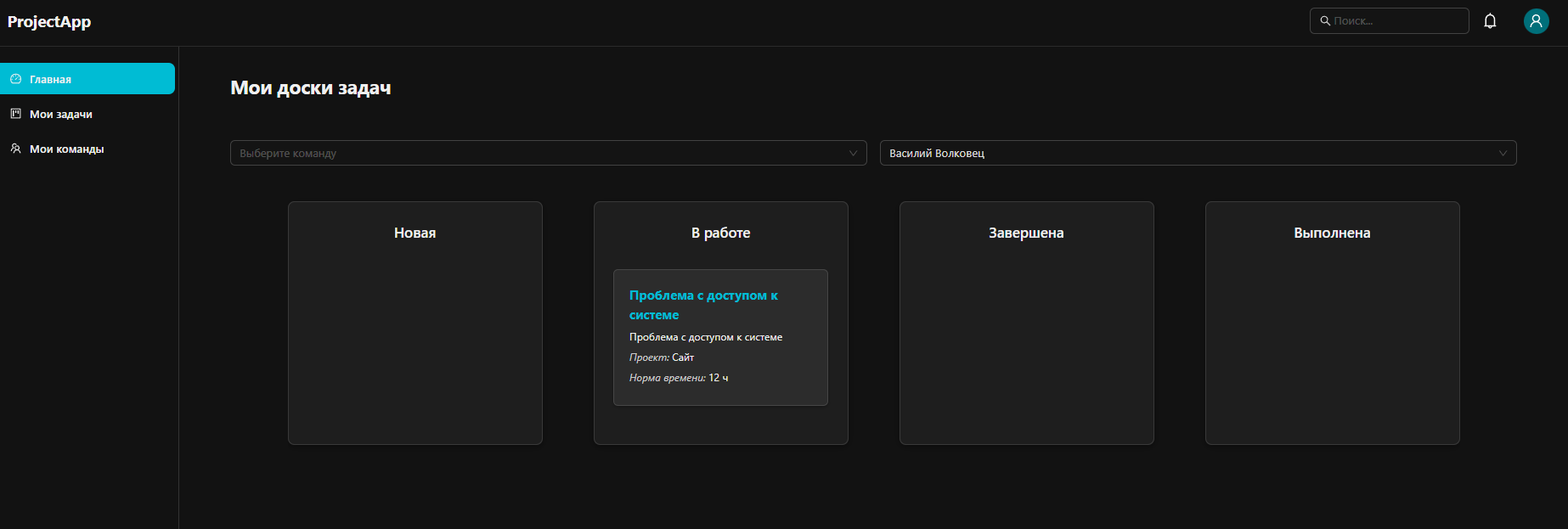


Рисунок Б.6– Страница досок задач

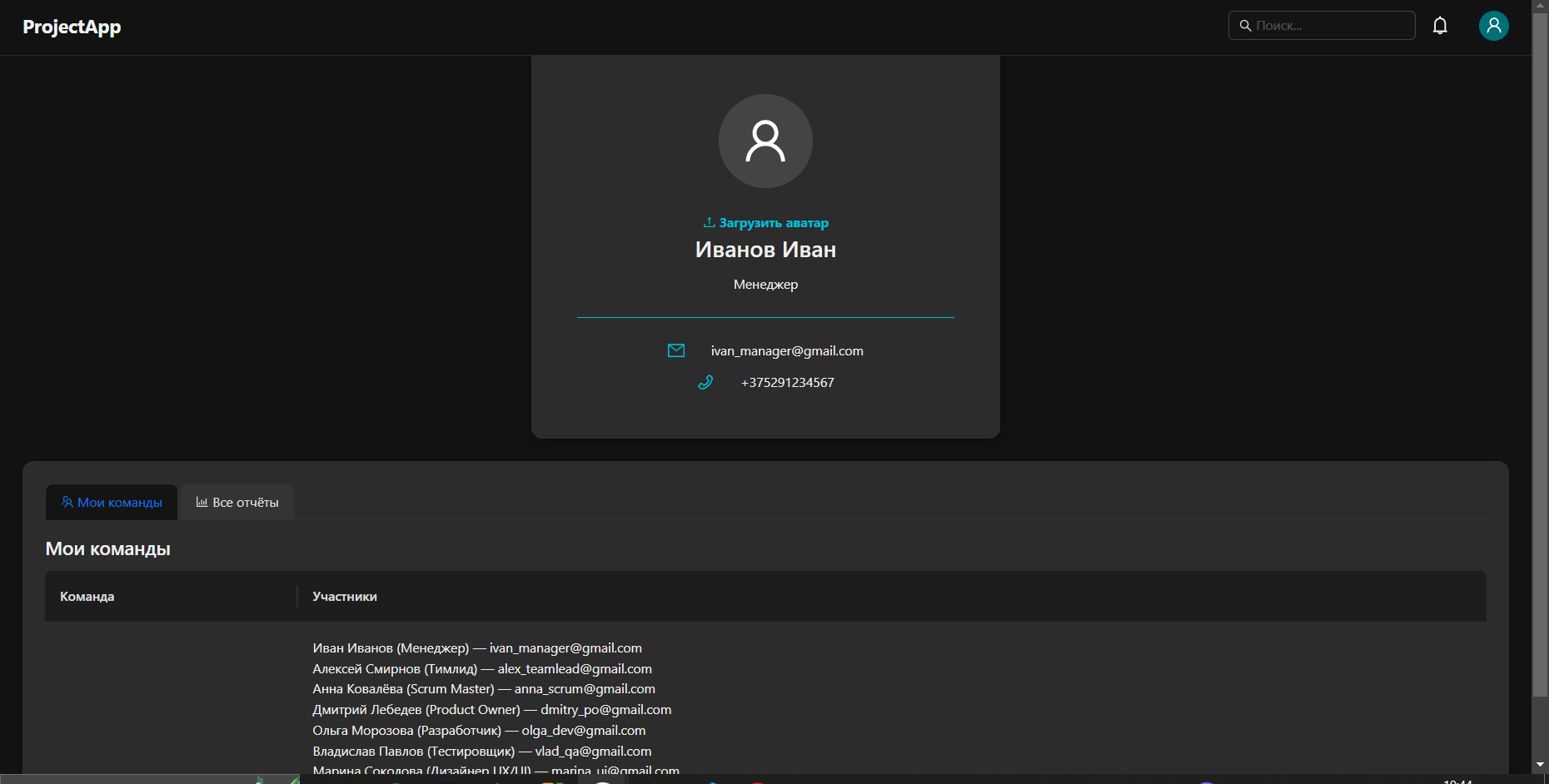


Рисунок Б.7 – Личный кабинет со стороны менеджера

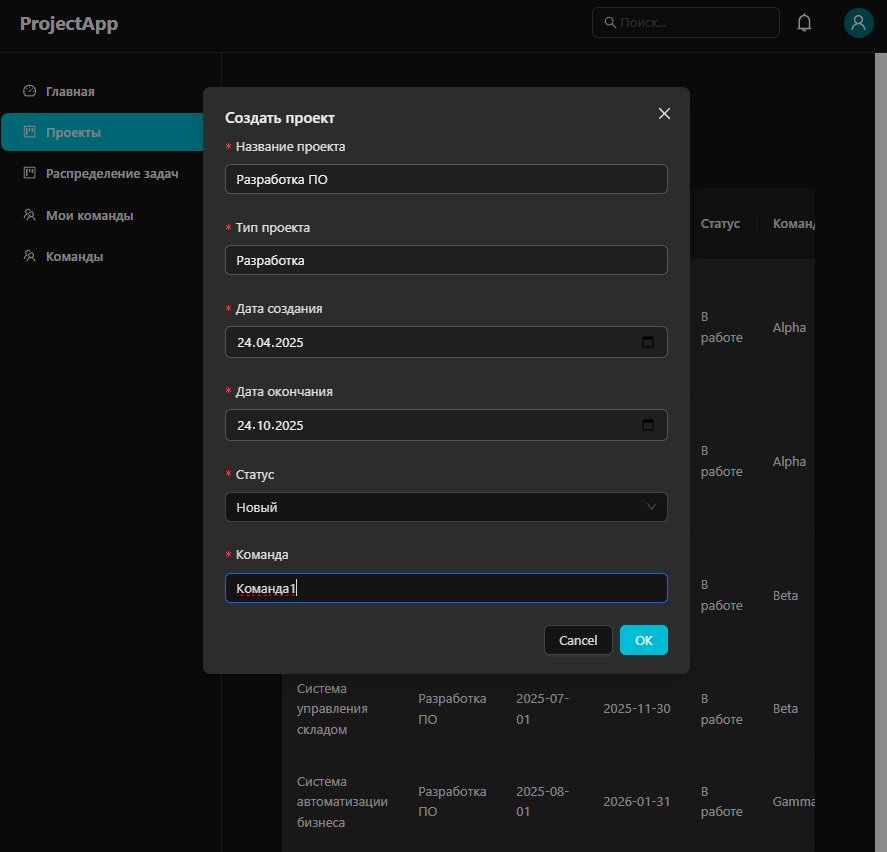


Рисунок Б.8 – Функционал создания проекта

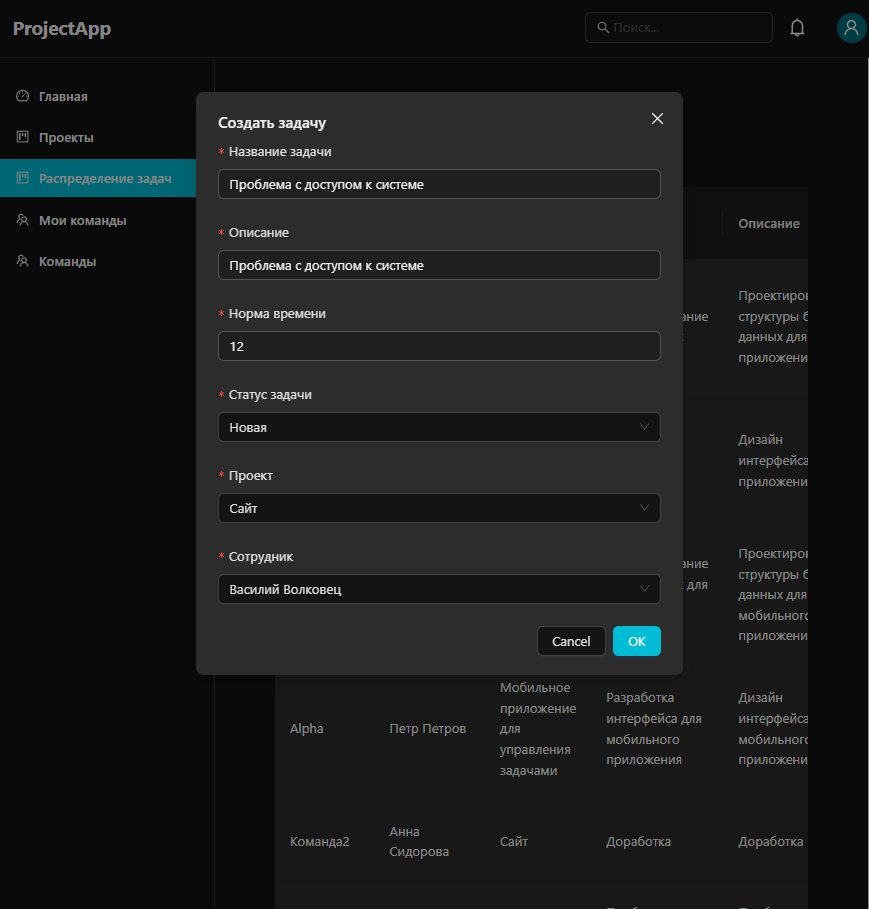


Рисунок Б.9 – Функционал добавления задачи

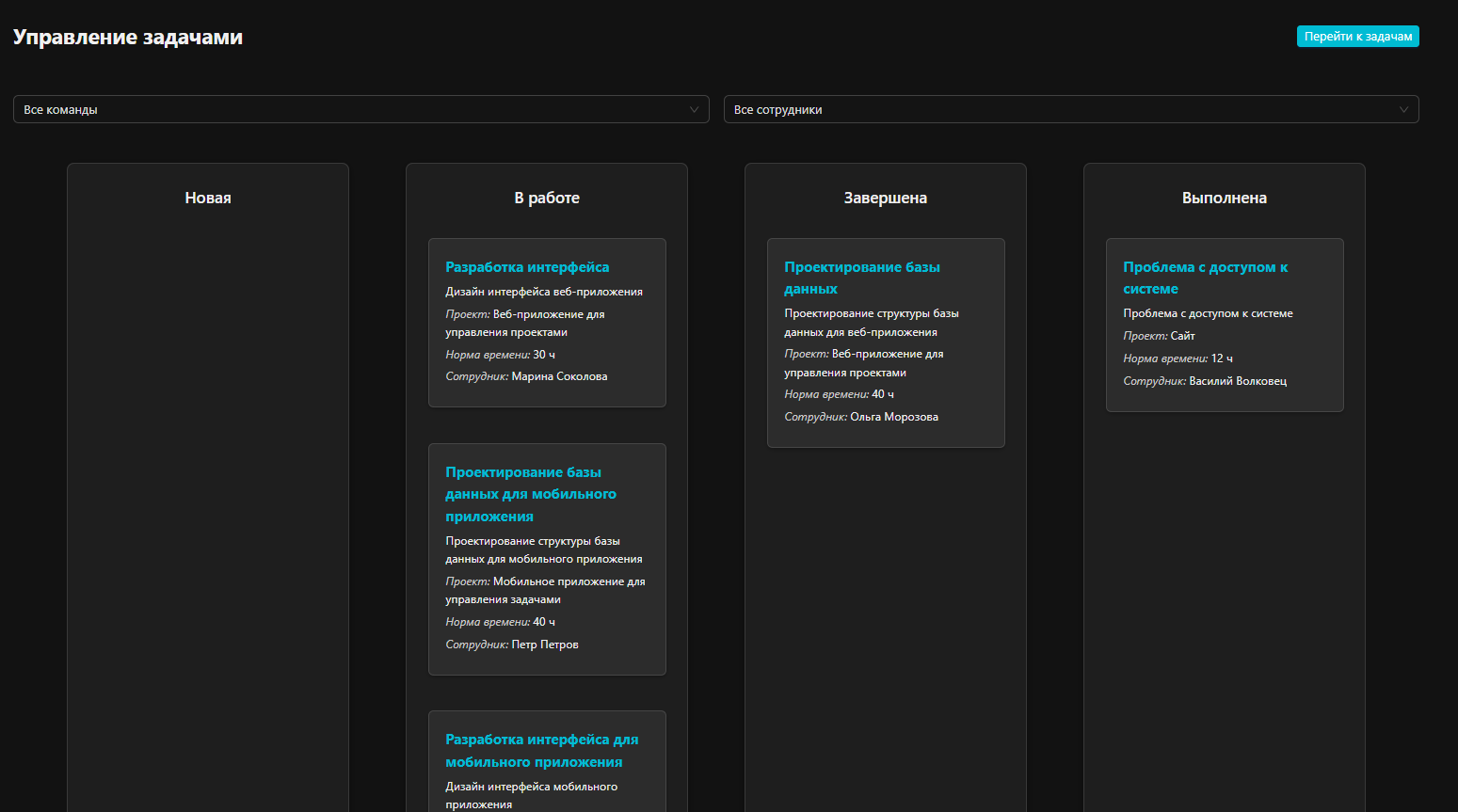
****

Рисунок Б.10 – Изменённый статус задачи «Проблема с доступом в системе»

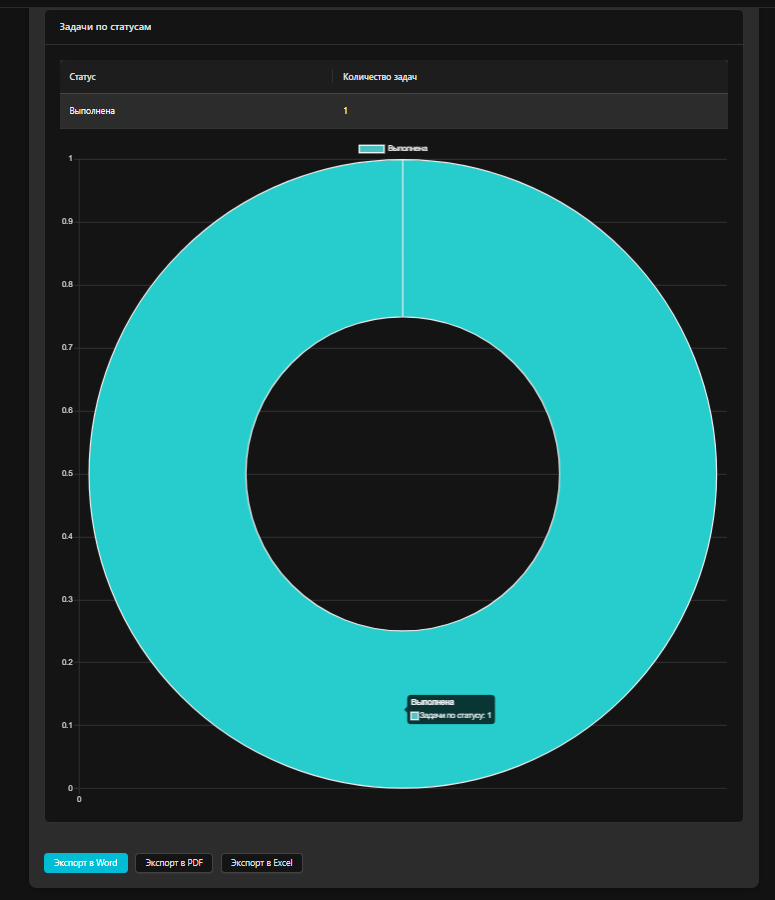


Рисунок Б.11 – Отображение отчётов сотрудника

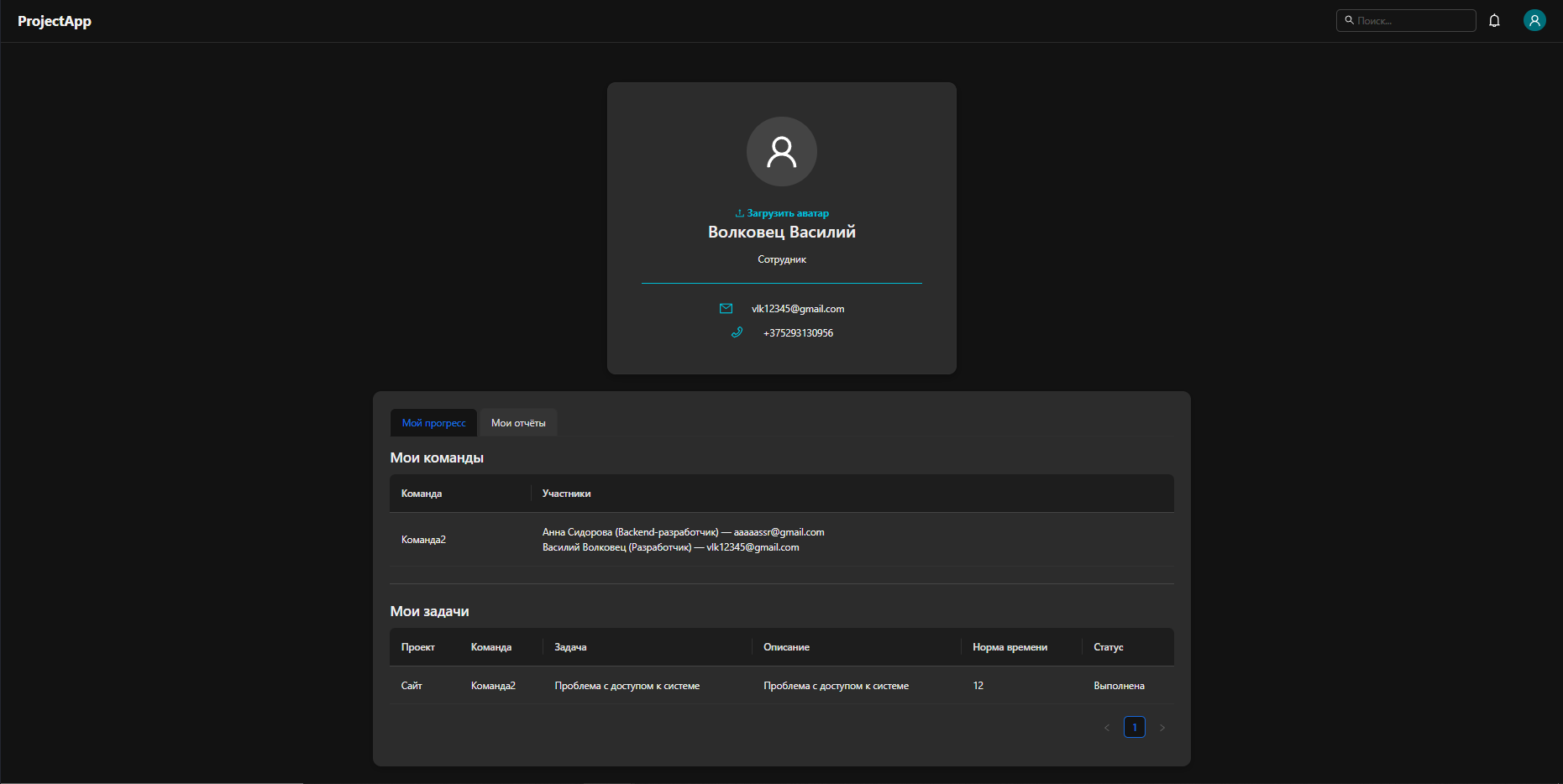
****

Рисунок Б.12 – Страница личного кабинета со стороны сотрудника

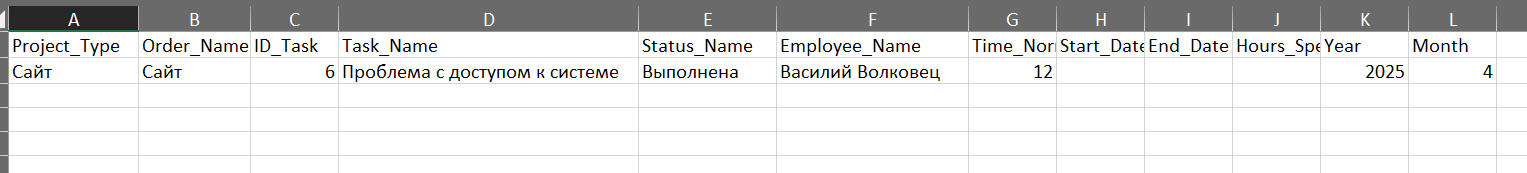


Рисунок Б.13 – Результат экспорта в Excel

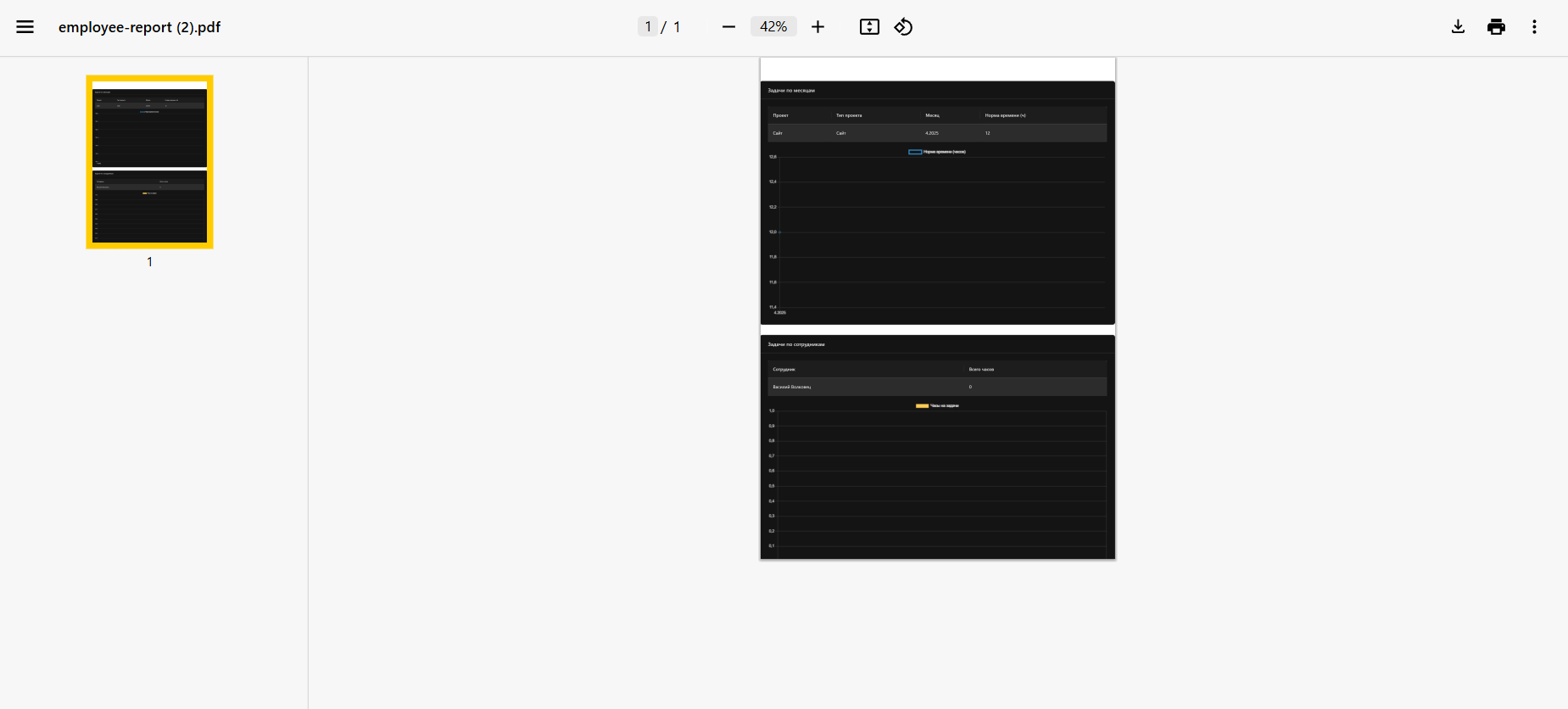


Рисунок Б.14 – Результат экспорта в PDF

**Графическая часть**

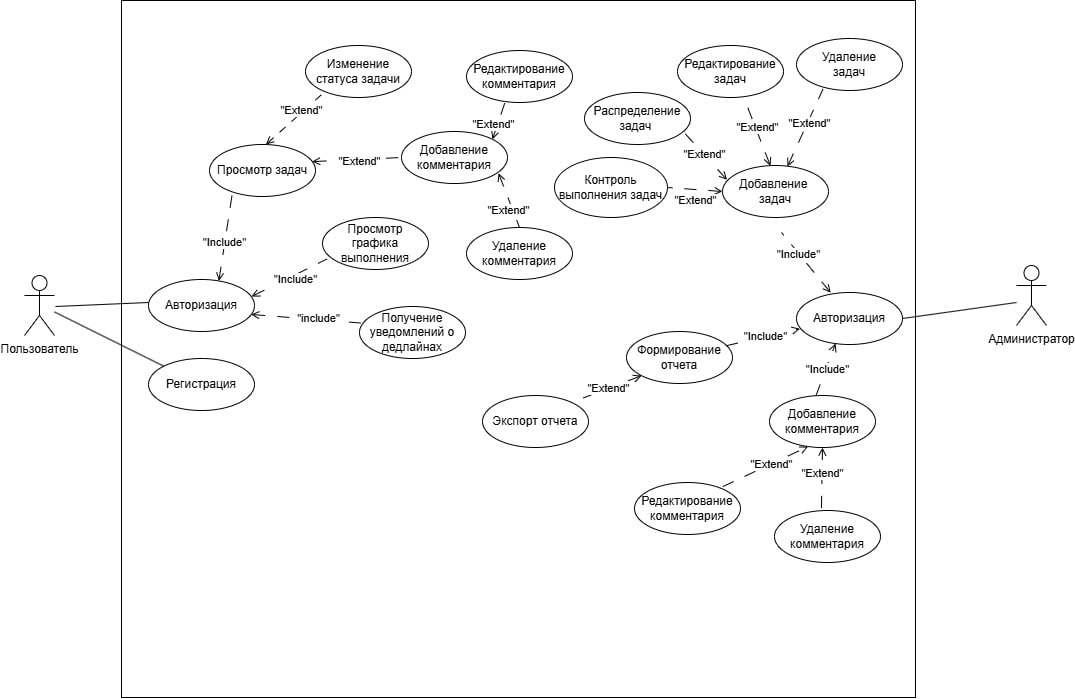


Рисунок 1 – Диаграмма вариантов использования

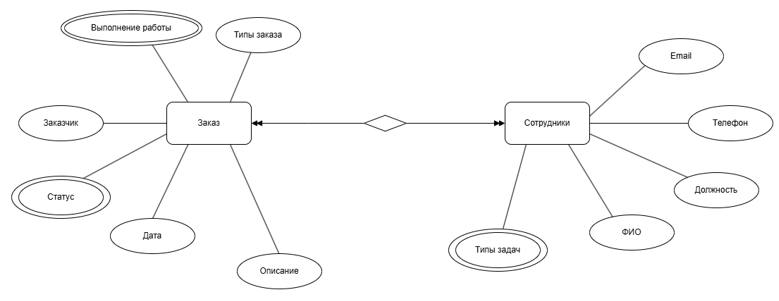


Рисунок 2 – Концептуальная модель диаграммы «Сущность-связь»

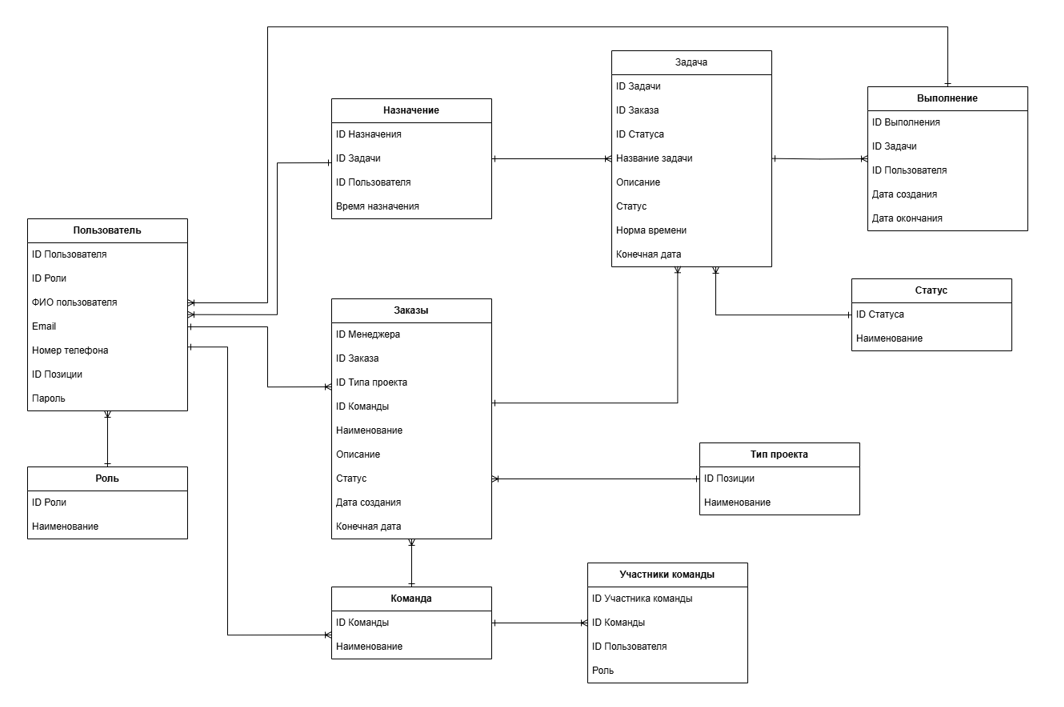


Рисунок 3 – Логическая модель базы данных диаграммы «сущность-связь»

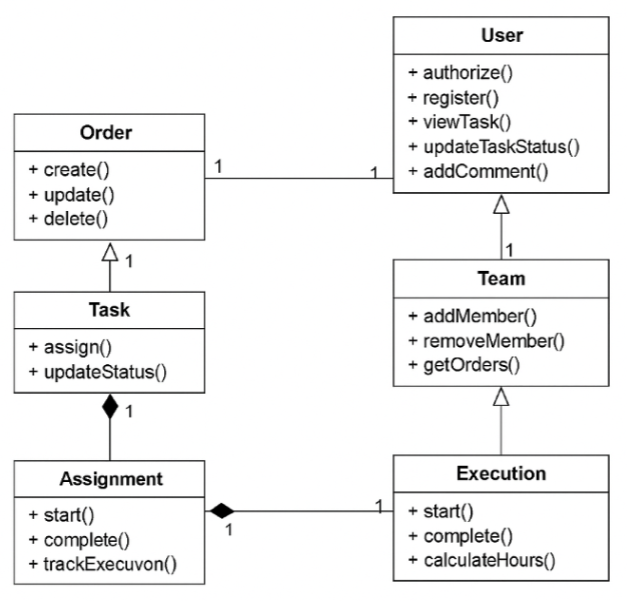


Рисунок 4 – Диаграмма классов

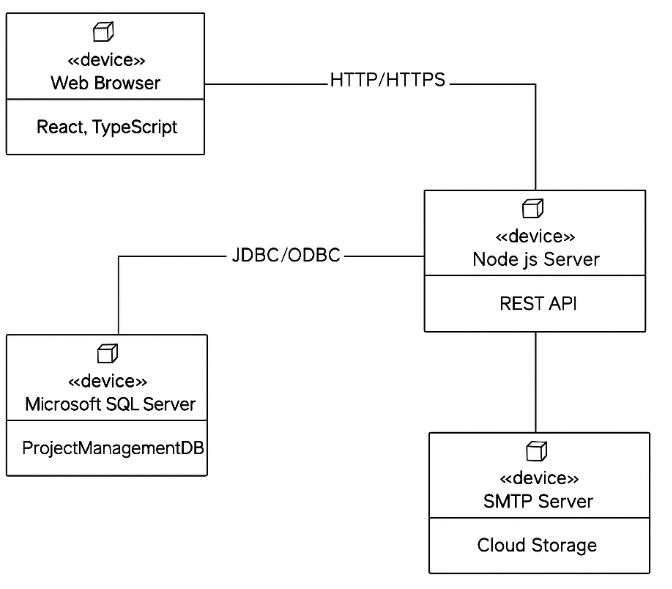


Рисунок 5 – Диаграмма развёртывания