МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

УДК 004.9(04)

К защите

Руководитель направления

Архипов И.О.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2025г.

**Конычев Алексей Кириллович**

«Разработка интерактивной панели управления проектированием объектов капитального строительства для платформы ПИРС»

**Выпускная квалификационная работа магистра**

Направление 09.04.04 «Программная инженерия»

Программа «Разработка программно-информационных систем»

Магистрант

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.К. Конычев

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.Г. Русских

Заведующий кафедрой ПО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.В. Леонов

Ижевск 2025

РЕФЕРАТ

Объем пояснительной записки составляет 108 страниц. Работа содержит 45 иллюстраций и 17 таблиц. Для написания работы было использовано 10 источников информации, ссылки на соответствующие источники указаны в работе.

В работе использованы такие ключевые слова, как: проектирование, объект капитального строительства, заказчик, исполнитель, автоматизация взаимодействия, интерактивная панель, канбан-доска, проектная документация, разработка программного обеспечения, цифровая платформа «ПИРС», React.js, Redux Toolkit, Material UI.

В процессе реализации проектов капитального строительства между заказчиком и исполнителями проектно-изыскательских работ нередко возникают следующие ключевые проблемы:

1) недостаточная эффективность коммуникации — затрудненный обмен информацией и несогласованность действий между участниками проектирования приводят к задержкам в выполнении задач и увеличению количества ошибок на различных этапах;

2) отсутствие прозрачности процессов — заказчик не всегда обладает возможностью отслеживать текущий статус выполнения задач, что затрудняет принятие обоснованных управленческих решений в установленные сроки;

3) несистемный документооборот — затруднения в поиске, актуализации и управлении проектной документацией увеличивают вероятность возникновения технических и организационных сбоев в ходе реализации проекта.

Универсальные системы управления задачами (Jira, Trello, Asana и др.) активно применяются для организации совместной работы, однако в контексте проектирования объектов капитального строительства они демонстрируют ряд ограничений. В частности:

1) отсутствует интеграция с цифровыми платформами проектирования и моделью данных, отражающей структуру проектов, разделов и проектной документации;

2) не поддерживается автоматическое создание задач, привязанных к действиям пользователей внутри строительной платформы (например, добавление или изменение документа);

3) требуют сложной интеграции и не могут функционировать как часть единого пользовательского интерфейса.

Таким образом, универсальные трекеры не удовлетворяют требованиям отрасли, что обусловливает необходимость создания специализированного решения.

В рамках данной работы реализована интерактивная панель управления проектированием, которая:

1) обеспечивает визуализацию иерархической структуры проектов, разделов и документов;

2) поддерживает поиск и фильтрацию по наименованию, статусу и другим параметрам;

3) предоставляет интерфейс управления задачами в формате канбан-доски;

4) автоматизирует направление задач на проверку главному инженеру проекта (ГИП), способствуя упрощению согласовательных процедур и повышению прозрачности взаимодействий.

Внедрение модуля осуществляется в рамках цифровой платформы «ПИРС», которая представляет собой специализированное программное обеспечение, ориентированное на комплексную автоматизацию процессов проектирования в строительстве [1]. Платформа обеспечивает централизованное управление и координацию деятельности всех участников, вовлеченных в разработку проектной документации.

С архитектурной точки зрения платформа «ПИРС» построена по клиент-серверной модели. Серверная часть реализует API, обеспечивающее доступ к данным, а клиентская часть отвечает за отображение информации и взаимодействие с пользователем. Разработанный в рамках данной работы модуль реализован на стороне клиентского приложения с использованием языка программирования TypeScript и современного технологического стека, включающего:

1) React.js — библиотека для построения пользовательских интерфейсов;

2) Node.js — серверная платформа для выполнения JavaScript-кода вне браузера;

3) Material UI — набор настраиваемых и расширяемых компонентов пользовательского интерфейса;

4) HTML — язык разметки для структурирования содержимого веб-страниц;

5) CSS — язык описания внешнего вида и стилей элементов веб-документа;

6) React Hook Form — инструмент для удобной и эффективной работы с веб-формами;

7) Redux Toolkit Query — библиотека для выполнения API-запросов и управления их кэшированием;

8) Yup — библиотека для декларативной валидации данных форм;

9) React Beautiful DND — решение для реализации перетаскивания элементов (drag-and-drop);

10) React Router Dom — библиотека для организации маршрутизации.

Разработанный модуль интегрирован в программное обеспечение «Платформа ПИРС», которое активно используется такими компаниями, как «ПСК «Инжиниринг», «Технология», «БИМИТ», «Академия строительства» и рядом других организаций, работающих на базе цифровой платформы «ПИРС». В настоящее время на платформе зарегистрировано свыше 350 организаций и более 4000 пользователей. Подтверждением внедрения решения является акт, прилагаемый к пояснительной записке.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 10](#_ey7877rmgl3w)

[1. ПРОЕКТИРОВАНИЕ ИНТЕРАКТИВНОЙ ПАНЕЛИ УПРАВЛЕНИЯ ПРОЕКТИРОВАНИЕМ ОБЪЕКТОВ КАПИТАЛЬНОГО СТРОИТЕЛЬСТВА 12](#_kmtsxhqs36t8)

[1.1. Обоснование актуальности разрабатываемой программно-информационной системы 12](#_7fjbrj6arx76)

[1.2. Характеристики прикладной значимости разрабатываемой программно-информационной системы 13](#_k4q0asgr3wg8)

[1.3. Основные методы разработки и архитектура программно-информационной системы 13](#_p7lt2mxrts9u)

[1.4. Аналитический обзор 15](#_z7axnc76qgqw)

[1.5. Классы и характеристики пользователей 16](#_grhgmvtkaphy)

[1.6. Функциональные требования 17](#_4zeqyrwozxmn)

[1.6.1. Просмотр иерархического списка проектов 17](#_ldqhwmbvocyo)

[1.6.2. Просмотр иерархического списка разделов проекта 18](#_vk354xj3nklk)

[1.6.3. Просмотр иерархического списка документов раздела 19](#_d1f941w0oh0h)

[1.6.4. Управление задачами с помощью канбан-доски 19](#_sfqokiymj8vd)

[1.6.5. Создание свободной задачи 20](#_6x1cyzbpv1p7)

[1.6.6. Фильтр для задач канбан-доски 21](#_owe3vmmpz50w)

[1.7. Архитектура системы 23](#_z1r9y9v6ounp)

[1.8. Требования к данным 26](#_s98s4j8fv3rw)

[1.8.1. Логическая модель данных 26](#)

[1.8.2. Словарь данных 27](#_1fc3rjtesdk5)

[1.9. Модель компонентов предметной области 31](#_i01xfkjx7s06)

[1.10. Интерфейсы программного обеспечения 36](#_vojknker7qfg)

[2. РЕАЛИЗАЦИЯ ФУНКЦИЙ ИНТЕРАКТИВНОЙ ПАНЕЛИ УПРАВЛЕНИЯ ПРОЕКТИРОВАНИЕМ ОБЪЕКТОВ КАПИТАЛЬНОГО СТРОИТЕЛЬСТВА 41](#_eqoqs97yw9vu)

[2.1. Реализация функции «Просмотр иерархического списка проектов» 41](#_3es03glmgdp8)

[2.1.1. Реализация алгоритмов 41](#_3065zsyc5nw1)

[2.1.2. Реализация компонентов 43](#_9n1olst964h5)

[2.1.3. Описание контрольных примеров 45](#_3js94nsq8f8y)

[2.2. Реализация функции «Просмотр иерархического списка разделов» 48](#_rc9m5ussrp5j)

[2.2.1. Реализация алгоритмов 48](#_bwj79ytndff9)

[2.2.2. Реализация компонентов 50](#_xuasz93sng4d)

[2.2.3. Описание контрольных примеров 52](#_bzkxchxtgndz)

[2.3. Реализация функции «Просмотр иерархического списка документов» 54](#_dqkwmayeezy3)

[2.3.1. Реализация алгоритмов 54](#_pik8w55ule84)

[2.3.2. Реализация компонентов 59](#)

[2.3.3. Описание контрольных примеров 60](#_7ym98blkpn5)

[2.4. Реализация функции «Управление задачами с помощью канбан-доски» 63](#_lsguwm1ei786)

[2.4.1. Реализация алгоритмов 63](#_nmfs706q27lr)

[2.4.2. Реализация компонентов 66](#_lbdpbko28v8e)

[2.4.3. Описание контрольных примеров 67](#_nt03991ob3m3)

[2.5. Реализация функции «Создание свободной задачи» 72](#_xdquvxnj2ass)

[2.5.1. Реализация алгоритмов 72](#_11n9fkm5fwwh)

[2.5.2. Реализация компонентов 77](#_n7k8ulyfl0ku)

[2.5.3. Описание контрольных примеров 77](#_h6r404fbongu)

[2.6. Реализация функции «Фильтр задач канбан-доски» 80](#_pva730rx3y0o)

[2.6.1. Реализация алгоритмов 80](#_u0np8rjifrxf)

[2.6.2. Реализация компонентов 82](#)

[2.6.3. Описание контрольных примеров 83](#_p6s1b7l46fcl)

[ЗАКЛЮЧЕНИЕ 86](#_xzajnc37vmwa)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 88](#_1110aa55vmzp)

[ПРИЛОЖЕНИЕ 1 89](#_w6e0hmaxith1)

[ПРИЛОЖЕНИЕ 2 104](#_ouqpabl2nycb)

ПЕРЕЧЕНЬ СПЕЦИАЛЬНЫХ ТЕРМИНОВ

Приведем список ключевых специализированных терминов, применяемых в данной работе:

1) React.js — это инструмент для создания веб-интерфейсов, который помогает разрабатывать интерактивные и модульные приложения с четкой структурой компонентов;

2) компонент в React — это самостоятельный элемент интерфейса, объединяющий в себе разметку, логику и стили, и способный многократно использоваться в разных частях приложения;

3) props в React — это параметры, передаваемые в компоненты для настройки их поведения или отображения, что позволяет создавать универсальные и гибкие элементы интерфейса;

5) state в React — это хранилище данных внутри компонента, которое определяет, как он выглядит и работает, и может обновляться в зависимости от действий пользователя или других событий;

6) канбан-доска — это визуальная система для контроля хода выполнения задач, представляющая рабочий процесс в виде карточек, перемещающихся между колонками (этапами);

7) проект — это совокупность задач, документов и этапов, направленных на создание конкретного объекта строительства, например, торгового центра или жилого дома;

8) главный инженер проекта (ГИП) — это специалист, который обеспечивает координацию всех участников проектирования, отвечает за целостность документации и соблюдение сроков;

9) проектная документация — это набор технических документов, описывающих архитектурные, инженерные и конструктивные решения, необходимый для прохождения экспертизы и получения разрешения на строительство;

10) рабочая документация — это детализированные чертежи и инструкции, по которым непосредственно выполняются строительные и монтажные работы на площадке;

11) техническое задание — это исходный документ, в котором сформулированы требования и основные параметры, которым должен соответствовать проектируемый объект.

# ВВЕДЕНИЕ

С развитием информационных технологий и автоматизацией бизнес-процессов всё более актуальной становится задача оптимизации управления проектированием в строительстве.

Проекты капитального строительства характеризуются высокой сложностью, множеством участников и сильной взаимозависимостью между этапами работ. В таких условиях эффективное управление требует специализированных инструментов, интегрированных в цифровую экосистему проектирования.

Обычные системы управления задачами (например, Jira, Trello и др.) не способны полноценно решить эту задачу, поскольку:

1) не поддерживают автоматическое создание задач, привязанных к действиям пользователей внутри строительной платформы;

2) не интегрированы с моделью данных области проектирования объектов капитального строительства;

3) не позволяют гибко визуализировать статус и контекст задач в разрезе архитектуры строительного проекта;

4) требуют сложной интеграции и не могут работать как часть единого интерфейса.

В отличие от этого, интерактивная панель управления проектированием, встроенная в цифровую платформу «ПИРС», предоставляет пользователям единый инструмент для автоматической генерации задач при ключевых действиях (например, создании или изменении документов и разделов), а также мониторинга хода выполнения, контроля сроков и координации всех участников процесса.

Объектом внедрения модуля является цифровая платформа «ПИРС», предназначенная для комплексного управления проектированием объектов капитального строительства.

Внедрение интерактивной панели повысит прозрачность процессов, ускорит принятие решений и обеспечит своевременное выполнение задач в рамках строительных проектов.

Целью работы является автоматизация процессов управления проектированием объектов капитального строительства для обеспечения эффективной координации всех участников проектирования.

Задачи работы:

1) создать интерфейсы интерактивной панели для просмотра иерархических списков проектов, разделов, документов и задач, с возможностью поиска и фильтрации по названиям и статусам;

2) реализовать алгоритмы для получения и отображения информации о проектах, разделах, документах и задачах, с поддержкой динамической фильтрации, создания задач и изменения их статусов с помощью канбан-доски;

3) внедрить разработанный модуль интерактивной панели управления в цифровую платформу «ПИРС».

В настоящее время для управления проектированием объектов применяются различные программные продукты. Однако при их сравнительном анализе становится очевидно, что ни один из существующих веб-сервисов не обладает полным набором возможностей, необходимых для эффективной, качественной и своевременной реализации проектных задач. Каждый из них в той или иной степени демонстрирует отсутствие одной или нескольких критически значимых характеристик.

В связи с этим, для достижения требуемого уровня управления проектированием объектов капитального строительства, необходима интерактивная панель, способная охватывать весь спектр ключевых функций — от обзора проектов, разделов и документов до распределения задач и контроля сроков исполнения.

# 1. ПРОЕКТИРОВАНИЕ ИНТЕРАКТИВНОЙ ПАНЕЛИ УПРАВЛЕНИЯ ПРОЕКТИРОВАНИЕМ ОБЪЕКТОВ КАПИТАЛЬНОГО СТРОИТЕЛЬСТВА

## 1.1. Обоснование актуальности разрабатываемой программно-информационной системы

Изначально строительные компании опирались на традиционные методы планирования и контроля — электронные таблицы, бумажные отчеты, специализированные программы для учёта времени. Однако по мере роста сложности проектов и увеличения числа участников такие инструменты стали требовать чрезмерных усилий для поддержания актуальности данных и теряли эффективность.

С развитием информационных технологий и появлением цифровых платформ в управление строительством начали внедряться более гибкие решения — ERP-системы и системы управления проектами. Они предлагают интеграцию функций по управлению задачами, документами, коммуникациями и финансовыми потоками. Тем не менее, большинство из них не учитывают специфику именно проектирования объектов капитального строительства, где требуется тесная связка с архитектурой данных, автоматическая генерация задач по действиям пользователей и визуализация процессов в строительной логике.

В условиях, когда платформа «ПИРС» уже формирует задачи автоматически при действиях пользователя (например, создании раздела, загрузке документа и пр.), использование типовых трекеров, таких как Jira, оказывается недостаточным: они не интегрированы с моделью данных «ПИРС», не поддерживают бизнес-логику платформы и не обеспечивают необходимый уровень визуализации и контроля.

В этой связи разрабатываемая интерактивная панель управления проектированием предоставляет комплексное решение, полностью встроенное в архитектуру платформы «ПИРС». Она позволяет автоматизировать управление задачами, повысить прозрачность процессов, обеспечить координацию участников и своевременное выполнение задач в рамках сложных строительных проектов.

## 1.2. Характеристики прикладной значимости разрабатываемой программно-информационной системы

Разрабатываемый модуль интерактивной панели управления проектированием объектов капитального строительства, внедренный в цифровую платформу «ПИРС» может быть использован организациями, нацеленными на решение следующих задач:

1) проектирование гражданских объектов: жилые здания и сооружения для городских и сельских инфраструктур;

2) проектирование промышленных объектов: производственные помещения, включая объекты повышенной опасности;

3) агропромышленные объекты: проектирование животноводческих комплексов, теплиц, а также зданий для молочного и мясного производства;

4) объекты электроэнергетики: проектирование и реконструкция энергообъектов для обеспечения предприятий и поселений.

Основное назначение цифровой платформы «ПИРС» заключается в автоматизации процессов управления проектированием, обеспечении координации участников проекта и повышении эффективности выполнения задач на всех этапах жизненного цикла проекта.

Проектировщики, заказчики и ГИП (главные инженеры проекта), использующие платформу «ПИРС», взаимодействуют в рамках проектных задач. Платформа позволяет организовать эффективное сотрудничество между участниками проекта, обеспечивая прозрачность процессов и оперативный обмен информацией.

## 1.3. Основные методы разработки и архитектура программно-информационной системы

В технической реализации цифровая платформа «ПИРС» использует клиент-серверную архитектуру с применением REST API в качестве основного способа взаимодействия между фронтендом и бэкендом. Данный подход обеспечивает модульность, расширяемость и независимость компонентов системы.

Модуль интерактивной панели управления проектированием объектов капитального строительства реализован в клиентском приложении. В разработке модуля использовался язык программирования TypeScript, с применением следующих технологий:

В процессе разработки использовались следующие технологии и инструменты:

1) React.js — библиотека для построения пользовательских интерфейсов;

2) Node.js — серверная платформа для выполнения JavaScript-кода вне браузера;

3) Material UI — набор настраиваемых и расширяемых компонентов пользовательского интерфейса;

4) HTML — язык разметки для структурирования содержимого веб-страниц;

5) CSS — язык описания внешнего вида и стилей элементов веб-документа.

6) React Hook Form — инструмент для удобной и эффективной работы с веб-формами;

7) Redux Toolkit Query — библиотека для выполнения API-запросов и управления их кэшированием;

8) Yup — библиотека для декларативной валидации данных форм;

9) React Beautiful DND — решение для реализации перетаскивания элементов (drag-and-drop);

10) React Router Dom — библиотека для организации маршрутизации и навигации внутри SPA-приложений.

## 1.4. Аналитический обзор

В качестве аналогов разрабатываемого модуля интерактивной панели, внедренного в цифровую платформу «ПИРС», существуют несколько программных продуктов: «Procore» [2], «SmartPM» [3] и «Aconex» [4].

Данные сервисы ориентированы на поиск проектировщиков и в области строительства и архитектуры, позволяют размещать заказы на проектирование и находить подходящих исполнителей. Сравнение ключевых характеристик данных платформ представлено ниже в таблице 1.1.

Таблица 1.1

Сравнение ключевых параметров основных конкурентов

разрабатываемого модуля

| Ключевые характеристики / Основные конкуренты | Procore | SmartPM | Aconex |
| --- | --- | --- | --- |
| Возможность создавать как комплексные проекты,  так и подпроекты | + | + | - |
| Высокая детализация жизненного цикла задачи  (более 30 этапов) | - | - | + |
| Возможность автоматической отправки задачи  на проверку | + | - | - |
| Расширенная система фильтрации задач  (более 10 параметров отбора) | + | - | + |
| Группировка документов в соответствии с их  назначением | - | - | - |
| Снабжение документов визуальной индикацией,  отражающей текущий статус подписания | + | + | - |
| Представление разделов в виде иерархического списка,  структурированного по категориям и подкатегориям | + | + | - |
| Снабжение разделов визуальной индикацией,  отражающей текущий статус выполнения | + | - | + |
| Фильтрация документов, разделов и задачи  по названию и статусу | + | - | - |

## 1.5. Классы и характеристики пользователей

Цифровая платформа включает четыре категории пользователей:

1) заказчик;

2) исполнитель;

3) главный инженер проекта (ГИП);

4) неавторизованный пользователь.

В соответствии с положениями публичной оферты платформы «ПИРС», Заказчиком считается пользователь, зарегистрированный в данной роли либо автоматически получивший этот статус при создании проекта. Заказчиком может выступать физическое лицо, достигшее 18 лет (вне зависимости от гражданства), индивидуальный предприниматель либо юридическое лицо, зарегистрированное в Российской Федерации или в другой юрисдикции. Авторизованный заказчик имеет доступ к интерактивной панели, где он может просматривать свои проекты, их структуру (разделы) и прикреплённые документы.

Исполнитель — это пользователь, зарегистрированный на платформе как участник проектных или инженерных работ и заключивший подрядный договор с заказчиком. В роли исполнителя может выступать физическое лицо (в том числе самозанятый или ИП) либо организация. Исполнитель получает доступ к проектам, в которых он задействован, может управлять задачами через канбан-доску и просматривать соответствующие разделы и документы.

Главный инженер проекта (ГИП) — это отдельная роль, назначаемая на уровне проекта. ГИП может быть назначен как заказчиком, так и выбран в процессе распределения ролей между участниками. Он осуществляет общее техническое руководство проектом, контролирует исполнение задач и координирует взаимодействие между участниками. Авторизованный ГИП одновременно выполняет функции исполнителя в рамках закрепленных за ним проектов.

Неавторизованный пользователь — это посетитель платформы, не прошедший регистрацию и вход в систему. Такой пользователь не имеет доступа к функционалу интерактивной панели, но может пройти процедуру регистрации, выбрав одну из ролей: «Проектировщик» (исполнитель), «Заказчик» или «Главный инженер проекта». Только после авторизации открывается доступ к соответствующим возможностям платформы.

## 1.6. Функциональные требования

### 1.6.1. Просмотр иерархического списка проектов

1.6.1.1. Страница интерактивной панели содержит иерархический список проектов, полученных по соответствующей форме собственности заказчика или проектировщика. Данный список проектов отображается в иерархической структуре, где каждый проект может быть связан с другими проектами или подпроектами.

Каждый проект представлен основной информацией, такой как количество договоров, разделов и задач, что позволяет пользователям быстро оценить объем работ и текущий статус проекта. Для комплексных проектов предусмотрена возможность динамического получения данных о связанных подпроектах. При работе с подпроектами комплексных проектов также применяется динамическая пагинация, что обеспечивает удобство просмотра больших объемов данных [3].

1.6.1.2. Функция просмотра иерархического списка проектов содержит следующие функциональные требования:

1) получение информации о сотрудниках организации по соответствующей форме собственности заказчика/проектировщика;

2) вывод сообщения в случае ошибки получения информации о проектах;

3) отображение иерархического списка проектов;

4) кэширование полученных данных и регулярное получение новой информации о проектах;

5) отображение иерархического списка проектов;

6) динамическое получение данных о дочерних проектах комплексных проектов;

7) визуальное отображение для вложенности проектов;

8) отображение главной информации о проекте: название, количество договоров, разделов и задач;

9) динамическое получение данных о проектах;

10) фильтрация проектов по названию;

11) фильтрация проектов по статусу.

### 1.6.2. Просмотр иерархического списка разделов проекта

1.6.2.1. Страница интерактивной панели содержит иерархический список разделов и подразделов, полученных по соответствующему проекту заказчика/проектировщика. Данный список разделов и отображается в иерархической структуре, где каждый раздел может принадлежать определенной категории или подкатегории.

Каждый раздел представлен основной информацией, такой как его название, шифр, а также количество задач и статус раздела. Предусмотрено наличие поиска и фильтрации разделов по их названию, статусу и шифру.

1.6.2.2. Функция просмотра иерархического списка разделов содержит следующие функциональные требования:

1) получение информации о разделах и подразделах по соответствующей форме собственности;

2) вывод сообщения в случае ошибки получения информации о разделах;

3) отображение иерархического списка разделов;

4) индикация категорий и разделов в соответствие со статусами вложенных разделов;

5) Возможность связывания разделов с другими разделами или подразделами для наглядного представления структуры;

6) отображение главной информации о разделе: количество договоров, разделов и задач, статус, сроки исполнения;

7) фильтрация разделов по названию, шифру и статусу.

### 1.6.3. Просмотр иерархического списка документов раздела

1.6.3.1. Страница интерактивной панели содержит иерархический список документов и актов, отчетов, дополнительных соглашений, полученных по соответствующему разделу проекта заказчика/проектировщика.

Данный список документов и актов, отчетов, дополнительных соглашений отображается в иерархической структуре, где каждый документ может быть связан с другими документами или дочерними элементами, такими как акты, отчеты и дополнительные соглашения.

Каждый документ представлен основной информацией, такой как его название, шифр, количество задач, а также статус документа. Предусмотрено наличие поиска и фильтрации документов, актов, отчетов, дополнительных соглашений по их названию, статусу (подписан/не подписан) и шифру.

1.6.3.2. Функция просмотра иерархического списка документов содержит следующие функциональные требования:

1) получение информации о документах (актах, отчетах и договорах) по соответствующей форме собственности;

2) вывод сообщения в случае ошибки получения информации о документах;

3) отображение иерархического списка документов;

4) индикация документов в зависимости от наличия подписей;

5) отображение главной информации о документе: название, дата подписания (при наличии), ФИО лиц, упомянутых в документе;

7) фильтрация документов по названию и наличию подписей.

### 1.6.4. Управление задачами с помощью канбан-доски

1.6.4.1. Канбан-доска предназначена для управления задачами, направленными на выполнение определенного этапа раздела, назначенная определенному исполнителю. Канбан-доска разделена на 4 колонки: «Все задачи», «В работе», «На проверке» и «Выполнено». Задачи можно перетаскивать по следующим маршрутам:

1) из «Все задачи» в «В работе»;

2) из «В работе» в «Все задачи»;

3) из «В работе» в «Выполнено».

При перетаскивании задачи из колонки «В работе» в колонку «Выполнено» задача попадает либо в колонку «На проверке», в случае, если она требует дополнительной проверки главным инженером проекта, либо в колонку «Выполнено», в случае, если не требует. Перетаскивание задачи в колонку «Выполнено» вызывает соответствующее модальное окно для завершения задачи, в зависимости от ее типа.

1.6.4.2. Функция управления задачами с помощью канбан-доски содержит следующие функциональные требования:

1) получение списка задач исполнителя по соответствующей форме собственности;

2) вывод сообщения в случае ошибки получения информации о задачах;

3) группировка задач по специализации и типу;

4) сортировка задач по дате (по возрастанию и убыванию), по дате и специализации (по возрастанию и убыванию);

5) отображение задач, разделенных по соответствующим колонкам;

6) регулярное фоновое обновление списка задач;

7) перетаскивание задач между разрешенными колонками;

8) определение случаев, когда перетаскивание задач невозможно ввиду отсутствия соответствующих полномочий;

9) отображение модального окна для завершения задачи при ее перетаскивании из колонки «В работе» в колонку «Выполнено». В окне отображается дополнительная информация о задаче, а также список файлов решения с возможностью их скачивания.

### 1.6.5. Создание свободной задачи

1.6.5.1. Свободная задача представляет собой задачу, создаваемую без привязки к конкретному этапу согласования или утверждения, как правило, с минимальными формальными требованиями. Такая задача имеет упрощенную структуру: включает текстовое описание, может быть назначена конкретному исполнителю и автоматически поступает в работу без необходимости предварительной проверки или согласования с главным инженером проекта (ГИП). При создании свободной задачи указываются:

1) проект, к которому относится задача;

2) содержание задачи, описывающее ее цель и требования;

3) специализация исполнителя, определяющая необходимые навыки и знания для выполнения задачи;

4) выбор исполнителя из списка подчиненных главного инженера проекта, а также возможность не указывать исполнителя (в случае, если исполнитель создает задачу для самостоятельного выполнения);

5) дата окончания выполнения задачи;

6) индикатор важной задачи для выделения особо значимых задач.

1.6.5.2. Функция свободной задачи содержит следующие функциональные требования:

1) получение списка проектов по соответствующей форме собственности;

2) получение списка специализаций по текущему исполнителю;

3) получение списка исполнителей по соответствующей форме собственности;

4) вывод сообщения в случае ошибки получения данных;

5) отображение модального окна, содержащего поля для ввода данных;

6) наличие валидации введенных полей;

7) формирование введенных пользователем данных для отправления на сервер;

8) отправление сформированных данных на сервер;

9) вывод сообщения в случае ошибки отправления данных.

### 1.6.6. Фильтр для задач канбан-доски

1.6.6.1. Фильтр представляет собой набор полей, изменение которых приводит к обновлению списка задач с учетом установленных пользователем ограничений. Поля фильтра предоставляют пользователям возможность выбора следующих параметров:

1) проект, по которому требуется загрузить задачи на канбан-доску;

2) диапазон дат для отображения задач;

3) тип задачи;

4) специализация исполнителя (фильтрация задач по необходимым навыкам);

5) группа проектировщиков (для уточнения ответственных за выполнение задач);

Также предусмотрена возможность сброса значений всех полей фильтра для быстрого обновления отображаемых задач.

1.6.6.2. Функция фильтра для задач канбан-доски содержит следующие функциональные требования:

1) получение списка проектов по соответствующей форме собственности;

2) получение списка специализаций по текущему исполнителю;

3) получение списка исполнителей по соответствующей форме собственности;

4) получение списка типов задач;

5) получение списка групп проектировщиков

4) вывод сообщения в случае ошибки получения данных;

5) отображение формы фильтра, содержащей поля для ввода данных;

6) наличие валидации введенных полей;

7) формирование введенных пользователем данных для отправления на сервер;

8) отправление сформированных данных на сервер;

9) вывод сообщения в случае ошибки отправления данных;

10) сброс установленных пользователем данных.

## 1.7. Архитектура системы

1.7.1. Модель сценариев использования предметной области

Диаграмма вариантов использования предметной области представлена на рисунке 1.1.

В системе предусмотрено четыре типа пользователей: неавторизованный пользователь, исполнитель, заказчик и главный инженер проекта (ГИП). Неавторизованные пользователи не имеют доступа к интерактивной панели и канбан-доске. Для работы с модулем необходима авторизация на платформе «ПИРС».

Авторизованные пользователи могут просматривать интерактивную панель с иерархическим списком проектов и подпроектов, сгруппированных по форме собственности. Каждый проект включает разделы и документы (договоры, акты, отчеты), представленные в иерархической структуре с возможностью поиска и фильтрации по ключевым параметрам (название, шифр, статус и др.).

Исполнители получают доступ к канбан-доске, где задачи можно перетаскивать между колонками в зависимости от стадии выполнения.

Главный инженер проекта (ГИП) проверяет задачи в колонке «На проверке». Если задача выполнена корректно, он может переместить её в колонку «Выполнено». При этом открывается страница проверки, где отображаются детали задачи перед окончательным завершением.

Диаграмма вариантов использования

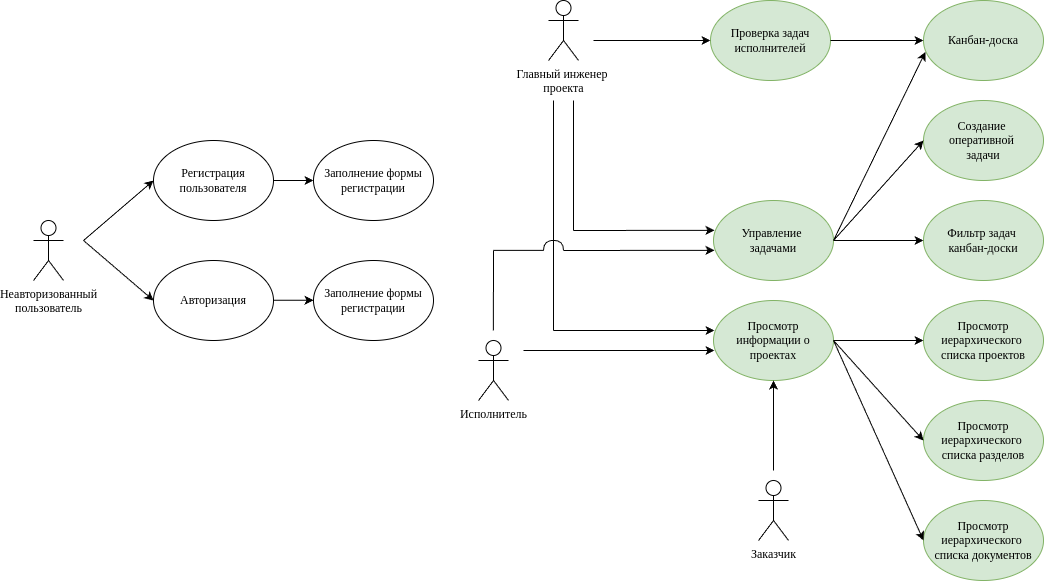


Рис. 1.1

1.7.2. Структурное моделирование модуля

В структурном моделировании, модуль разделен на два компонента: просмотр информации об объектах капитального строительства и управление задачами на разработку проектной и рабочей документации. Структурная схема модуля представлена ниже на рис. 1.2.

Пользователь системы (в роли заказчика или исполнителя) переходит на страницу дашборда для просмотра перечня проектов. После выбора конкретного проекта или подпроекта осуществляется загрузка иерархического списка разделов, сгруппированных по соответствующим категориям и подкатегориям. Разделы представлены в виде структурированного дерева, отражающего вложенность и взаимосвязь элементов.

Каждый раздел содержит прикрепленные документы (договоры, акты, отчеты), доступные для просмотра при выборе соответствующего раздела. Документы дополнительно структурированы по типовым папкам: «Договоры», «Акты», «Отчёты».

Для удобства навигации реализован механизм фильтрации отображаемой информации по наименованию и статусу проекта, раздела и документа.

Пользователь системы в роли исполнителя переходит на канбан-доску, где отображается перечень задач, распределенных по четырем колонкам:

1) все задачи;

2) в работе;

3) на проверке;

4) выполнено.

Задачи могут быть перемещены из колонки «Все задачи» в «В работе» и обратно. После завершения задачи она может быть перенесена в колонку «На проверке». Если задача требует обязательной проверки, она остаётся в данной колонке до прохождения проверки. В противном случае задача автоматически перемещается в колонку «Выполнено».

По результатам проверки главный инженер проекта (ГИП) может либо вернуть задачу в работу, либо подтвердить ее выполнение, переместив её в колонку «Выполнено».

Кроме того, на странице канбан-доски доступна возможность создания задачи класса «Свободная задача». Такая задача автоматически попадает в колонку «В работе» и не требует проверки.

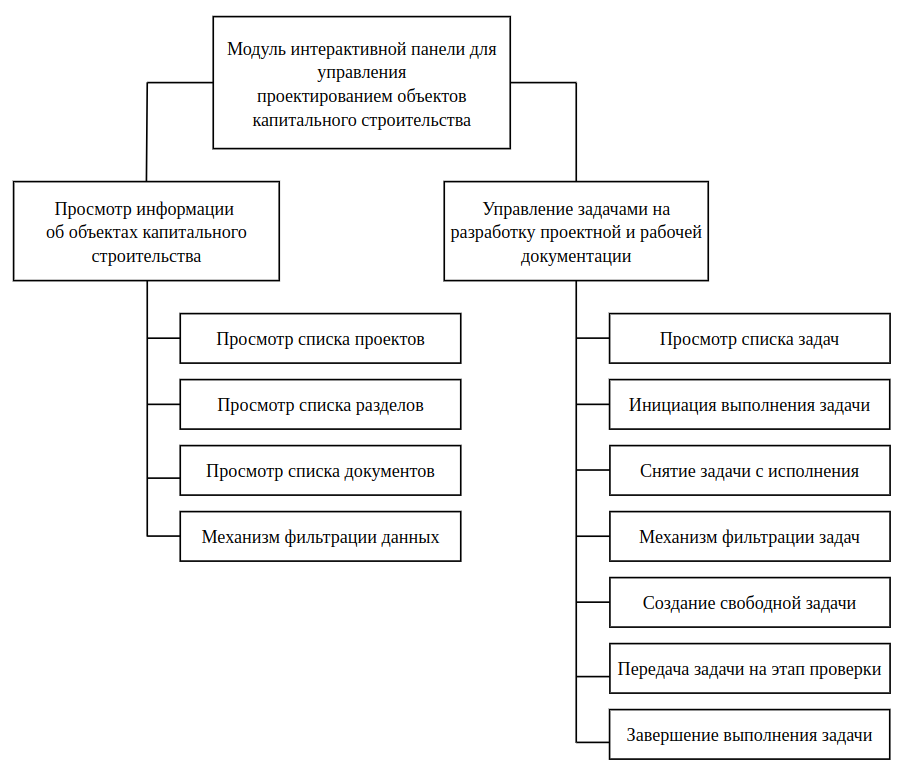
Структурная схема модуля

Рис. 1.2

## 1.8. Требования к данным

### 1.8.1. Логическая модель данных

Разрабатываемый модуль интегрирован в клиентское приложение, взаимодействующее с API серверной части системы. Для эффективного управления данными, связанными с функционированием интерактивной панели, команда бэкенд-разработчиков цифровой платформы «ПИРС» разработала специализированную логическую модель базы данных (см. рис. 1.3). Указанная модель представляет собой фрагмент общей логической структуры реляционной базы данных, лежащей в основе платформы.

Логическая модель данных модуля интерактивной панели управления проектированием объектов капитального строительства

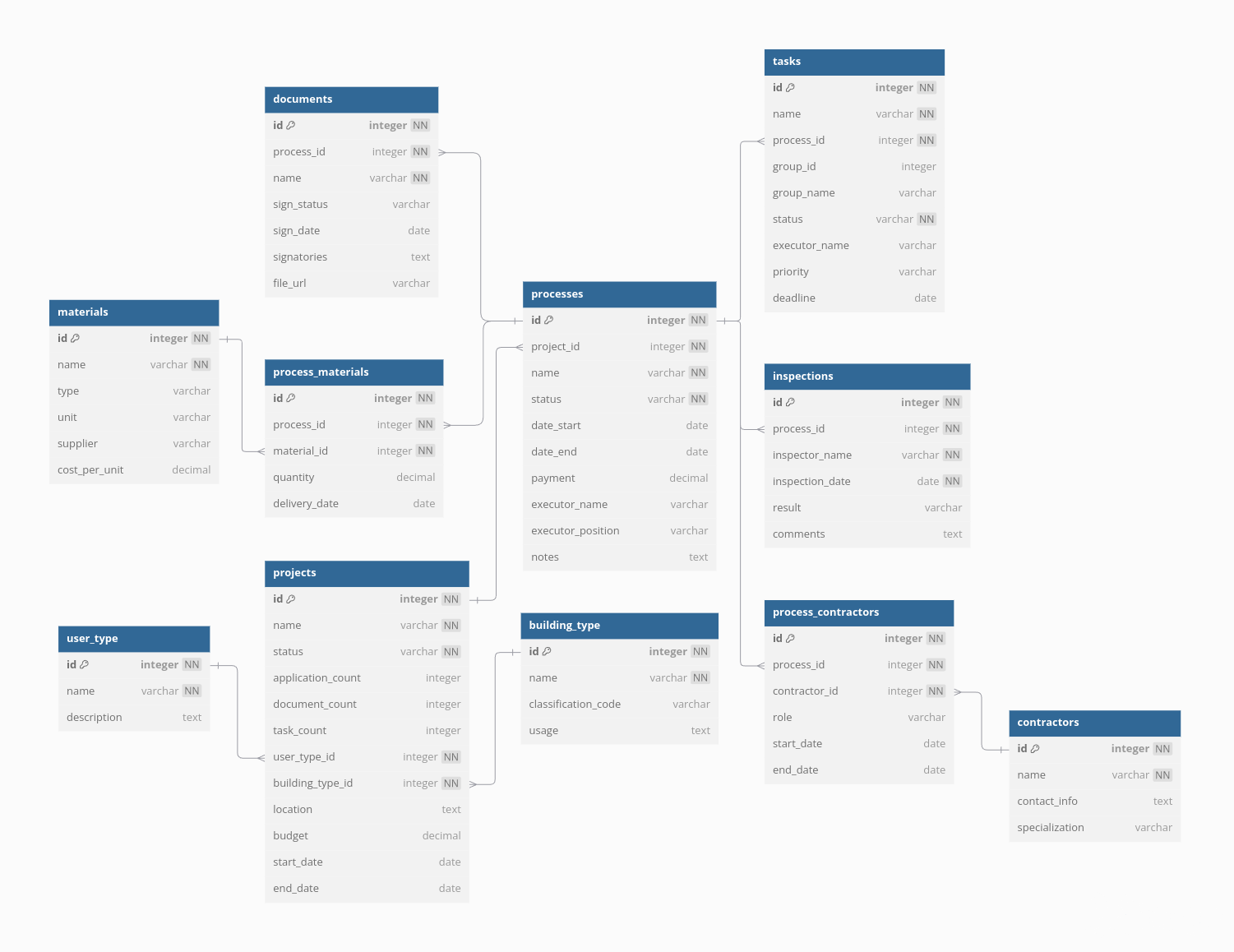


Рис. 1.3

### 1.8.2. Словарь данных

Ниже приведено подробное описание модели данных модуля интерактивной панели для управления проектированием объектов капитального строительства.

В таблице 1.4 приведена структура модели «User\_type», которая содержит информацию о формах собственности пользователя.

Таблица 1.2

Структура модели «User\_type»

| Поле | Формат | Ключ | Ссылка  внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Description | Text |  |  |

В таблице 1.3 приведена структура модели «Building\_type», которая содержит информацию о специализации задач.

Таблица 1.3

Структура модели «Building\_type»

| Поле | Формат | Ключ | Ссылка  внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Classification\_code | Varchar(20) |  |  |
| Usage | Text |  |  |

В таблице 1.4 приведена структура модели «Process\_executors», которая содержит информацию о доступных на раздел исполнителях.

Таблица 1.4

Структура модели «Process\_executors»

| Поле | Формат | Ключ | Ссылка внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Process\_id | Integer | FK | Process.id |

В таблице 1.5 приведена структура модели «Projects», которая содержит информацию о проектах.

Таблица 1.5

Структура модели «Projects»

| Поле | Формат | Ключ | Ссылка  внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Status | Integer |  |  |
| Application\_count | Integer |  |  |
| Document\_count | Integer |  |  |
| Task\_count | Integer |  |  |
| User\_type\_id | Integer |  |  |
| Building\_type\_id | Integer |  |  |
| Location | Varchar(60) |  |  |
| Start\_date | Date |  |  |
| End\_date | Date |  |  |

В таблице 1.6 приведена структура модели «Documents», которая содержит информацию о документах.

Таблица 1.6

Структура модели «Documents»

| Поле | Формат | Ключ | Ссылка внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Process\_id | Integer | FK | Process.id |
| Sign\_status | Integer |  |  |
| Sign\_date | Date |  |  |
| Signatories | Varchar(120) | FK | Executor.id |

В таблице 1.7 приведена структура модели «Processes», которая содержит информацию о разделах.

Таблица 1.7

Структура модели «Processes»

| Поле | Формат | Ключ | Ссылка внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Project\_id | Integer | FK | Project.id |
| Status | Integer |  |  |
| Task\_count | Integer |  |  |
| Executor\_id | Integer | FK | Executor.id |
| Executor\_name | Varchar(60) |  |  |
| Payment | Decimal |  |  |
| Start\_date | Date |  |  |
| End\_date | Date |  |  |

В таблице 1.8 приведена структура модели «Executors», которая содержит информацию об исполнителях.

Таблица 1.8

Структура модели «Executors»

| Поле | Формат | Ключ | Ссылка  внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| First\_name | Varchar(40) |  |  |
| Last\_name | Varchar(40) |  |  |
| Date\_of\_birth | Date |  |  |
| Email | Varchar(40) |  |  |
| Passport | Varchar(10) |  |  |

В таблице 1.9 приведена структура модели «Tasks», которая содержит информацию о задачах.

Таблица 1.9

Структура модели «Tasks»

| Поле | Формат | Ключ | Ссылка  внешнего ключа |
| --- | --- | --- | --- |
| Id | Integer | PK |  |
| Name | Varchar(20) |  |  |
| Process\_id | Integer | FK | Process.id |
| Group\_id | Integer | FK | Group.id |
| Group\_name | Varchar(60) |  |  |
| Status | Integer |  |  |
| Executor\_id | Integer | FK | Executor.id |
| Priority | Boolean |  |  |
| Deadline | Date |  |  |

## 1.9. Модель компонентов предметной области

Модуль реализован с использованием библиотеки React, где пользовательский интерфейс формируется на основе компонентов — функций, отвечающих за его логику и отображение. На рисунке 1.4 представлена компонентная модель, отражающая структуру предметной области [5].

Модель компонентов предметной области

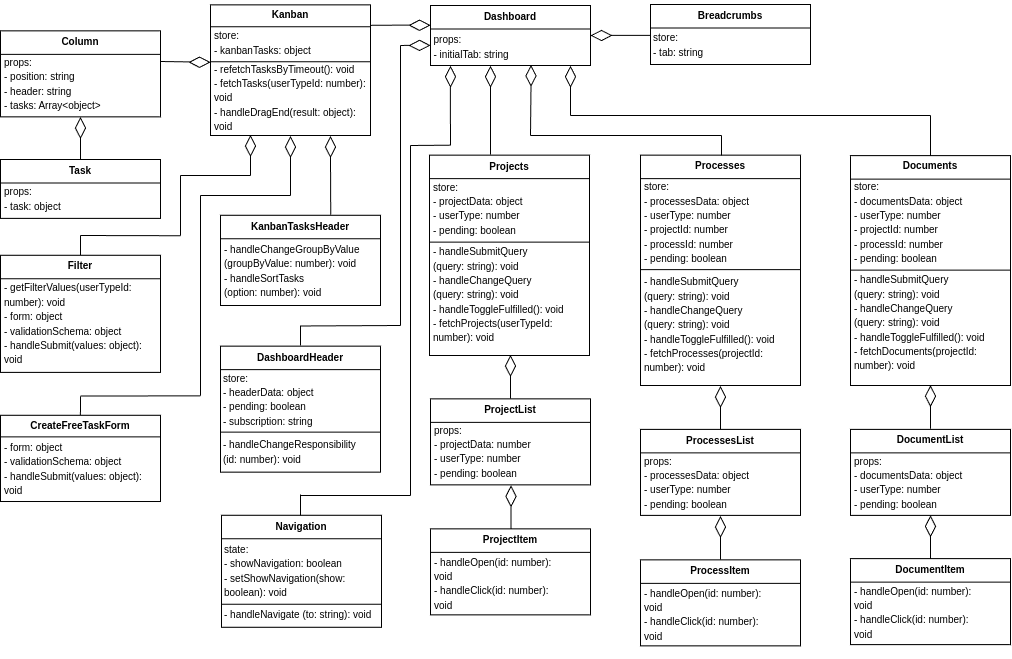


Рис. 1.4

Последовательно рассмотрим каждый из представленных компонентов:

1) Dashboard. Компонент Dashboard представляет собой центральную точку управления пользовательским интерфейсом панели и отвечает за отображение контента в зависимости от текущей вкладки. При инициализации компонент устанавливает начальную вкладку и следит за ее изменением, перенаправляя пользователя на соответствующий маршрут.

2) Breadcrumbs. Компонент отображает навигационную цепочку, отражающую текущую позицию пользователя в иерархии проектов и разделов. Он формирует последовательность элементов — проектов и разделов — на основе выбранных пользователем значений. Если элемент в цепочке активен как ссылка, по клику происходит переключение вкладки. Компонент помогает пользователю ориентироваться в структуре данных и быстро возвращаться к родительским элементам.

3) Projects. Компонент управляет отображением и фильтрацией списка проектов на дашборде. Он поддерживает поиск по названию проекта и переключение между выполненными и активными проектами и адаптирует интерфейс под мобильные устройства. Значения фильтров используются для отправки запроса на сервер, чтобы получать только соответствующие проекты. Полученные данные подгружаются постранично с помощью динамической пагинации.

4) ProjectList. Компонент отвечает за отображение списка проектов с учетом параметров фильтрации и поиска. Реализует динамическую пагинацию: данные загружаются порционно, а не одним большим списком, что особенно важно для производительности. Компонент также отслеживает состояние загрузки и общее количество проектов.

5) ProjectItem. Компонент отображает отдельный проект в списке и управляет его взаимодействием с пользователем. Он показывает название проекта, индикаторы статуса, счётчики заявок, неподписанных документов и задач. Также отображаются кнопки для открытия вложенных проектов и перехода на страницу проекта. Компонент поддерживает отображение иерархии за счет визуальных отступов, указывающих на вложенность.

При клике на проект загружаются данные проекта и его разделов, Компонент также позволяет открывать и закрывать подчинённые проекты. Если подчиненные проекты еще не загружены, инициируется запрос на сервер.

6) Processes. Компонент отвечает за отображение и фильтрацию списка разделов проекта. Инициирует загрузку разделов, основываясь на выбранном пользователем проекте. Отображает иерархический список разделов. Пользователь также может переключать отображение выполненных разделов и выполнять фильтрацию по названию раздела. Фильтрация данных осуществляется на стороне клиента, что обеспечивает реактивное обновление интерфейса при изменении фильтров или строки поиска.

7) ProcessList. Компонент отображает элементы иерархического списка, которые могут быть как папками, так и разделами. Он управляет раскрытием и сворачиванием папок, фильтрацией и отображением дочерних элементов. Папки могут содержать дочерние элементы, которые могут быть другими папками или разделами. Когда папка раскрыта, её дочерние элементы отображаются рекурсивно с увеличением отступа на каждом уровне вложенности. Отображаются цветовые метки рядом с разделами, которые зависят от статуса соответствующего раздела.

8) ProcessItem. Компонент отображает раздел, включая информацию о дате начала и окончания, цене, статусе и других характеристиках раздела. Внутри компонента реализованы различные функции, такие как отображение сроков выполнения проекта, информации о цене, количествах заявок и задач, а также об исполнителе, если он есть. При наведении на элемент отображается дополнительная информация о разделе, а при клике данный раздел открывается в новом окне.

9) Documents. Компонент отвечает за отображение и управление списком договоров и документов. Он включает в себя фильтрацию и поиск, отображение состояния загруженности данных. Фильтрация данных осуществляется на стороне клиента, что обеспечивает реактивное обновление интерфейса [6].

10) DocumentList. Этот компонент отображает элементы иерархического списка договоров, которые могут быть как папками, так и разделами. Он управляет раскрытием и сворачиванием папок, фильтрацией и отображением дочерних элементов. Папки могут содержать дочерние элементы, которые могут быть другими папками или разделами. Когда папка раскрыта, её дочерние элементы отображаются рекурсивно с увеличением отступа на каждом уровне вложенности. Рядом с разделами отображаются цветовые метки, зависящие от статуса соответствующего договора.

11) DocumentItem. Этот компонент отображает информацию о договоре, включая данные о дате начала и окончания, стоимости, статусе и других характеристиках договора. Внутри компонента реализованы различные функции, такие как отображение сроков выполнения, информации о стоимости, количестве заявок и задач, а также об исполнителе, если он есть. При наведении на элемент отображается дополнительная информация о договоре, а при клике данный договор открывается в новом окне.

12) DashboardHeader. Компонент представляет собой верхнюю панель навигации для дашборда. Он управляет отображением кнопок навигации для перехода на просмотра списка проектов и разделов, документов и задач, а также канбан-доски. Включает функциональность для отображения статистики (количество задач, документов и заявок для каждой формы собственности), а также отображает информацию о подписке пользователя.

13) KanbanTasksHeader. Компонент отображает дополнительные заголовки и поля для выбора группировки задач и способа их сортировки. При изменении выбранной группировки и способа сортировки выполняется обновление данных для отображения [7].

14) Kanban. Основной визуальный компонент канбан-доски. Также он предоставляет функции для перетаскивания задач между колонками и повторного получения информации о задачах с определенным временным интервалом.

Получает информацию об отображаемых задачах устройстве и методов обновления задач. Если пользователь заходит с мобильного устройства, компонент ничего не рендерит, что предотвращает отображение неподдерживаемой версии. Компонент также включает модальное окно для создания задач типа «свободная задача».

15) Filter. Компонент отображает интерактивные поля для фильтрации задач, представленных на канбан-доске. Фильтр включает поля выбора проекта, дат начала и окончания сроков задач, выбора типа задачи и исполнителя. Компонент также содержит логику загрузки справочных данных (списков проектов, пользователей и типов задач) и автоматической отправки данных фильтра, в случае внесения изменений, а также хранит значения полей фильтра на стороне пользователя.

16) CreateFreeTaskForm. Компонент представляет собой форму создания задачи типа «свободная задача» в виде модального окна. Форма включает поля выбора проекта, ввода текста задачи, выбора группы специальностей, исполнителя, указания контрольного срока задачи и переключатель для определения важности задачи. При изменении значений в полях проекта или специализации, сбрасывается значение исполнителя. Компонент также содержит логику загрузки справочных данных (списков проектов и пользователей), валидации и отправки введенных данных.

17) Column. Компонент реализует одну колонку канбан-доски, которая может отображаться в трех разных режимах: как пустая колонка, как статичная (без возможности перетаскивания задач), либо как динамичная колонка с поддержкой режима «drag-and-drop». Каждая задача отображается с помощью дочернего компонента Task.

18) Task. Компонент представляет собой визуальное отображение карточки задачи в колонке канбан-доски. Внутри компонента, если задача помечена как важная, отображается соответствующая метка с фоном, вычисленным динамически. Основной блок задачи включает в себя название группы, которая выделяется цветом, информацию о разделе и статусе, информацию об исполнителе, название самой задачи и дату. При этом дата визуально подсвечивается, если задача просрочена или подходит к сроку. Также, если задача находится в колонке, содержащей завершенные задачи, и при этом имеет статус проверки, рядом с ней отображается индикатор, сигнализирующий о результате проверки.

## 1.10. Интерфейсы программного обеспечения

В рамках работы модуля клиентское и серверное приложения обмениваются данными через специально предусмотренные интерфейсы. Способ связи — API, реализованное в соответствии с принципами REST. Клиентская часть отправляет запросы и получает ответы от сервера при помощи встроенного в браузер инструмента Fetch. В таблицах ниже представлены описания HTTP-запросов, содержащие исключительно информацию, относящуюся к данной предметной области, без включения вспомогательных или технически служебных данных [9].

В таблице 1.10 приведено описание запроса на сервер для получения списка форм собственности пользователя.

Таблица 1.10

Получение списка форм собственности пользователя

| URL запроса | https://pirs.online/user-type/get |
| --- | --- |
| Метод запроса | GET |
| Параметры запроса | Идентификатор формы собственности |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | Список форм собственности. Каждая форма собственности содержит идентификатор и название. |

В таблице 1.11 приведено описание запроса на сервер для получения списка форм собственности пользователя.

Таблица 1.11

Получение списка проектов

| URL запроса | https://pirs.online/project/get |
| --- | --- |
| Метод запроса | GET |
| Параметры запроса | Идентификатор формы собственности, поисковый запрос, статус проектов |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | Порядковый номер страницы, количество страниц |
| Тело ответа | Список проектов. Каждый проект содержит уникальный идентификатор, название, статус, количество заявок на исполнение, количество документов и задачи. |

В таблице 1.12 приведено описание запроса на сервер для получения списка форм собственности пользователя.

Таблица 1.12

Получение списка разделов

| URL запроса | https://pirs.online/process/get |
| --- | --- |
| Метод запроса | GET |
| Параметры запроса | Идентификатор формы собственности, идентификатор проекта |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | Иерархический список разделов. Каждый раздел содержит уникальный идентификатор, название, статус, даты начала и завершения работ, сумму оплаты и ФИО исполнителя. |

В таблице 1.13 приведено описание запроса на сервер для получения списка форм собственности пользователя.

Таблица 1.13

Получение списка документов

| URL запроса | https://pirs.online/document/get |
| --- | --- |
| Метод запроса | GET |
| Параметры запроса | Идентификатор формы собственности, идентификатор раздела |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | Иерархический список документов. Каждый документ содержит уникальный идентификатор, название, статусы подписания, дату подписания (при наличии), а также ФИО лиц, упомянутых в документе. |

В таблице 1.14 приведено описание HTTP запроса для получения списка задач на канбан-доске.

Таблица 1.14

Получение списка задач в канбан-доске

| URL запроса | https://pirs.online/kanban/get-tasks |
| --- | --- |
| Метод запроса | GET |
| Параметры запроса | Идентификатор формы собственности, данные фильтра по задачам |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | Список задач, распределенных между соответствующими колонками. Каждая задача содержит уникальный идентификатор, название, идентификаторы и названия группы и раздела, к которым она принадлежит, статус, информацию об исполнителе. |

В таблице 1.15 приведено описание HTTP запроса для перемещения задачи в следующую колонку.

Таблица 1.15

Перемещение задачи между колонками в канбан-доске

| URL запроса | https://pirs.online/kanban/complete-task |
| --- | --- |
| Метод запроса | PATCH |
| Параметры запроса | Идентификатор формы собственности, идентификатор задачи |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | - |

В таблице 1.16 приведено описание HTTP запроса для получения списка задач на канбан-доске.

Таблица 1.16

Возвращение задачи в колонку «Все задачи» в канбан-доске

| URL запроса | https://pirs.online/kanban/delete-task |
| --- | --- |
| Метод запроса | DELETE |
| Параметры запроса | Идентификатор формы собственности, идентификатор задачи |
| Тело запроса | - |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | - |

В таблице 1.17 приведено описание HTTP запроса для создания свободной задачи.

Таблица 1.17

| URL запроса | https://pirs.online/kanban/create-free-task |
| --- | --- |
| Метод запроса | POST |
| Параметры запроса | - |
| Тело запроса | Идентификатор проекта, текст задачи, идентификатор группы специальностей, идентификатор исполнителя, контрольный срок задачи и индикатор важности задачи. |
| Заголовки запроса | Токен авторизации |
| Заголовки ответа | - |
| Тело ответа | - |

# 2. РЕАЛИЗАЦИЯ ФУНКЦИЙ ИНТЕРАКТИВНОЙ ПАНЕЛИ УПРАВЛЕНИЯ ПРОЕКТИРОВАНИЕМ ОБЪЕКТОВ КАПИТАЛЬНОГО СТРОИТЕЛЬСТВА

## 2.1. Реализация функции «Просмотр иерархического списка проектов»

### 2.1.1. Реализация алгоритмов

Общая схема работы алгоритма просмотра иерархического списка проектов представлена на рис. 2.1.

Пользователь, находящийся в роли заказчика или проектировщика, открывает интерактивную панель управления объектами капитального строительства. При инициализации панели автоматически отправляется запрос на сервер для получения списка форм собственности.

Полученные данные представляют собой массив объектов, содержащих идентификатор и наименование каждой формы собственности. Пользователь выбирает одну из форм собственности, в ответ на данное действие отправляется запрос на сервер с целью получения перечня проектов, относящихся к выбранной форме собственности. Результат запроса представляет собой иерархический список, включающий проекты и их подчинённые подпроекты.

Отображенный список доступен для фильтрации. Пользователь может воспользоваться текстовым полем для ввода ключевого слова, по которому производится фильтрация списка проектов. Дополнительно пользователь может включить или выключить фильтр по статусу «Выполнено», чтобы отображались только завершенные или только незавершенные проекты.

Загрузка данных о проектах осуществляется с использованием динамической пагинации — при прокрутке списка или выполнении фильтрации серверу отправляется запрос на получение следующей порции данных в зависимости от текущих параметров (форма собственности, текст фильтра, статус выполнения). Полученные данные автоматически добавляются к уже отображаемым.

Общая схема работы алгоритма просмотра иерархического списка проектов

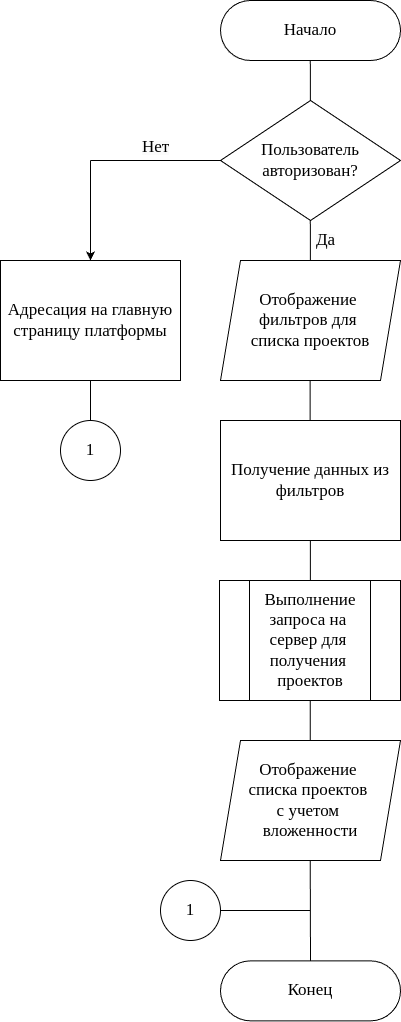


Рис. 2.1

Структура входных данных:

1) идентификатор текущего пользователя типа number;

2) массив форм собственности типа Array<{ id: number, title: string }> (получается с сервера при инициализации панели);

3) параметры фильтрации:

- строка фильтрации по названию проекта типа string;

- логическое значение статуса «Выполнено» типа boolean.

Промежуточными данными являются:

1) список проектов и подпроектов, получаемый с сервера, вида

Array<{ id: number, title: string, completed: boolean, parentId: Nullable<number>, dateEnd: string }>;

2) текущая страница пагинации типа number.

Структура выходных данных:

1) если пользователь авторизован — отображение компонентов React, включающих следующие структурные единицы:

- выпадающий список с элементами — для выбора формы собственности;

- список проектов для иерархического отображения;

- поле ввода для текстовой фильтрации;

- переключатель статуса «Выполнено»;

- сообщения об ошибках (валидация или ошибка фильтрации).

2) если пользователь авторизован, но при выполнении запроса возникла ошибка — отображение компонента React с сообщением ошибки;

3) если пользователь неавторизован — выходные данные отсутствуют.

### 2.1.2. Реализация компонентов

2.1.2.1. Компонент Projects представляет собой контейнер верхнего уровня, отвечающий за управление списком проектов, включая загрузку данных, выполнение пользовательских запросов и отображение состояния загрузки.

Сводка состояний (store) компонента:

1) Array<object> projectData – массив объектов, содержащий данные всех загруженных проектов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

Сводка функций компонента:

1) void handleSubmitQuery(string query) – отправка пользовательского запроса на фильтрацию или поиск проектов. Выполняет запрос к серверу и обновляет projectData на основе результатов;

2) void handleChangeQuery(string query) – обновление значения поискового запроса в локальном состоянии, но без немедленного выполнения запроса;

3) void handleToggleFulfilled() – переключение состояния pending, указывающего на выполнение или завершение операции (например, начала/окончания загрузки).

2.1.2.2. Компонент ProjectList отвечает за визуализацию списка проектов. Получает данные через пропсы и отображает каждый проект в виде отдельного элемента списка с возможностью взаимодействия (например, открытия подробной информации).

Сводка пропов компонента:

1) Array<object> projectData – массив объектов, содержащий данные всех загруженных проектов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

2.1.2.3. Компонент ProjectItem отображает отдельный проект внутри списка. Обеспечивает взаимодействие с пользователем: выбор проекта, открытие подробной информации, переход к деталям. Используется внутри компонента ProjectList.

Сводка функций компонента:

1) void handleClick(number id) – обработка клика по элементу проекта. Выбирает данный проект на основе его идентификатора;

2) void handleOpen(number id) – открытие подробной информации о проекте по переданному идентификатору id. Инициирует переход на другую страницу с подробной информацией о проекте по переданному идентификатору.

### 2.1.3. Описание контрольных примеров

Опишем контрольные примеры для функции просмотра иерархического списка проектов:

1) авторизованный пользователь в роли заказчика или исполнителя просматривает информацию о проектах на интерактивной панели, выбрав необходимую форму собственности. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.2;

Просмотр списка проектов

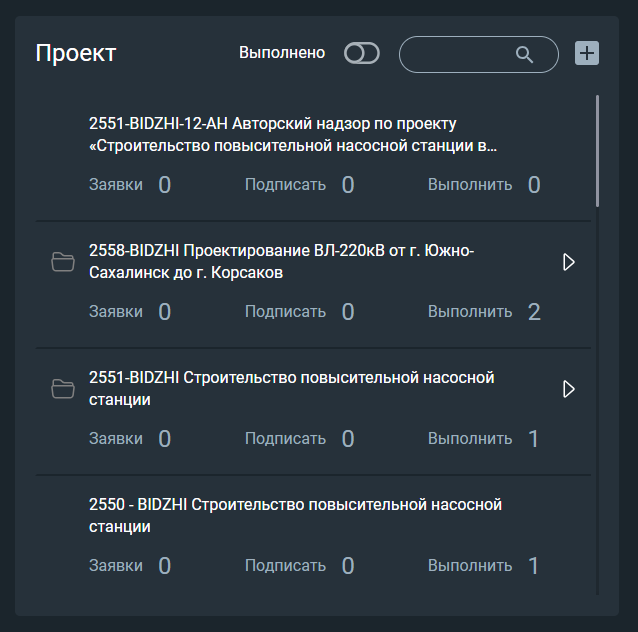


Рис. 2.2

2) авторизованный пользователь в роли заказчика или исполнителя запрашивает информацию о подпроектах на интерактивной панели, выбрав соответствующий комплексный проект. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.3;

Просмотр списка подпроектов комплексного проекта

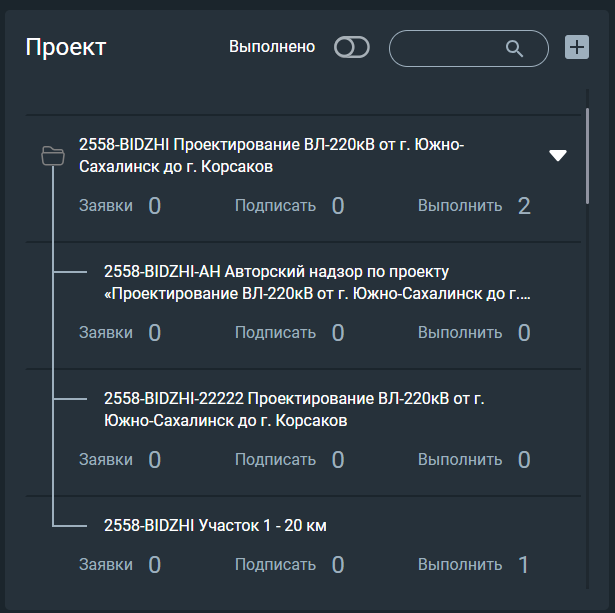


Рис. 2.3

3) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список проектов. При прокрутке списка вниз система инициирует динамическую подгрузку данных, автоматически запрашивая и отображая следующую порцию элементов без необходимости явного действия со стороны пользователя. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.4;

Инициация динамической подгрузки списка проектов

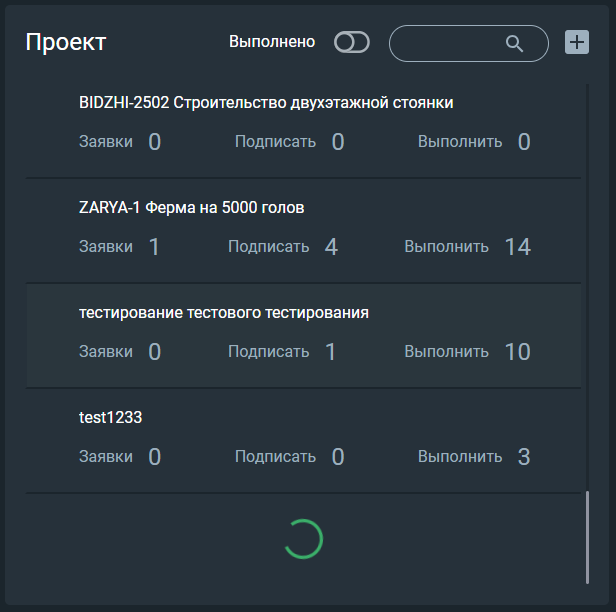


Рис. 2.4

4) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список проектов, предварительно отфильтровав их по названию. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.5;

Просмотр списка проектов, отфильтрованных по названию

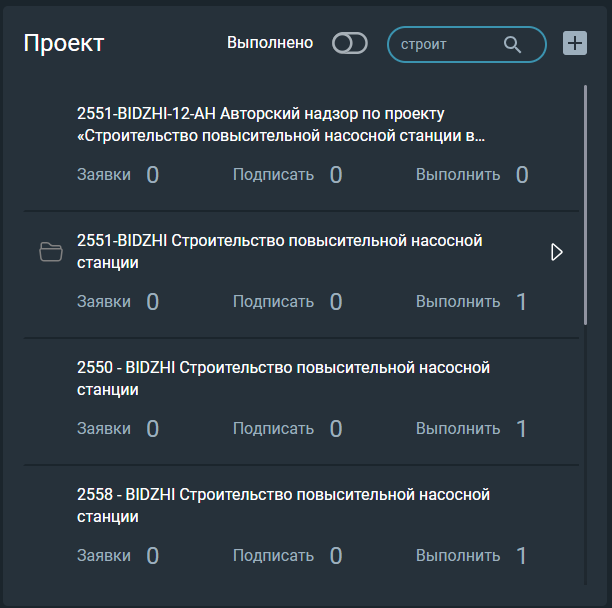


Рис. 2.5

5) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список проектов, предварительно отфильтровав их по статусу «Выполнено». Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.6;

Просмотр списка проектов, отфильтрованных по статусу «Выполнено»

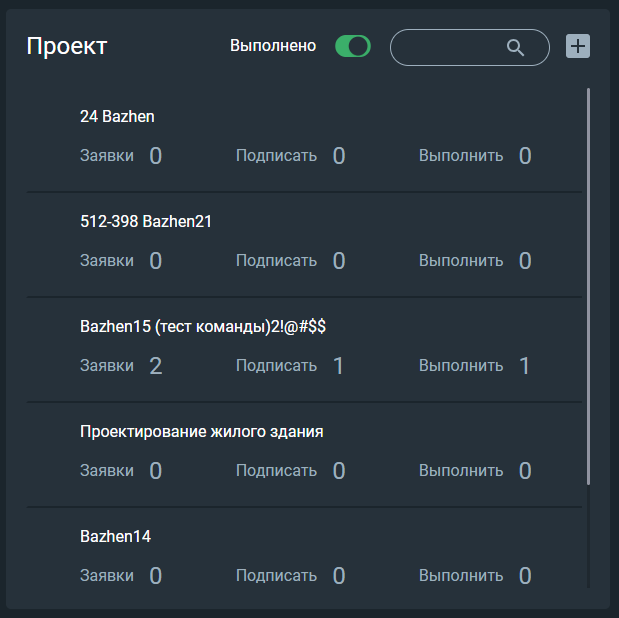


Рис. 2.6

## 2.2. Реализация функции «Просмотр иерархического списка разделов»

### 2.2.1. Реализация алгоритмов

Общая схема работы алгоритма просмотра иерархического списка разделов представлена на рис. 2.7.

Пользователь, находящийся в роли заказчика или проектировщика, заходит на панель управления проектированием объектов капитального строительства. После выбора формы собственности и проекта отправляется запрос на сервер с целью получения перечня всех разделов, относящихся к выбранному проекту.

Применяется рекурсивный алгоритм для окраски папок (категорий разделов) в зависимости от статуса вложенных разделов. Если хотя бы один раздел внутри папки находится в статусе «В работе», вся папка и все её подкатегории окрашиваются в желтый цвет. Если в папке есть раздел в статусе «приостановлен», то папка и её подкатегории окрашиваются в красный цвет. Статус «приостановлен» является более приоритетным, чем статус «в работе». Алгоритм проходит по всем уровням иерархии и обновляет цвета папок с учетом статусов разделов.

Пользователь также может использовать текстовое поле для поиска разделов по ключевому слову. В результате поиска система фильтрует иерархический список разделов, отображая только те, которые соответствуют запросу. Кроме того, пользователь может активировать фильтр для отображения только выполненных разделов. В этом случае система обновляет иерархический список, показывая только завершенные разделы и пересчитывает статусы папок, исключая невыполненные разделы.

Рекурсивное обновление списка разделов с учетом поиска и фильтрации гарантирует, что раскраска папок будет корректно обновлена в зависимости от оставшихся разделов. Алгоритм автоматически перестраивает иерархический список, визуализируя все изменения, такие как фильтрация по выполнению или по поисковому запросу.

Общая схема работы алгоритма просмотра иерархического списка разделов

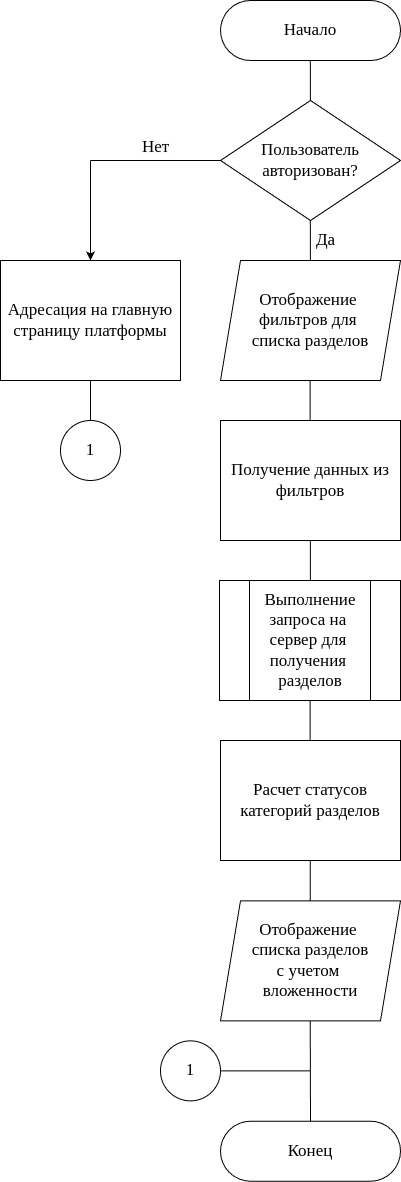


Рис. 2.7

Структура входных данных:

1) идентификатор текущего пользователя типа number;

2) массив форм собственности типа Array<{ id: number, title: string }> — получаемый с сервера при инициализации панели, представляет собой список доступных форм собственности для выбора пользователем.

3) идентификатор выбранного проекта типа number.

Промежуточные данные:

1) Список проектов и подпроектов, получаемый с сервера, вида: Array<{ id: number, title: string, parentId: Nullable<number>, status: number, dateEnd: string }.

Этот список содержит данные о разделах и категориях разделов, в том числе их идентификаторы, названия и статусы выполнения. Данные о разделах и их статусах используются для отображения и окрашивания иерархического списка.

Структура выходных данных:

1) если пользователь авторизован — отображение компонентов React, включающих следующие структурные единицы:

- выпадающий список с элементами — для выбора формы собственности;

- список разделов для иерархического отображения;

- поле ввода для текстовой фильтрации;

- переключатель статуса «Выполнено»;

- сообщения об ошибках (валидация или ошибка фильтрации).

2) если пользователь авторизован, но при выполнении запроса возникла ошибка — отображение компонента React с сообщением ошибки;

3) если пользователь неавторизован — выходные данные отсутствуют.

### 2.2.2. Реализация компонентов

2.2.2.1. Компонент Processes представляет собой контейнер верхнего уровня, отвечающий за управление иерархическим списком разделов, включая загрузку данных, выполнение пользовательских запросов и отображение состояния загрузки.

Сводка состояний (store) компонента:

1) Array<object> processData – массив объектов, содержащий данные всех загруженных разделов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

Сводка функций компонента:

1) void handleSubmitQuery(string query) – отправка пользовательского запроса на фильтрацию или поиск проектов. Выполняет запрос к серверу и обновляет projectData на основе результатов;

2) void handleChangeQuery(string query) – обновление значения поискового запроса в локальном состоянии, но без немедленного выполнения запроса;

3) void handleToggleFulfilled() – переключение состояния pending, указывающего на выполнение или завершение операции (например, начала/окончания загрузки).

2.1.2.2. Компонент ProcessList отвечает за визуализацию списка разделов. Получает данные через пропсы и отображает каждый раздел в виде отдельного элемента списка с возможностью взаимодействия (например, открытия подробной информации).

Сводка пропов компонента:

1) Array<object> processData – массив объектов, содержащий данные всех загруженных разделов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

2.1.2.3. Компонент ProcessItem отображает отдельный раздел внутри списка. Обеспечивает взаимодействие с пользователем: выбор раздела, открытие подробной информации, переход к деталям. Используется внутри компонента ProcessList.

Сводка функций компонента:

1) void handleClick(number id) – обработка клика по элементу раздел. Выбирает данный раздел на основе его идентификатора;

2) void handleOpen(number id) – открытие подробной информации о разделе по переданному идентификатору. Инициирует переход на другую страницу с подробной информацией о разделе по переданному идентификатору.

### 2.2.3. Описание контрольных примеров

Опишем контрольные примеры для функции просмотра иерархического списка разделов:

1) авторизованный пользователь в роли заказчика или исполнителя просматривает информацию о разделах на интерактивной панели, выбрав необходимые форму собственности и проект. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.8;

Просмотр списка разделов

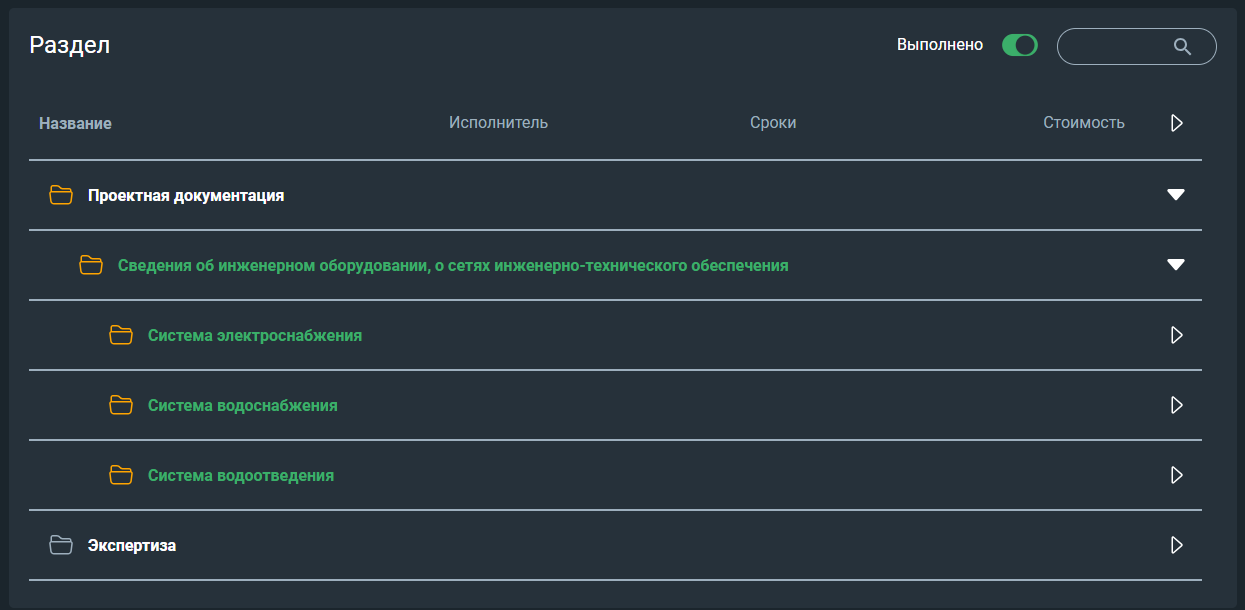


Рис. 2.8

2) авторизованный пользователь в роли заказчика или исполнителя просматривает информацию о разделах на интерактивной панели, выбрав соответствующую категорию разделов. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.9;

Просмотр списка разделов соответствующей категории разделов

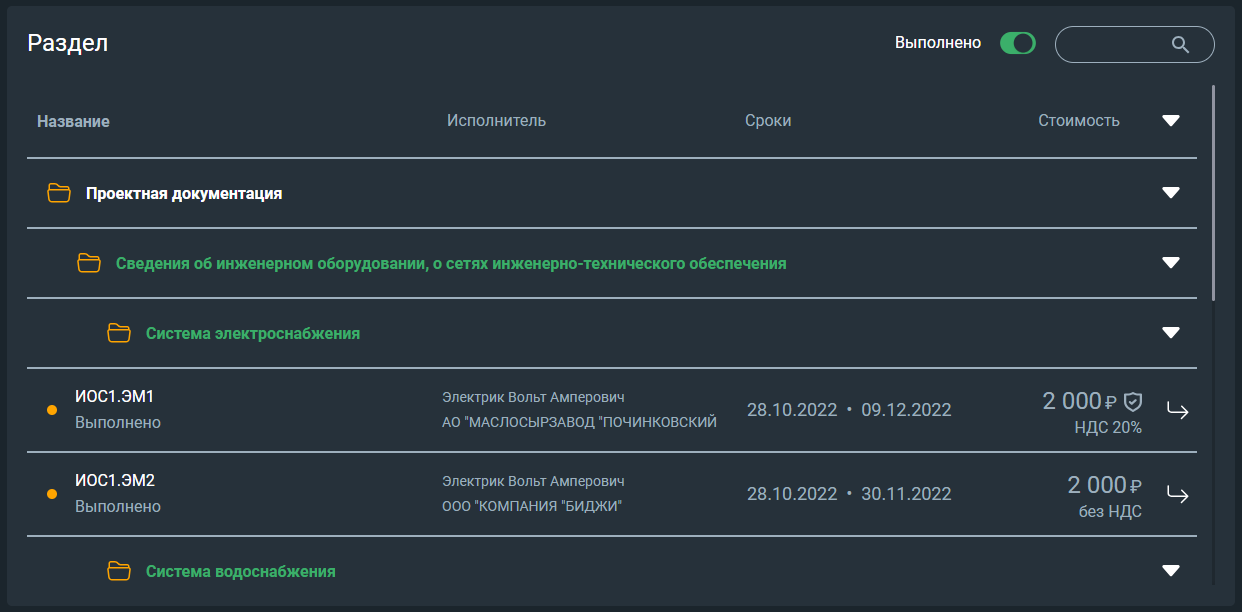


Рис. 2.9

3) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список разделов, предварительно отфильтровав их по названию. Иконки папок категорий разделов автоматически перекрашиваются в соответствие со статусами вложенных разделов. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.10;

Просмотр списка разделов, отфильтрованных по названию

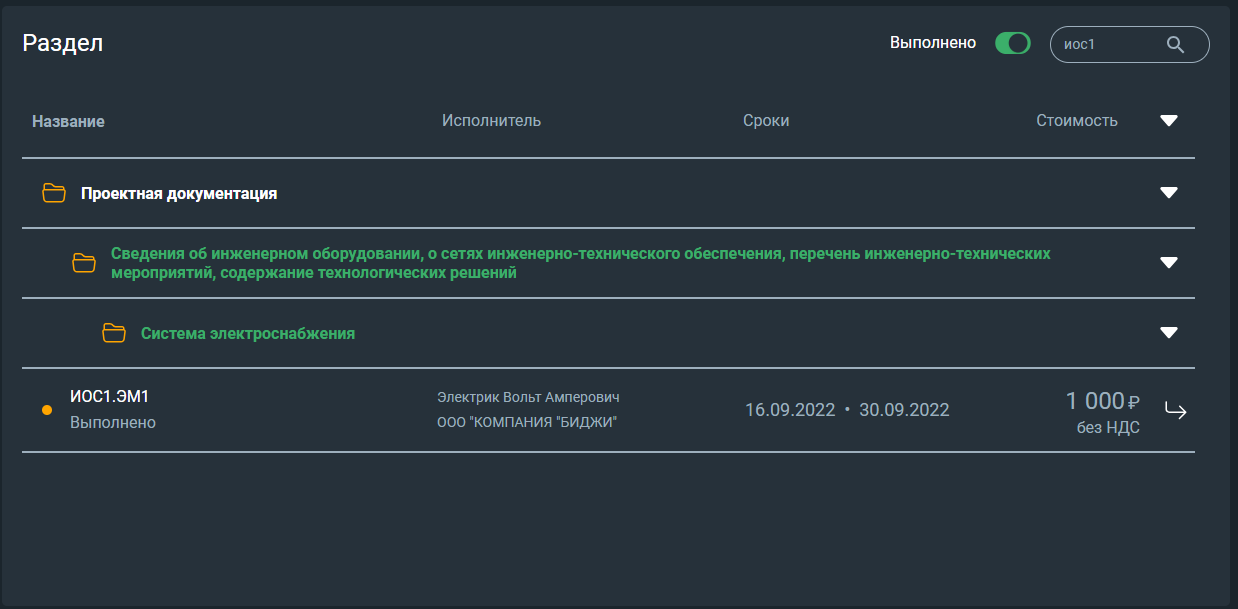


Рис. 2.10

5) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список разделов, предварительно отфильтровав их по статусу «Выполнено». Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.11;

Просмотр списка разделов, отфильтрованных по статусу «Выполнено»

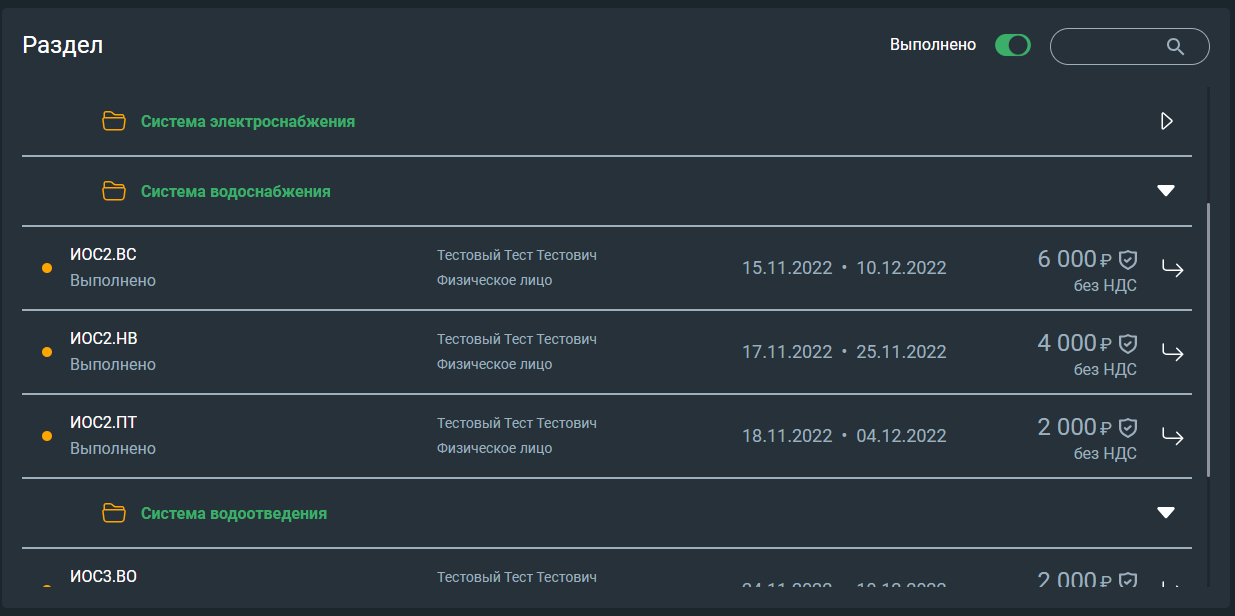


Рис. 2.11

## 

## 2.3. Реализация функции «Просмотр иерархического списка документов»

### 2.3.1. Реализация алгоритмов

Общая схема работы алгоритма просмотра иерархического списка документов представлена на рис. 2.12.

Пользователь, находящийся в роли заказчика или проектировщика, заходит на панель управления проектированием объектов капитального строительства. После выбора формы собственности, проекта и раздела отправляется запрос на сервер с целью получения перечня всех документов, относящихся к выбранному проекту. Документы классифицируются по соответствующим папкам в зависимости от их типа: «Акты», «Отчеты», «Договоры».

Для документов типа «Договор» предусмотрено визуальное отображение статуса подписания, включая информацию о сторонах, подписавших документ, а также даты подписания. В зависимости от текущего статуса подписания отображаются соответствующие индикаторы:

1) не подписан ни одной стороной;

2) подписан одной стороной;

3) подписан обеими сторонами.

Пользователю доступна функция текстового поиска по ключевым словам. При вводе поискового запроса система фильтрует список и отображает только те документы, которые соответствуют заданному критерию.

Дополнительно предусмотрена возможность включения фильтра, позволяющего отобразить только расторгнутые договоры. При этом остальные типы документов продолжают отображаться в общем списке.

Общая схема работы алгоритма просмотра списка документов

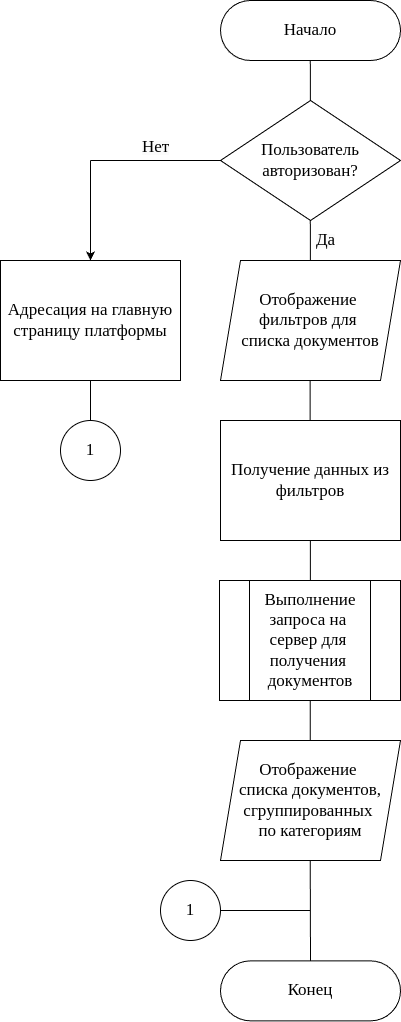


Рис. 2.12

Структура входных данных:

1) идентификатор текущего пользователя типа number;

2) массив форм собственности типа Array<{ id: number, title: string }> — получаемый с сервера при инициализации панели, представляет собой список доступных форм собственности для выбора пользователем.

3) идентификатор выбранного проекта типа number;

4) идентификатор выбранного раздела типа number;

Промежуточные данные:

1) Список документов, получаемых с сервера, вида: Array<{ id: number, title: string, signature: Array<number>, dates: Array<string>; }.

Этот список содержит данные о документах, в том числе их идентификаторы, названия, статусы и даты подписания. Данные о типах документов используются для дальнейшей классификации.

Структура выходных данных:

1) если пользователь авторизован — отображение компонентов React, включающих следующие структурные единицы:

- выпадающий список с элементами — для выбора формы собственности;

- список документов для отображения;

- поле ввода для текстовой фильтрации;

- переключатель статуса «Выполнено»;

- сообщения об ошибках (валидация или ошибка фильтрации).

2) если пользователь авторизован, но при выполнении запроса возникла ошибка — отображение компонента React с сообщением ошибки;

3) если пользователь неавторизован — выходные данные отсутствуют.

Алгоритм получения списка проектов, разделов и документов представлен на рис. 2.13.

На первом этапе алгоритма осуществляется получение данных фильтров, введенных пользователем в интерфейсе. Эти значения формируют основу для последующего запроса.

На втором этапе выполняется формирование запроса, в который включаются все параметры, указанные пользователем. Структура запроса соответствует ожидаемому серверным API формату, обеспечивая корректную передачу фильтрационных условий. На этом этапе происходит сериализация параметров в подходящую структуру данных.

Третий этап заключается в отправке запроса на соответствующую контрольную точку серверного API.

На завершающем этапе происходит обработка полученных данных. После получения ответа от сервера система проверяет корректность и целостность возвращаемого результата. В случае успешного ответа выполняется обновление соответствующих интерфейсных элементов: списков проектов, разделов и документов.

Схема работы алгоритма получения списка проектов, разделов и документов



Рис. 2.13

### 2.3.2. Реализация компонентов

2.3.2.1. Компонент Documents представляет собой контейнер верхнего уровня, отвечающий за управление списком документов, включая загрузку данных, выполнение пользовательских запросов и отображение состояния загрузки.

Сводка состояний (store) компонента:

1) Array<object> documentData – массив объектов, содержащий данные всех загруженных документов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

Сводка функций компонента:

1) void handleSubmitQuery(string query) – отправка пользовательского запроса на фильтрацию или поиск проектов. Выполняет запрос к серверу и обновляет projectData на основе результатов;

2) void handleChangeQuery(string query) – обновление значения поискового запроса в локальном состоянии, но без немедленного выполнения запроса;

3) void handleToggleFulfilled() – переключение состояния pending, указывающего на выполнение или завершение операции (например, начала/окончания загрузки).

2.3.2.2. Компонент DocumentList отвечает за визуализацию списка документов. Получает данные через пропсы и отображает каждый документ в виде отдельного элемента списка с возможностью взаимодействия.

Сводка пропов компонента:

1) Array<object> documentData – массив объектов, содержащий данные всех загруженных документов, доступных пользователю;

2) number userType – числовое значение, указывающее форму собственности текущего пользователя;

3) boolean pending – булевый индикатор, сигнализирующий о том, находится ли компонент в состоянии загрузки данных (true – данные загружаются, false – загрузка завершена).

2.3.2.3. Компонент DocumentItem отображает отдельный документ внутри списка. Обеспечивает взаимодействие с пользователем: выбор документа, открытие подробной информации, переход к деталям. Используется внутри компонента DocumentList.

Сводка функций компонента:

1) void handleClick(number id) – обработка клика по элементу документа. Выбирает данный документ на основе его идентификатора;

2) void handleOpen(number id) – открытие подробной информации о документе по переданному идентификатору. Инициирует переход на другую страницу с подробной информацией о документе по переданному идентификатору.

### 2.3.3. Описание контрольных примеров

Опишем контрольные примеры для функции просмотра иерархического списка документов:

1) авторизованный пользователь в роли заказчика или исполнителя просматривает категории документов на интерактивной панели, выбрав необходимые форму собственности, проект и раздел. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.14;

Просмотр списка документов

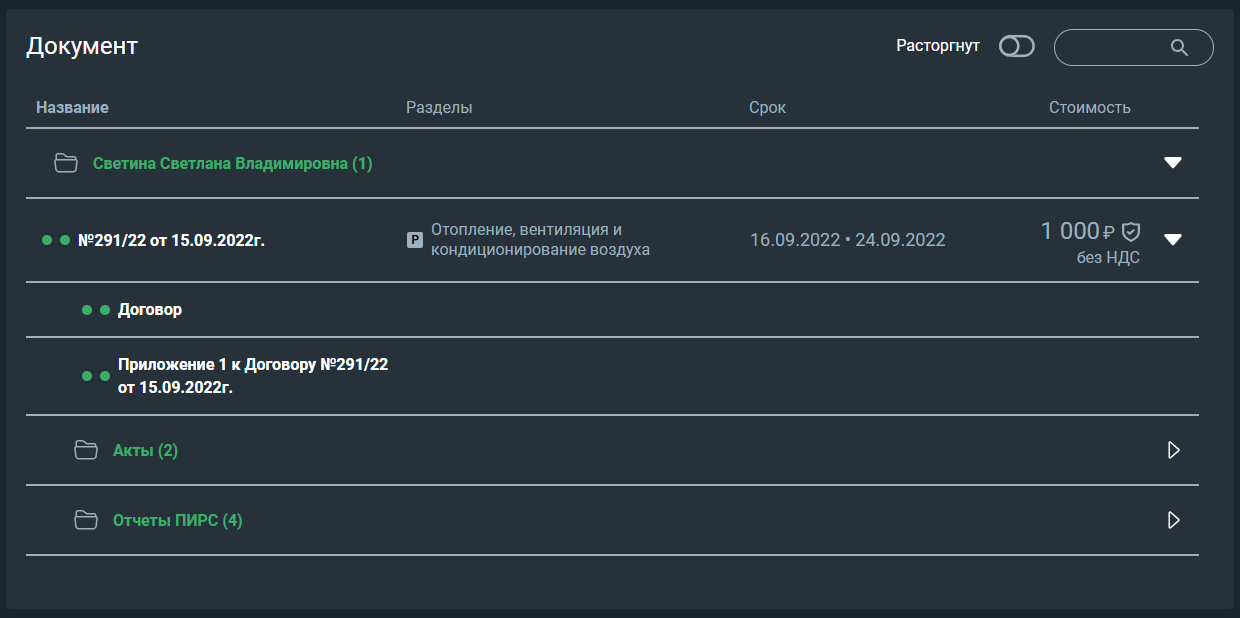


Рис. 2.14

2) авторизованный пользователь в роли заказчика или исполнителя просматривает информацию о документах на интерактивной панели, выбрав соответствующий тип документов. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.15;

Просмотр списка документов соответствующего типа

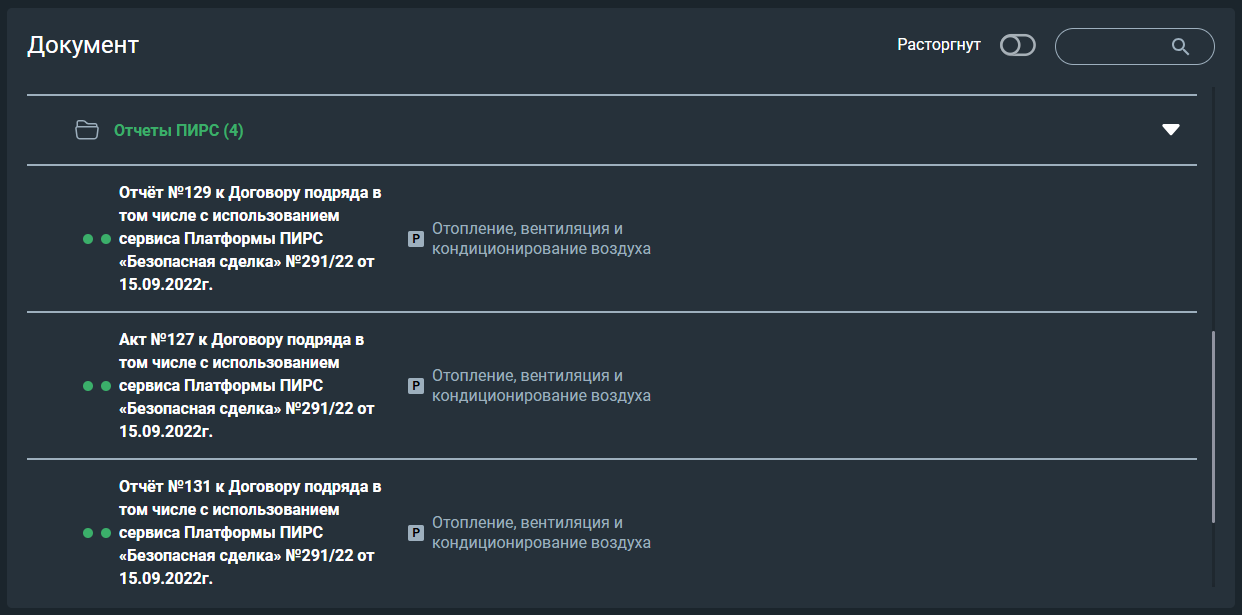


Рис. 2.15

4) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список документов, предварительно отфильтровав их по названию. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.16;

Просмотр списка документов, отфильтрованных по названию

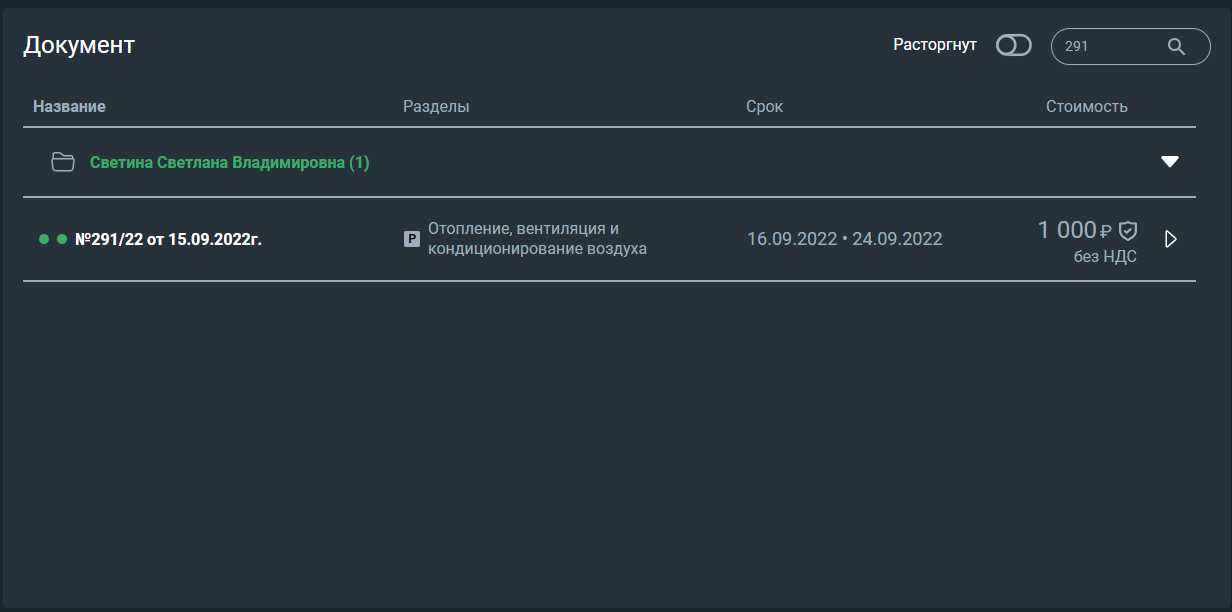


Рис. 2.16

5) авторизованный пользователь, находящийся в роли заказчика или исполнителя, просматривает список документов, предварительно отфильтровав договоры по статусу «Расторгнут». Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.17;

Просмотр списка документов, отфильтрованных по статусу «Расторгнут»

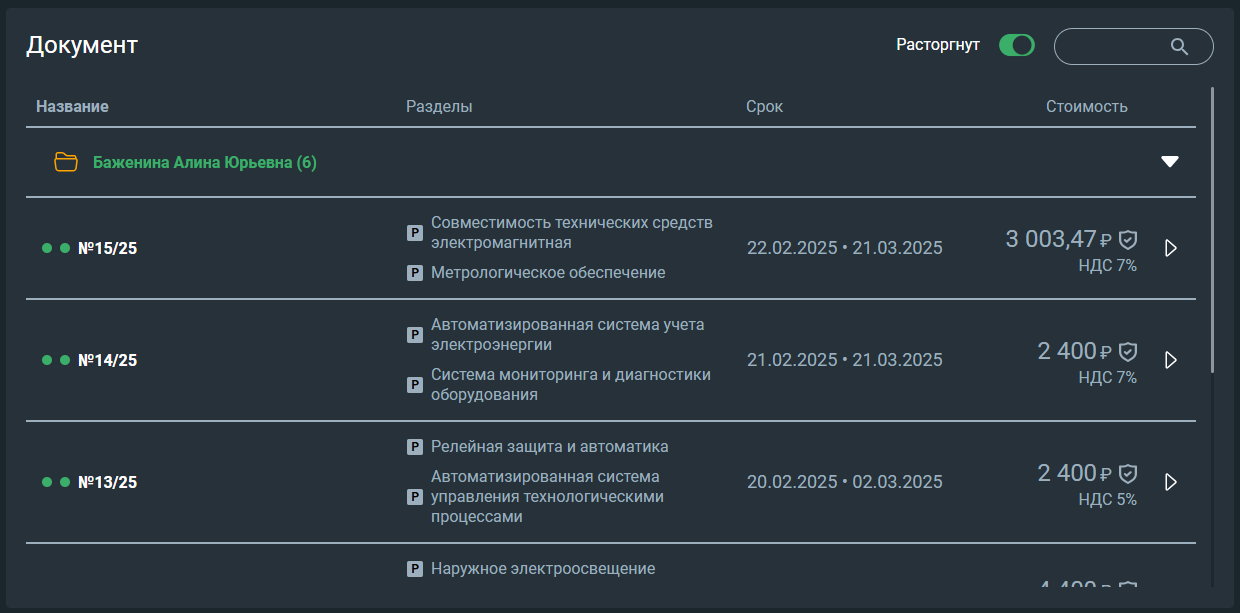


Рис. 2.17

## 2.4. Реализация функции «Управление задачами с помощью канбан-доски»

### 2.4.1. Реализация алгоритмов

Общая схема работы алгоритма управления задачами с помощью канбан-доски представлена на рис. 2.18.

После открытия интерактивной панели управления проектированием объектов капитального строительства пользователь переходит на вкладку канбан-доски. При этом выполняется запрос к серверу на получение списка задач, каждая из которых содержит информацию о своем текущем статусе. Полученные задачи распределяются по колонкам: «Все задачи», «В работе», «На проверке», «Выполнено».

Пользователю отображается интерфейс канбан-доски, включающий фильтр и список задач, распределенных по соответствующим колонкам. Пользователь может перемещать задачи между колонками. При попытке перемещения задачи выполняется проверка, разрешен ли данный переход.

Если перемещение разрешено, возможны следующие сценарии:

1) перемещение задачи в колонку «Все задачи». Задаче присваивается статус `план`, и отправляется соответствующий запрос на обновление статуса на сервере.

2) перемещение задачи в колонку «В работе». Задаче присваивается статус «В работе», и отправляется запрос на обновление статуса.

3) завершение задачи — перемещение в колонку «На проверке» или «Выполнено». Выполняется проверка, требуется ли задача к проверке. Если требуется проверка — присваивается статус «на проверке», отправляется запрос на обновление статуса. Если проверка не требуется — присваивается статус «выполнено», отправляется запрос на обновление статуса.

После успешной обработки запроса задача перемещается в соответствующую колонку согласно новому статусу.

Общая схема работы алгоритма управления задачами

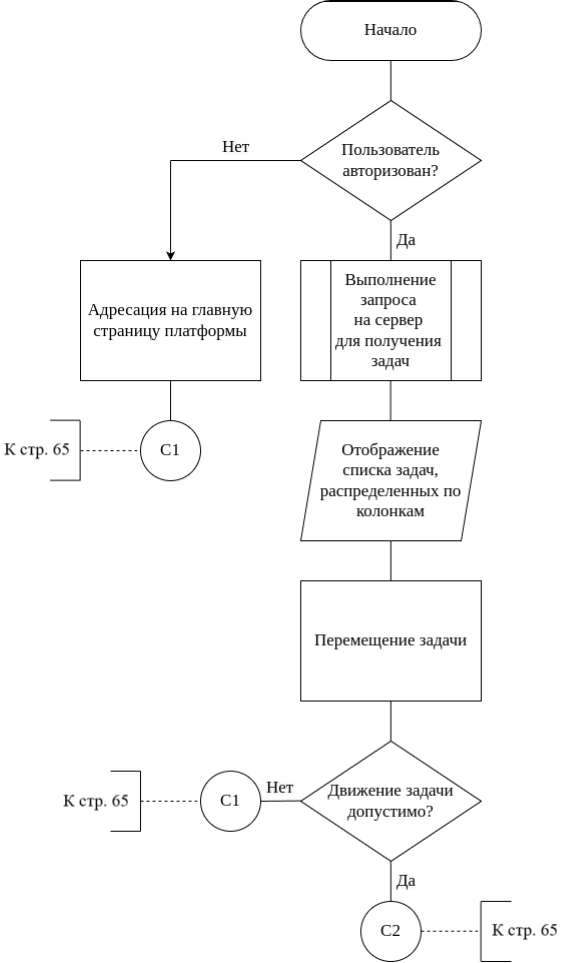
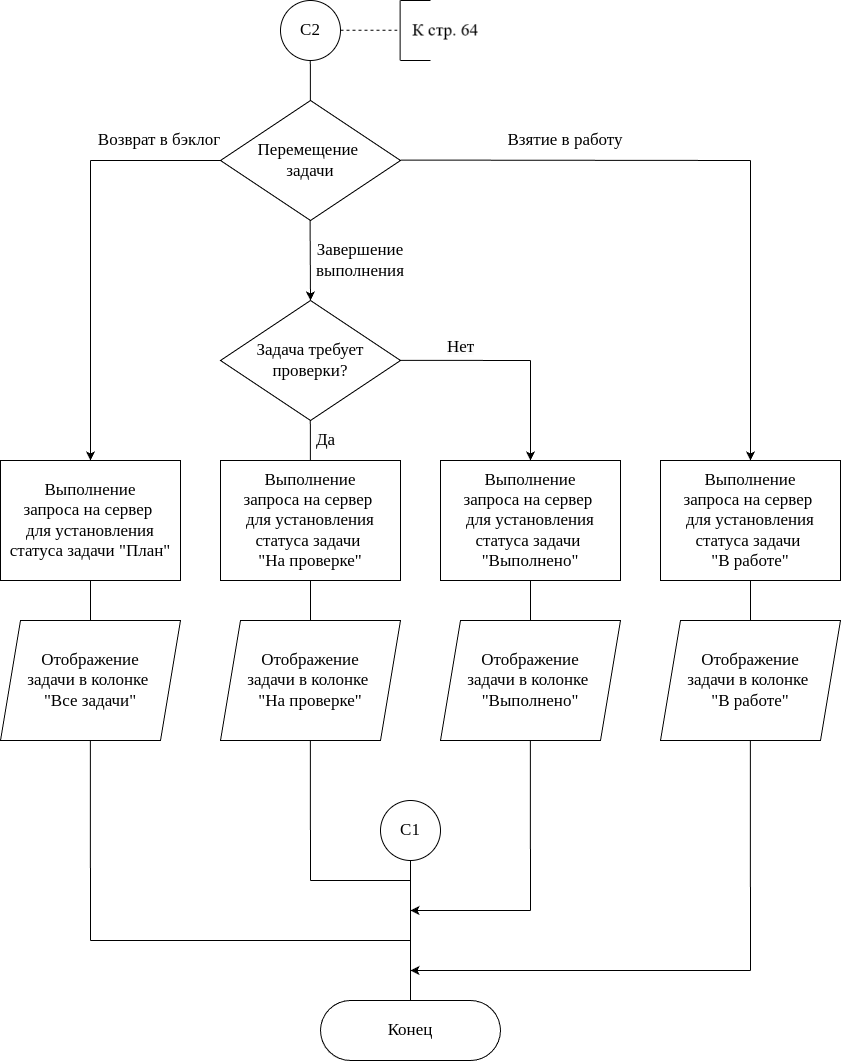


Рис. 2.18



Продолжение рис. 2.18

Структура входных данных:

1) идентификатор выбранной формы собственности типа Nullable<number>;

2) идентификатор проекта типа Nullable<number>;

3) дата начала задач Nullable<Date>;

4) дата начала сроков задач типа Nullable<Date>;

5) дата окончания сроков задач типа Nullable<Date>;

6) идентификатор типа задачи типа Nullable<number>;

7) идентификатор специализации исполнителя типа Nullable<number>;

8) идентификатор исполнителя типа Nullable<number>;

9) способ сортировки задач типа string;

10) способ группировки задач типа Nullable<number>.

Промежуточные данные:

1) Список задач, получаемых с сервера, вида: Array<{ id: number, title: string, code: Nullable<string>, executorName: string, executorId: number, dateDeadline: string, processName: string, processId: number, status: string }>.

Этот список содержит данные о задачах, в том числе их идентификаторы, названия, сроки исполнения, информацию об исполнителях и информацию о разделе. Данные о статусах задач используются для дальнейшей классификации.

Структура выходных данных:

1) если пользователь авторизован — отображение компонентов React, включающих следующие структурные единицы:

- колонки, соответствующие статусам задач «Все задачи», «В работе», «На проверке» и «Выполнено»;

- задачи, распределенные по колонкам в соответствии с текущим статусом;

- поддержка drag-n-drop для перемещения задач между колонками с обработкой сценариев, включающих назначение нового статуса, проверку валидности перехода и отправку запроса на сервер;

- сообщения об ошибках (валидация или ошибка фильтрации).

2) если пользователь авторизован, но при выполнении запроса возникла ошибка — отображение компонента React с сообщением ошибки;

3) если пользователь неавторизован — выходные данные отсутствуют.

### 2.4.2. Реализация компонентов

2.4.2.1. Компонент Kanban представляет собой контейнер верхнего уровня для отображения и управления канбан-доской. Он отвечает за загрузку задач, их распределение по статусам, обработку логики перемещения между колонками и обновление состояния задач.

Сводка состояний (store) компонента:

1) kanbanTasks: Array<object> – массив объектов, содержащий сгруппированные по статусу задачи:

2) handleDragEnd(task: object, from: string, to: string): void – функция, вызываемая при завершении перетаскивания задачи. Реализует проверку допустимости перемещения, определение целевого статуса (в зависимости от логики проверки), отправку запроса на сервер, а также обновление состояния задач при успешной операции.

2.4.2.2. Компонент Column представляет собой одну из колонок канбан-доски, отображающую список задач с определенным статусом. Он получает от родительского компонента данные для отображения и предоставляет интерфейс для drag-and-drop задач внутри колонки.

Сводка пропов компонента:

1) position: string – ключ статуса, например «в работе, используется для идентификации колонки;

2) header: string – заголовок колонки, отображаемый в UI;

3) tasks: Array<object> – список задач, соответствующих текущему статусу.

2.4.2.3. Компонент Task представляет собой отдельную карточку задачи на канбан-доске. Он отображает основную информацию о задаче и участвует в механизме drag-and-drop.

Сводка пропов компонента:

1) task: object – объект задачи, включающий следующие поля:

2) handleDragEnd(task: object, from: string, to: string): void – функция, вызываемая при завершении перетаскивания задачи.

### 2.4.3. Описание контрольных примеров

Опишем контрольные примеры для функции управления задачами с помощью канбан-доски:

1) авторизованный пользователь в роли заказчика или исполнителя просматривает информацию о задачах на канбан-доске интерактивной панели, выбрав необходимую форму собственности. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.20;

Просмотр информации о задачах на канбан-доске

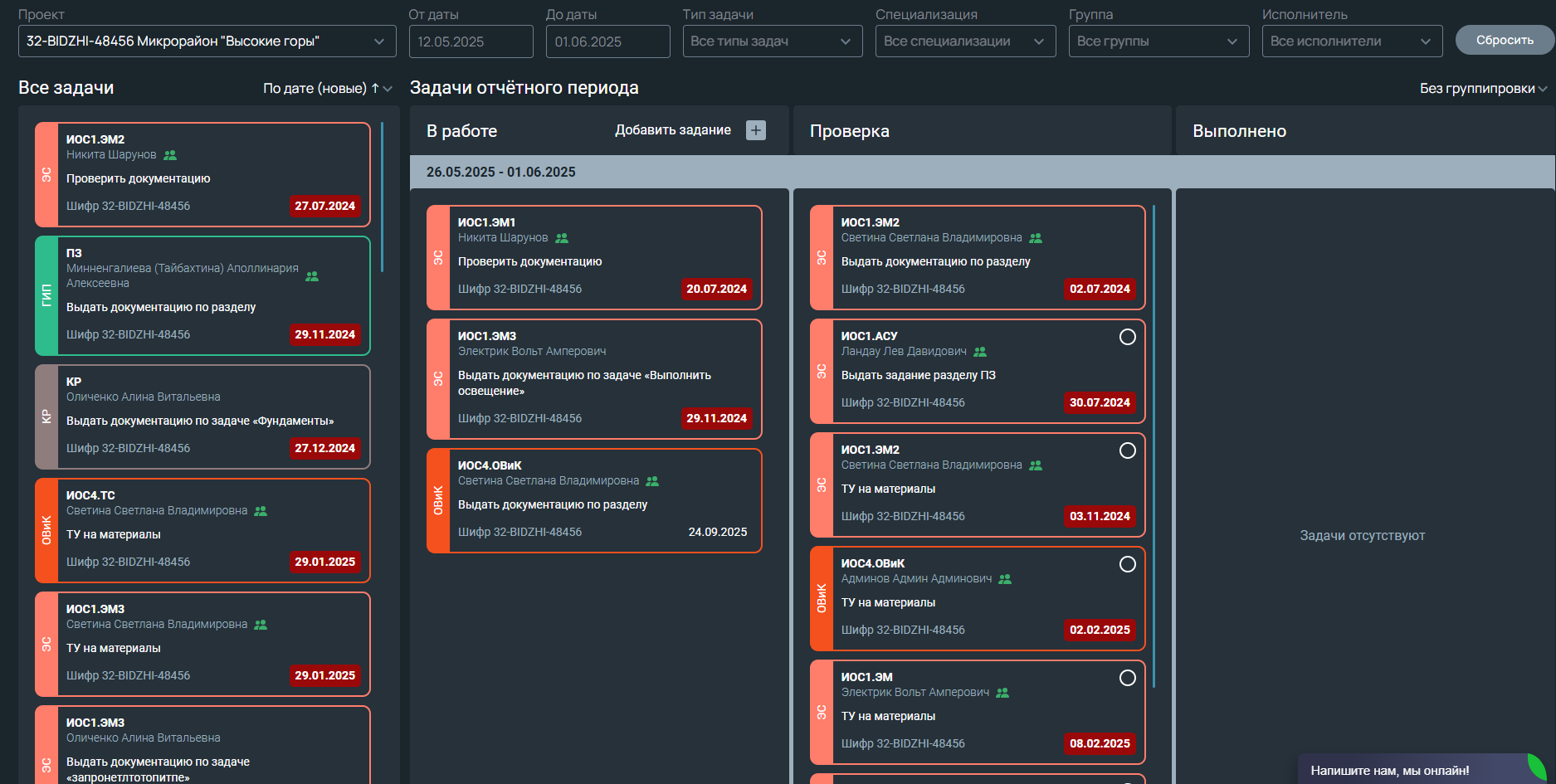


Рис. 2.19

2) авторизованный пользователь в роли заказчика или исполнителя перетаскивает задачу из колонки «Все задачи» в колонку «В работе». Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.20;

Перетаскивание задачи из колонки «Все задачи» в колонку «В работе»

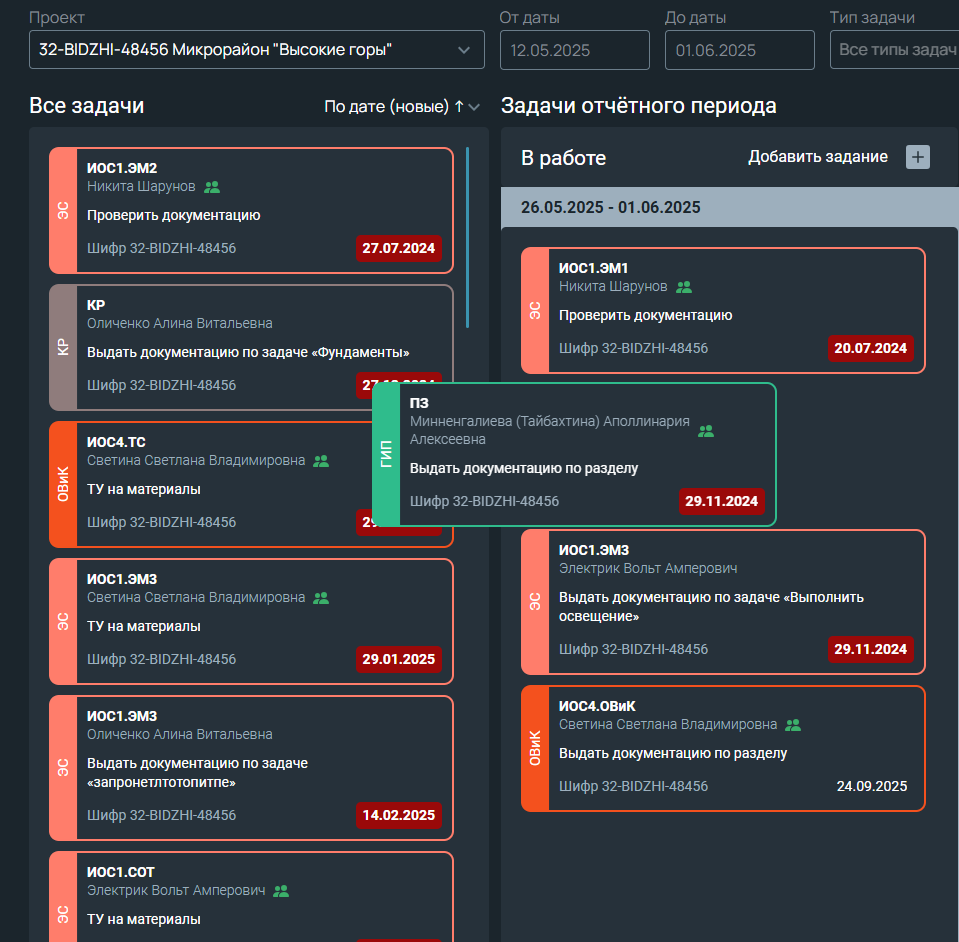


Рис. 2.20

3) авторизованный пользователь в роли заказчика или исполнителя переместил задачу из колонки «Все задачи» в колонку «В работе». Задача успешно переместилась в соответствующую колонку. Экранная форма представлена на рис. 2.21;

Результат перемещения задачи из колонки «Все задачи» в колонку «В работе»

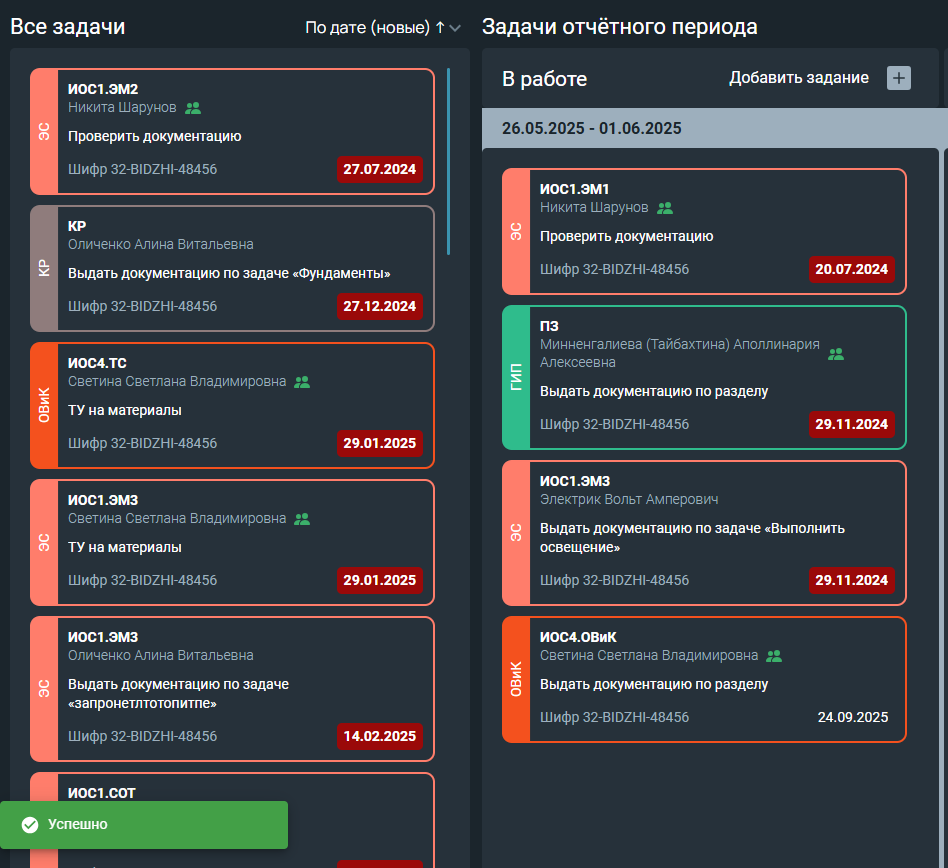


Рис. 2.21

4) авторизованный пользователь в роли заказчика или исполнителя перетаскивает задачу из колонки «В работе» в колонку «На проверке». Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.22;

Перетаскивание задачи из колонки «В работе» в колонку «На проверке»

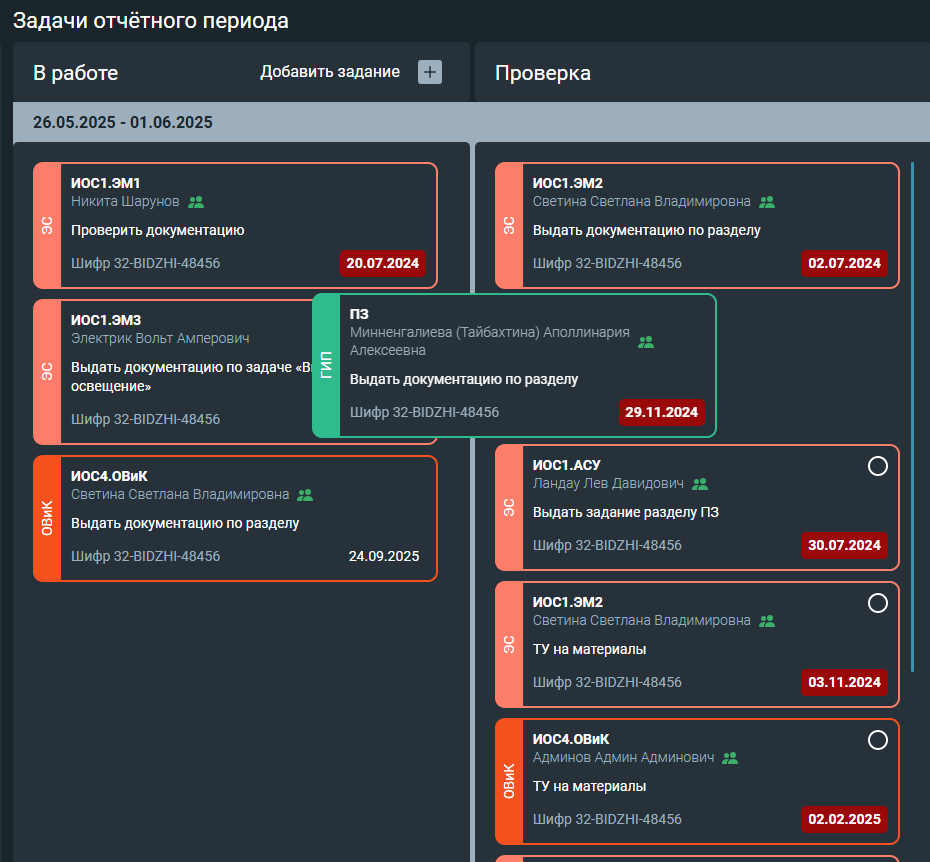


Рис. 2.22

5) авторизованный пользователь в роли заказчика или исполнителя переместил задачу из колонки «В работе» в колонку «На проверке». Задача успешно переместилась в соответствующую колонку. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.23;

Перетаскивание задачи из колонки «На проверке» в колонку «Выполнено»

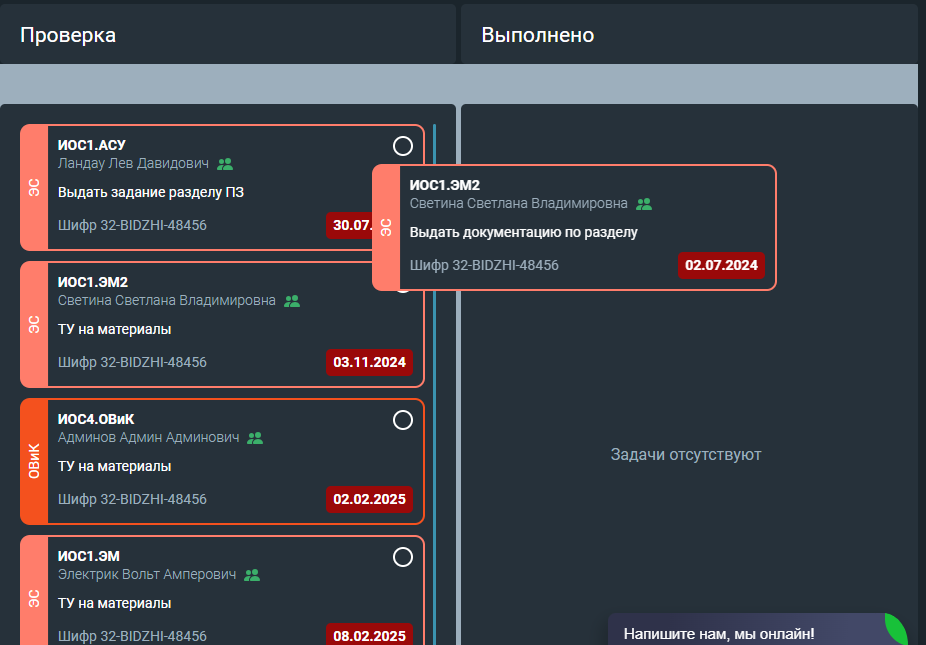


Рис. 2.23

7) авторизованный пользователь в роли заказчика или исполнителя переместил задачу из колонки «На проверке» в колонку «Выполнено». Задача успешно переместилась в соответствующую колонку. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.24;

Результат перемещения задачи в колонку «Выполнено»

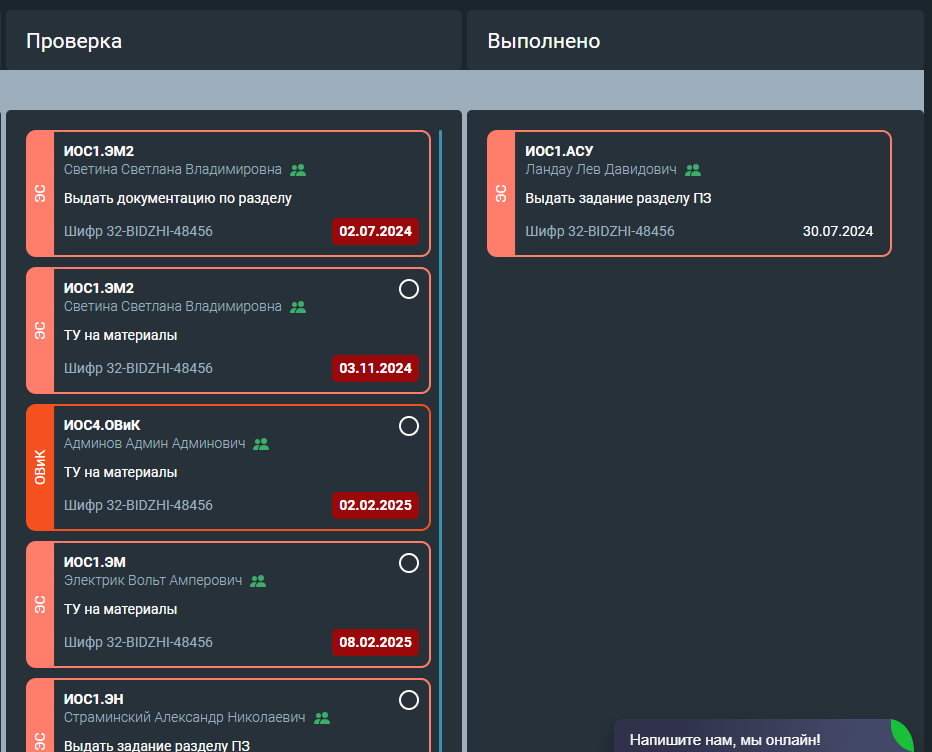


Рис. 2.24

8) авторизованный пользователь, находящийся в роли заказчика или исполнителя, осуществляет попытку перемещения задачи между колонками канбан-доски с использованием недопустимого перехода (запрещённого маршрута движения задачи). В результате система автоматически возвращает задачу в исходную колонку и отображает соответствующее сообщение об ошибке. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.25.

Попытка перемещения задачи между колонками канбан-доски с использованием недопустимого перехода

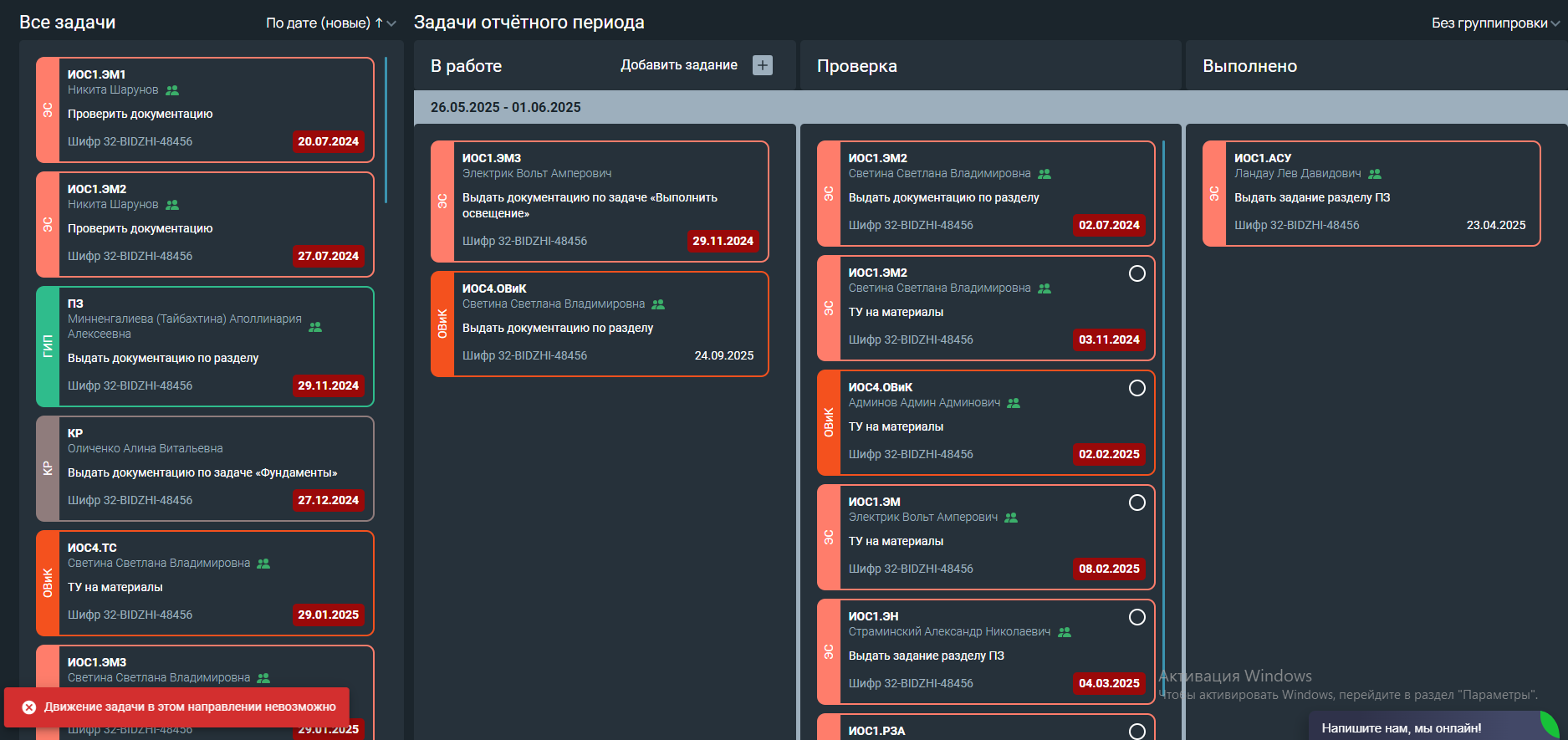


Рис. 2.25

## 2.5. Реализация функции «Создание свободной задачи»

### 2.5.1. Реализация алгоритмов

Общая схема работы алгоритма создания свободной задачи представлена на рис. 2.26.

Когда пользователь нажимает кнопку «Создать задачу», расположенную в верхней части колонки «В работе» канбан-доски, на экране появляется модальное окно с формой создания новой задачи. В момент открытия формы автоматически выполняются два запроса: первый — на получение списка доступных проектов, второй — на получение списка специализаций. Данные загружаются и отображаются в соответствующих полях формы.

Когда пользователь выбирает специализацию, отправляется дополнительный запрос на сервер для получения списка исполнителей, доступных для выбранной специализации. После этого пользователь может выбрать конкретного исполнителя из обновленного списка.

Пользователь заполняет поля формы: выбирает проект, указывает текст задачи, выбирает специализацию и исполнителя, устанавливает контрольный срок выполнения. При необходимости пользователь может также отметить задачу как важную, используя переключатель.

Перед отправкой формы проводится валидация введенных данных - проверка, что все обязательные поля заполнены, а указанный срок выполнения корректен. Если все проверки пройдены успешно, формируется и отправляется запрос на создание задачи. После успешного ответа от сервера модальное окно закрывается, и новая карточка задачи автоматически добавляется в колонку «В работе» на канбан-доске.

Общая схема работы алгоритма создания свободной задачи

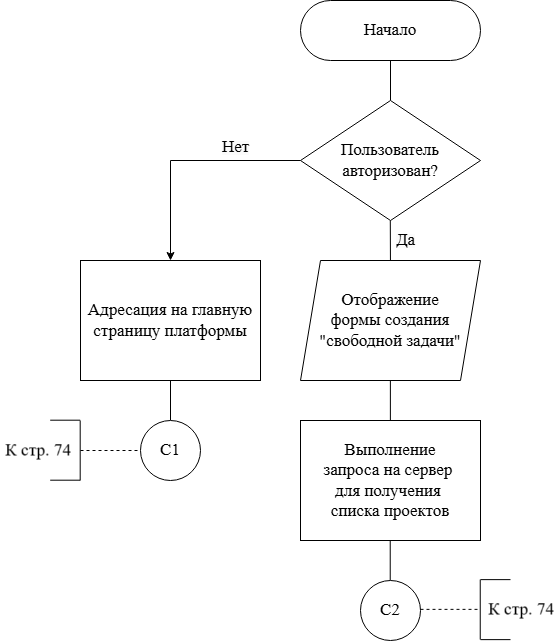
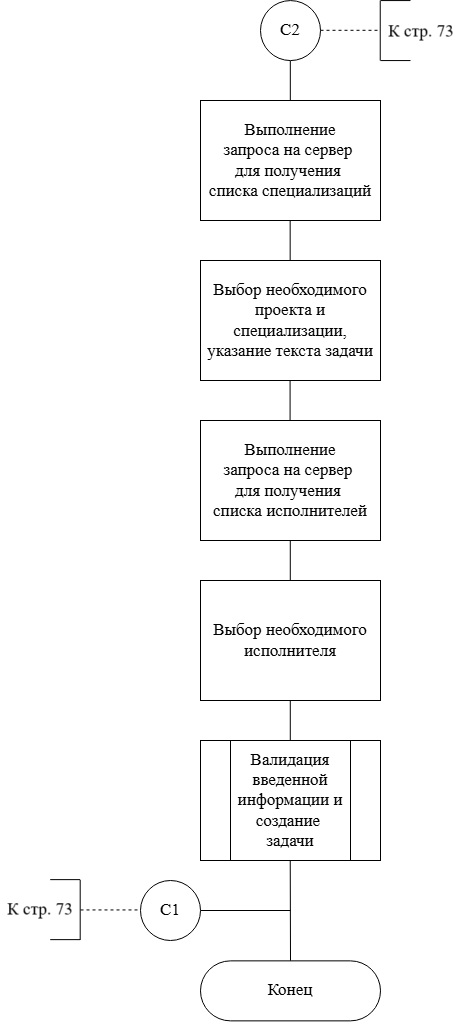


Рис. 2.26



Продолжение рис. 2.26

Структура входных данных:

1) идентификатор текущего пользователя типа number;

2) список проектов типа Array<{ id: number, title: string }> (получается с сервера при открытии формы создания задачи);

3) список специализаций типа Array<{ id: number, title: string }> (получается с сервера при открытии формы создания задачи);

4) список исполнителей по специализации типа Array<{ id: number, fullName: string }> (получается с сервера после выбора специализации пользователем);

5) идентификатор текущей формы собственности типа number.

Промежуточными данными являются:

1) идентификатор выбранной специализации типа number;

2) идентификатор выбранного проекта типа number;

3) идентификатор выбранного исполнителя типа number;

4) текст задачи типа string;

5) контрольный срок типа string;

6) логический флаг важности задачи типа boolean;

7) состояние валидации формы: корректность и полнота заполнения всех обязательных полей.

Структура выходных данных:

1) если все поля заполнены корректно — отправка запроса на создание задачи и отображение новой карточки задачи в колонке «В работе» на канбан-доске;

2) если форма заполнена некорректно — отображение сообщений об ошибках валидации (незаполненные поля или прошедшая дата окончания сроков);

3) если при отправке формы произошла ошибка сервера — отображение компонента React с сообщением об ошибке.

Алгоритм валидации формы «свободной задачи» и ее создания представлен на рис. 2.27.

На первом этапе алгоритма осуществляется проверка введенных пользователем данных. Пользователь заполняет форму создания задачи.

Далее следует этап проверки корректности данных. Если данные не соответствуют требованиям валидации, пользователю отображается соответствующие сообщения ошибок, и процесс создания задачи прерывается до устранения ошибок. Если данные прошли валидацию успешно, выполняется запрос на сервер для создания задачи.

Сформированный объект задачи, содержащий все необходимые поля, передается на соответствующую контрольную точку. Запрос сопровождается необходимыми заголовками, включая токен авторизации пользователя.

На завершающем этапе выполняется отображение созданной «свободной задачи» в колонке «В работе» на интерфейсе канбан-доски.

Схема работы алгоритма создания «свободной задачи»



Рис. 2.27

### 2.5.2. Реализация компонентов

2.5.1.1. Компонент CreateFreeTaskForm представляет собой форму, предназначенную для создания новой задачи. Он управляет формой ввода данных, проводит валидацию, обрабатывает отправку формы и взаимодействует с сервером для создания задачи. Компонент отображается по пользовательскому действию (например, нажатию кнопки «Создать задачу») и закрывается после успешного создания задачи или отмены действия.

Сводка состояний компонента:

1) form: object – объект, содержащий текущие значения всех полей формы (проект, специализация, исполнитель, текст задачи, контрольный срок, флаг важности). Значения обновляются при взаимодействии пользователя с формой;

2) validationSchema: object – схема валидации, определяющая обязательные поля, допустимые типы данных и ограничения (контрольный срок должен быть датой в будущем, поля формы не могут быть пустыми и т.д.).

Сводка функций компонента:

1) handleSubmit(values: object): void – функция, вызываемая при отправке формы. Выполняет проверку данных на соответствие validationSchema, формирует и отправляет запрос на сервер для создания задачи, и при успешной операции инициирует обновление интерфейса (например, добавление новой задачи в канбан-доску и закрытие модального окна). В случае ошибки отображает сообщение пользователю.

### 2.5.3. Описание контрольных примеров

Опишем контрольные примеры для функции создания свободной задачи:

1) авторизованный пользователь в роли исполнителя нажимает кнопку «Создать задачу», расположенную в верхней части колонки «В работе», после чего открывается модальное окно с формой создания свободной задачи. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.28;

Просмотр формы создания свободной задачи

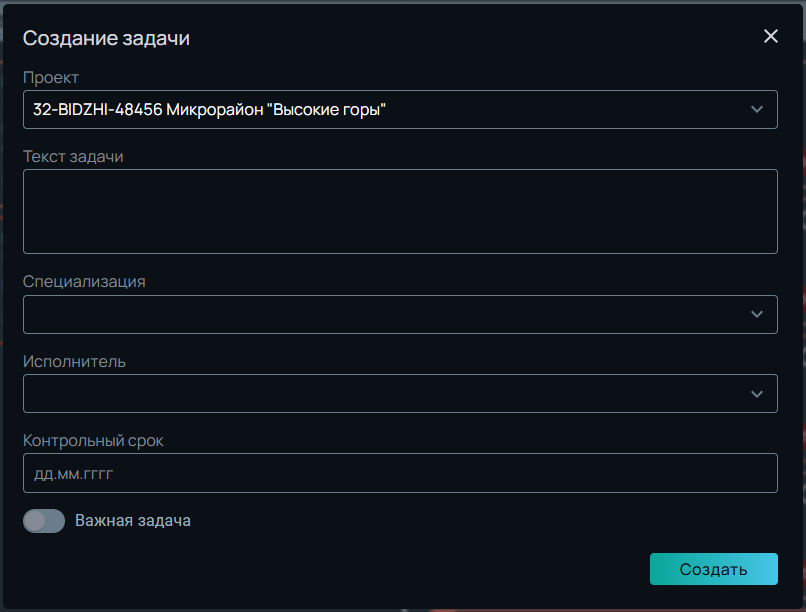


Рис. 2.28

2) авторизованный пользователь в роли исполнителя отправляет пустую форму свободной задачи, в результате чего срабатывает валидация и отображаются сообщения об ошибках. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.30;

Попытка отправки пустой формы

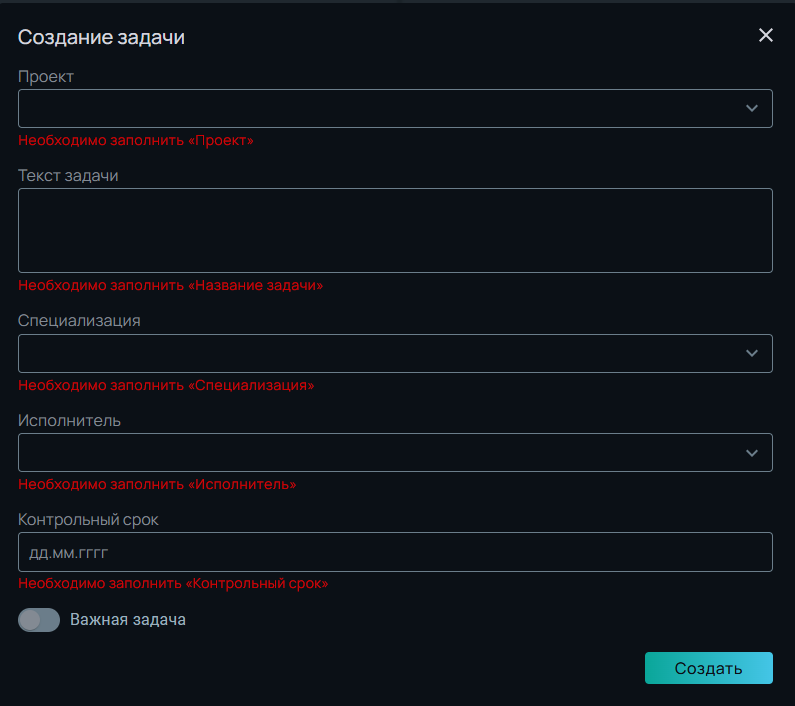


Рис. 2.29

3) авторизованный пользователь в роли исполнителя выбирает требуемую специализацию, после чего загружается список доступных исполнителей по данной специализации. Экранная формы для данного контрольного примера представлена на рис. 2.30;

Получение списка доступных исполнителей по выбранной специализации



Рис. 2.30

4) авторизованный пользователь в роли исполнителя заполняет форму создания свободной задачи, указывая необходимые параметры. После прохождения клиентской валидации и отправки формы созданная задача отображается в пользовательском интерфейсе в колонке «В работе» канбан-доски. Экранная формы для данного контрольного примера представлена на рис. 2.31;

Заполненная форма создания «свободной задачи», прошедшая валидацию

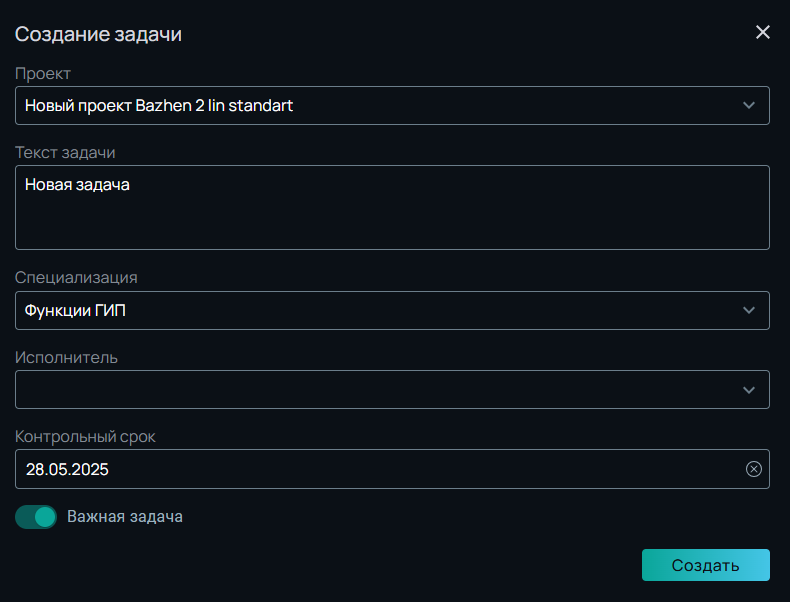


Рис. 2.31

## 2.6. Реализация функции «Фильтр задач канбан-доски»

### 2.6.1. Реализация алгоритмов

Общая схема работы алгоритма фильтра задач канбан-доски представлена на рис. 2.32.

Когда пользователь заходит на канбан-доску, в верхней части интерфейса отображается панель фильтрации задач. Эта панель содержит несколько полей: проект, диапазон дат, тип задачи, специализация исполнителя, группа проектировщиков и конкретный исполнитель. Цель этих полей — предоставить пользователю возможность гибко настраивать отображение задач на доске в соответствии с нужными условиями.

При первой загрузке канбан-доски отправляются запросы на сервер для получения справочных данных: загружаются доступные проекты, типы задач, специализации и группы проектировщиков. Эти данные нужны для заполнения соответствующих выпадающих списков в фильтре. Параллельно с этим происходит проверка, есть ли в localStorage сохраненные значения фильтра. Если такие значения найдены, они автоматически подставляются в соответствующие поля, и на основании этих данных сразу же выполняется запрос на получение списка задач. Это позволяет восстановить пользовательские настройки фильтрации, если он уже работал с доской ранее.

Если пользователь выбирает специализацию, система дополнительно отправляет запрос на сервер, чтобы получить список исполнителей, соответствующих выбранной специализации. Это поведение обусловлено тем, что список исполнителей напрямую зависит от выбранных навыков, и не загружается заранее вместе с остальными справочными данными. После получения ответа от сервера поле «Исполнитель» обновляется, и пользователь может выбрать конкретного человека из доступного списка.

Каждый раз, когда пользователь изменяет хотя бы одно поле фильтра, независимо от того, что именно он меняет — проект, тип задачи, дату или что-либо еще — происходит две вещи. Во-первых, новое состояние фильтра сохраняется в localStorage, чтобы сохранить текущие настройки для будущих сессий. Во-вторых, формируется и отправляется новый запрос на сервер для получения актуального списка задач с учетом всех выбранных параметров. Это обеспечивает мгновенное обновление отображаемых задач без необходимости дополнительных действий со стороны пользователя.

Общая схема работы алгоритма фильтра задач канбан-доски

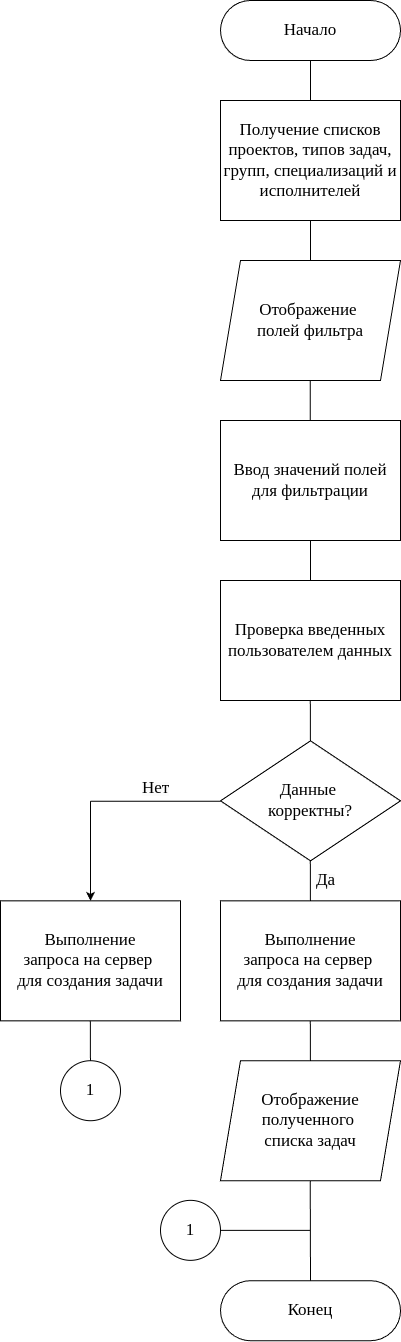


Рис. 2.32

Структура входных данных:

1) идентификатор текущей формы собственности типа number;

2) идентификатор текущего пользователя типа number.

Структура промежуточных данных:

1) идентификатор проекта типа Nullable<number>;

2) дата начала сроков типа string;

3) дата окончания сроков типа string;

4) идентификатор класса задачи типа Nullable<number>;

5) идентификатор специализации типа Nullable<number>;

6) идентификатор группы типа Nullable<number>.

Структура выходных данных:

При успешной авторизации пользователя:

1) если все поля заполнены корректно — отправка запроса на получение списка задач, подходящих под значения фильтров;

2) если форма заполнена некорректно — отображение сообщений об ошибках валидации;

3) если при отправке формы произошла ошибка сервера — отображение компонента React с сообщением об ошибке.

### 2.6.2. Реализация компонентов

2.6.2.1. Компонент Filter представляет собой форму для фильтрации списка документов на основе выбранного типа пользователя, текста поиска, даты и других параметров. Обрабатывает валидацию, отправку формы и связывается с сервером через вспомогательные методы.

1) form: object – объект, содержащий текущие значения всех полей формы (проект, специализация, исполнитель, текст задачи, контрольный срок, флаг важности). Значения обновляются при взаимодействии пользователя с формой;

2) validationSchema: object – схема валидации, определяющая обязательные поля, допустимые типы данных и ограничения.

Сводка функций компонента:

1) handleSubmit(values: object): void – функция, вызываемая при отправке формы. Выполняет проверку данных на соответствие validationSchema, формирует и отправляет запрос на сервер для получения списка задач. В случае ошибки отображает сообщение пользователю;

2) getFilterValues(userTypeId: number): void – функция, вызываемая при получении значений полей фильтра. Предназначена для загрузки ранее сохраненных параметров фильтрации из локального хранилища браузера. По переданному идентификатору формы собственности формируется уникальный ключ, с помощью которого из извлекаются соответствующие данные. Если значения найдены, они используются для автоматического заполнения полей фильтра; если данные отсутствуют, все поля остаются пустыми.

### 2.6.3. Описание контрольных примеров

Опишем контрольные примеры для функции фильтра задач канбан-доски:

1) авторизованный пользователь в роли исполнителя открывает панель канбан-доски. При первом открытии все поля фильтрации остаются пустыми, так как пользователь ранее не взаимодействовал с фильтром. Отображается полный список задач без каких-либо ограничений. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.33.

Список задач без ограничений

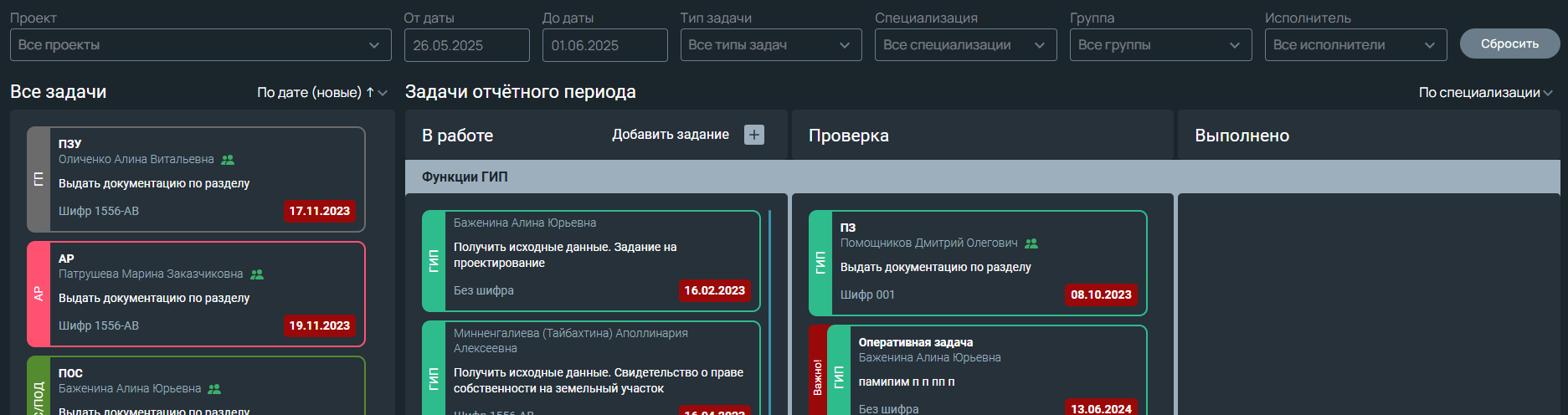


Рис. 2.33

2) авторизованный пользователь в роли исполнителя выбирает конкретный проект из доступного списка. Все остальные параметры фильтрации остаются без изменений. Отображаются задачи, актуальные в рамках конкретного проекта. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.34.

Список задач по определенному проекту

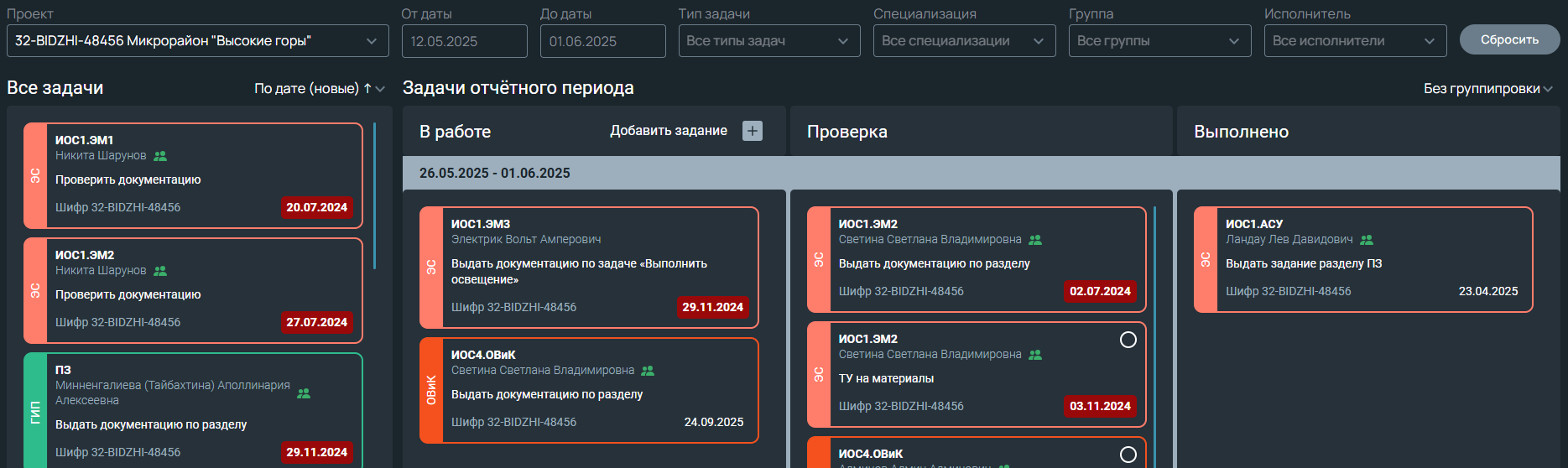


Рис. 2.34

3) авторизованный пользователь в роли исполнителя задает ограничения по срокам завершения задач. После этого фильтр применяет соответствующие условия, и в канбан-доске отображаются только те задачи, у которых срок завершения входит в указанный интервал. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.35.

Список задач, срок завершения которых входит в указанный временной интервал

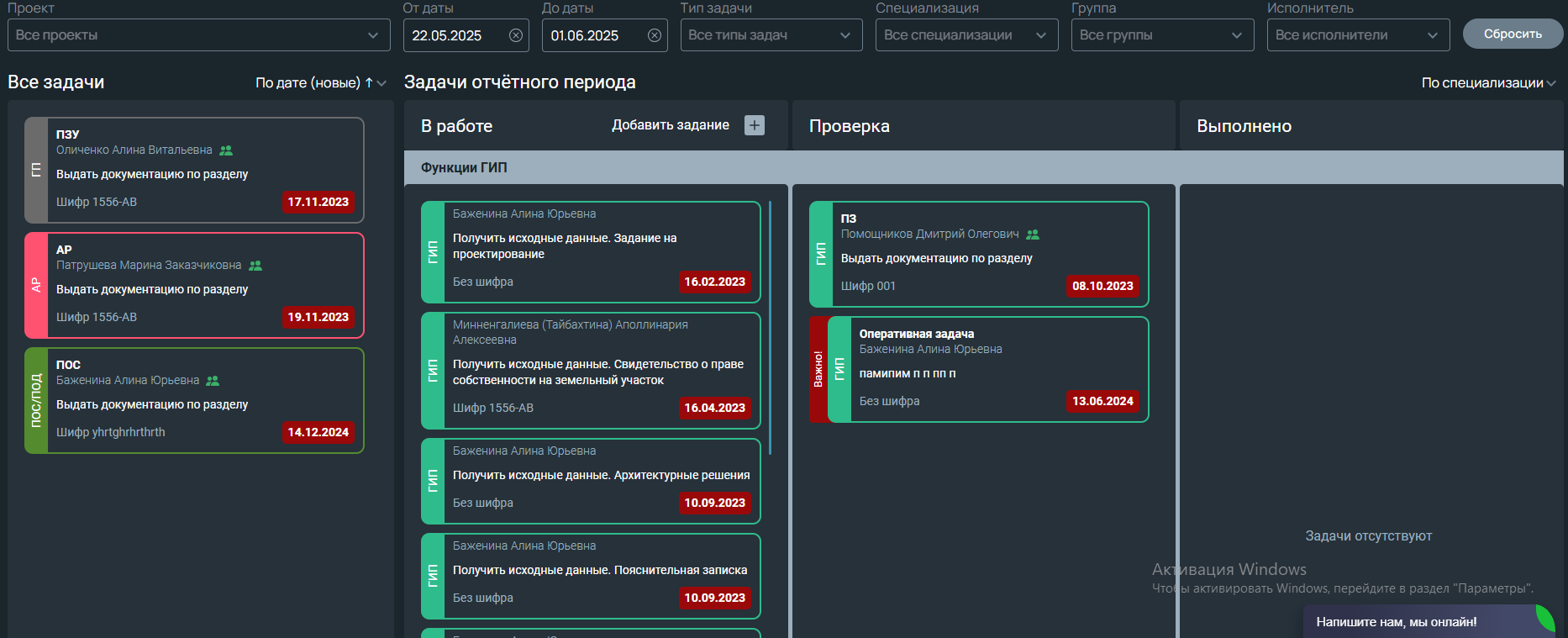


Рис. 2.35

4) авторизованный пользователь в роли исполнителя заполняет все доступные поля фильтра: выбирает проект, указывает даты, статус задач, приоритет, тип задачи и другие доступные параметры. После применения фильтра производится комплексная фильтрация задач. Отображаются только задачи, полностью соответствующие заданным критериям. Экранная форма, иллюстрирующая поведение интерфейса в данном случае, представлена на рис. 2.36.

Список задач при комплексной фильтрации

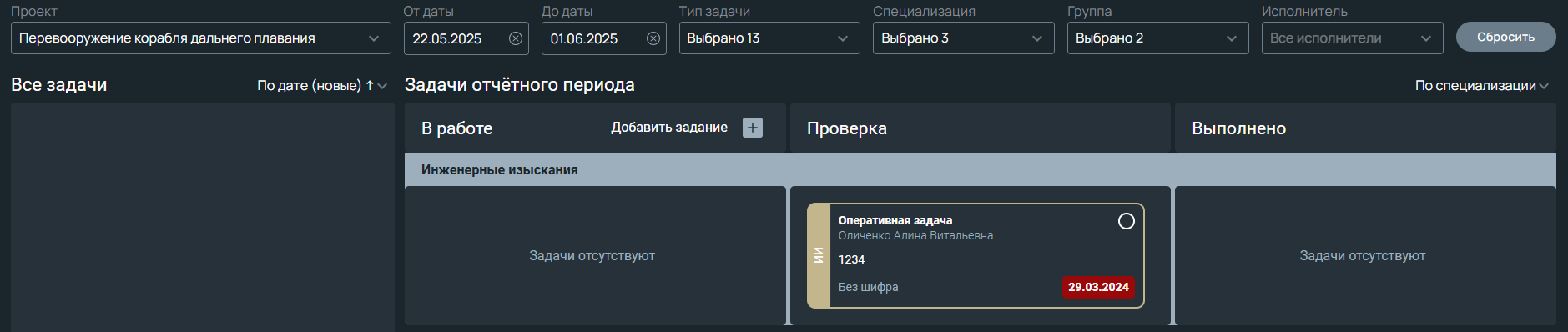


Рис. 2.36

# ЗАКЛЮЧЕНИЕ

В ходе работы над проектом были выявлены ключевые проблемы, затрудняющие эффективное управление проектированием в капитальном строительстве:

1) неэффективная коммуникация — отсутствие сквозной координации между заказчиком и исполнителем приводит к задержкам, дублированию задач и ошибкам в проектных решениях;

2) низкая прозрачность процессов — заказчик и другие участники проекта не имеют актуальной картины статуса задач, что затрудняет принятие своевременных управленческих решений;

3) несистемный документооборот — отсутствие унифицированного доступа к документам, сложности в их структурировании и поиске повышают риск потери данных и ошибок при проектировании.

Стандартные средства управления задачами не решают этих проблем, так как не учитывают специфику процессов проектирования, не интегрируются с архитектурой данных платформы и не поддерживают автоматическую генерацию задач на основе действий пользователей.

Для решения обозначенных проблем был разработан модуль интерактивной панели управления проектированием, встроенный в цифровую платформу «ПИРС»‎. Он автоматизирует:

1) визуальное отслеживание и фильтрацию проектов, разделов и документов;

2) управление задачами через канбан-доску, с учётом этапов проектирования;

3) автоматическую отправку задач на проверку главному инженеру проекта (ГИП), обеспечивая чёткий цикл согласования;

4) полную интеграцию с архитектурой и бизнес-логикой платформы «ПИРС»‎.

Благодаря модулю достигается необходимая прозрачность, сокращаются коммуникационные потери и повышается управляемость проектного процесса в условиях высокой сложности и многоуровневого взаимодействия.

В результате работы над данным модулем были реализованы:

1) интерфейсы интерактивной панели для просмотра иерархических списков проектов, разделов, документов и задач, с возможностью поиска и фильтрации по названиям и статусам;

2) алгоритмы для получения и отображения информации о проектах, разделах, документах и задачах, с поддержкой динамической фильтрации, создания задач и изменения их статусов с помощью канбан-доски.

Разработанный модуль интегрирован в программное обеспечение «Платформа ПИРС», которое активно используется такими компаниями, как «ПСК «Инжиниринг», «Технология», «БИМИТ», «Академия строительства» и рядом других организаций, работающих на базе цифровой платформы «ПИРС». В настоящее время на платформе зарегистрировано свыше 350 организаций и более 4000 пользователей. Подтверждением внедрения решения является акт, прилагаемый к пояснительной записке.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ПИРС 3.0, «Сервис проектировщиков России», [Электронный ресурс] // URL: https://pirs.online (дата обращения: 31.05.2025).

2. Procore, «Construction Management Software», [Электронный ресурс] // URL: https://www.procore.com (дата обращения: 31.05.2025).

3. SmartPM, «Project Controls & Schedule Analytics Platform», [Электронный ресурс] // URL: https://www.smartpmtech.com (дата обращения: 31.05.2025).

4. Aconex, «Construction Project Management Software», [Электронный ресурс] // URL: https://www.oracle.com/industries/aconex (дата обращения: 31.05.2025).

5. React.js. React «A JavaScript library for building user interfaces» [Электронный ресурс] // URL: https://reactjs.org (дата обращения: 31.05.2025).

6. Node.js. «Платформа Node.js» [Электронный ресурс] // URL: https://nodejs.org (дата обращения: 31.05.2025).

7. Redux Toolkit. «Менеджер состояний Redux Toolkit» [Электронный ресурс] // URL: https://redux-toolkit.js.org (дата обращения: 31.05.2025).

8. Методические указания по оформлению курсовых и выпускных квалификационных работ [Текст] / В.П. Соболева. – М.: Ижевский государственный технический университет имени М.Т. Калашникова, 2018. – 27 с.

9. Wikipedia. «Wikipedia – the free encyclopedia» [Электронный ресурс] // URL: https://en.wikipedia.org/wiki (дата обращения: 31.05.2025).

10. Material-UI. «Material-UI: A popular React UI framework» [Электронный ресурс] // URL: https://material-ui.com/ (дата обращения: 31.05.2025).

# ПРИЛОЖЕНИЕ 1

ТЕКСТЫ ПРОГРАММ

П.1.1. Projects.tsx

import React, { FC, useState, ChangeEvent, useEffect, FormEvent } from "react";

import { IconButton } from "@mui/material";

import { Add, Search } from "@mui/icons-material";

import cn from "classnames";

import style from "./style.module.scss";

import Switch from "../../../../newUI/Switch/Switch";

import SearchField from "../SearchField";

import DashboardSkeleton from "../Skeleton";

import ProjectsList from "./Projects/ProjectsList";

import { useAppDispatch, useAppSelector } from "../../../../../app/store";

import {

getDashboardDestinations,

setShowFulfilledSections,

} from "../../../../../app/feature/dashboard/destinations";

import {

getDashboardProjects,

setFulfilled,

setPreQuery,

} from "../../../../../app/feature/dashboard/projects";

export const Projects: FC = () => {

const dispatch = useAppDispatch();

const [query, setQuery] = useState<string>("");

const [showSearchBar, setShowSearchBar] = useState<boolean>(false);

const { pending, preQuery, fulfilled } = useAppSelector(getDashboardProjects);

const { userType } = useAppSelector(getDashboardDestinations);

const handleSubmit = (event: FormEvent) => {

event.preventDefault();

setQuery(preQuery);

};

const handleChange = (event: ChangeEvent<HTMLInputElement>) => {

const queryValue = event.target.value;

dispatch(setPreQuery(queryValue));

if (!queryValue.length) {

setQuery("");

}

};

const handleToggleFulfilled = () => {

dispatch(setFulfilled(!fulfilled));

dispatch(setShowFulfilledSections(!fulfilled));

};

useEffect(() => {

setQuery(preQuery);

}, []);

useEffect(() => {

if (fulfilled) {

dispatch(setFulfilled(false));

dispatch(setShowFulfilledSections(false));

}

if (query.length) {

dispatch(setPreQuery(""));

}

}, [userType]);

useEffect(() => {

setQuery(preQuery);

}, []);

const searchPending = pending && preQuery.length;

return (

<div className={cn(style.wrapper, style.projectsPane)}>

<form

onSubmit={handleSubmit}

className={cn(style.header, style.projectHeader)}

>

{isMobile ? (

<>

<h3>Проект</h3>

<div className={style.mobileHeader}>

<p>Выполнено</p>

<Switch checked={fulfilled} onChange={handleToggleFulfilled} />

<IconButton

onClick={() => setShowSearchBar(!showSearchBar)}

className={style.mobileSearchBar}

>

<Search style={{ color: "#9DAFBD" }} />

</IconButton>

<a

className={style.addProject}

href={`/project/create?userTypeId=${userType}`}

target="blank"

>

<IconButton className={style.roundButton}>

<Add />

</IconButton>

</a>

</div>

{showSearchBar && (

<SearchField

pending={searchPending}

className={style.searchBar}

value={preQuery}

onChange={handleChange}

handleSubmit={handleSubmit}

mobile

/>

)}

</>

) : (

<>

<h3>Проект</h3>

<p>Выполнено</p>

<Switch

className={style.fulfilledSwitch}

checked={fulfilled}

onChange={handleToggleFulfilled}

/>

<SearchField

pending={searchPending}

className={style.searchBar}

value={preQuery}

onChange={handleChange}

handleSubmit={handleSubmit}

/>

<a

className={style.addProject}

href={`/project/create?userTypeId=${userType}`}

target="blank"

>

<IconButton className={style.roundButton}>

<Add />

</IconButton>

</a>

</>

)}

</form>

{userType ? (

<ProjectsList fulfilled={fulfilled} query={query} />

) : (

<DashboardSkeleton count={skeletonCount} />

)}

</div>

);

};

П.1.2. CreateFreeTaskForm.tsx

import { Controller } from "react-hook-form";

import { Modal } from "src/FSD/shared/uiKit/v2/Modal";

import { Select } from "src/FSD/shared/uiKit/v2/Select";

import { TextField } from "src/FSD/shared/uiKit/v2/TextField";

import { DateField } from "src/FSD/shared/uiKit/v2/DateField";

import { Button } from "src/FSD/shared/uiKit/v2/Button";

import { SwitchLabel } from "src/FSD/shared/uiKit/v2/SwitchLabel";

import { SelectSpecialization } from "src/FSD/features/v2/SelectSpecialization";

import { useCreateFreeTask } from "../lib/hooks";

export const CreateFreeTaskForm = () => {

const dispatch = useAppDispatch();

const isViewPending = useAppSelector(KanbanModel.getIsViewPending);

const isTaskCreationPending = useAppSelector(

KanbanModel.getIsTaskCreationPending

);

const selectOptions = useAppSelector(KanbanModel.getSelectOptions);

const newTaskPending = useAppSelector(KanbanModel.getFilterPending);

const { userType: userTypeId } = useAppSelector(getDashboardDestinations);

const isSubmitting = isTaskCreationPending || isViewPending;

const {

handleSubmit,

register,

watch,

control,

setValue,

formState: { errors },

} = useForm<KanbanType.FreeTaskFields>({

defaultValues,

mode: "onChange",

resolver: yupResolver(createFreeTaskSchema),

});

const projectValue = watch("project");

const partGroupValue = watch("partGroup");

const onSubmit = async (data: KanbanType.FreeTaskFields) => {

await dispatch(KanbanApi.createFreeTask(data));

dispatch(

KanbanModel.setCreateFreeTask({

isOpen: false,

})

);

};

// Выгрузка списка проектов

useEffect(() => {

dispatch(KanbanApi.fetchProjectsByUserTypeId(userTypeId));

}, [userTypeId]);

useEffect(() => {

const partGroups = [];

if (partGroupValue?.id) {

partGroups.push({

id: partGroupValue.id,

title: "",

});

}

dispatch(

KanbanApi.fetchUsersListByUserTypeId({

userTypeId,

projectId: projectValue?.id as number,

partGroups,

})

);

}, [userTypeId, projectValue, partGroupValue]);

useEffect(() => {

const userId = getUserId();

const { project } = getFilterValuesFromLocalStorage(userId, userTypeId);

if (project) {

setValue("project", project);

}

}, []);

return (

<Modal.Layout>

<Modal.Header>Создание задачи</Modal.Header>

<Modal.Form onSubmit={handleSubmit(onSubmit)}>

<Controller

name="project"

control={control}

render={({ field: { value, onChange } }) => (

<Select

value={value}

changeHandler={(\_, newValue) => {

onChange(newValue);

setValue("executor", null);

}}

isLoading={newTaskPending.getProjects}

label="Проект"

options={selectOptions.projects}

error={Boolean(errors.project)}

helperText={errors.project?.message}

/>

)}

/>

<TextField

{...register("content")}

label="Текст задачи"

size={"small"}

multiline

minRows={3}

maxRows={12}

error={Boolean(errors.content)}

helperText={errors.content?.message}

/>

<Controller

name="partGroup"

control={control}

render={({ field: { value, onChange } }) => (

<SelectSpecialization

value={value}

changeHandler={(\_, newValue) => {

onChange(newValue);

setValue("executor", null);

}}

error={Boolean(errors.partGroup)}

helperText={errors.partGroup?.message}

/>

)}

/>

<Controller

name="executor"

control={control}

render={({ field: { value, onChange } }) => (

<Select

key={value?.id ?? null}

value={value}

changeHandler={(\_, newValue) => {

onChange(newValue);

}}

isLoading={newTaskPending.getUsersList}

label="Исполнитель"

options={selectOptions.users}

error={Boolean(errors.executor)}

helperText={errors.executor?.message}

/>

)}

/>

<Controller

name="dateDeadline"

control={control}

render={({ field: { value, onChange } }) => (

<DateField

label="Контрольный срок"

changeDateHandler={(date) => {

onChange(date);

}}

startDateProp={value}

error={Boolean(errors.dateDeadline)}

helperText={errors.dateDeadline?.message}

/>

)}

/>

<Controller

name="importance"

control={control}

render={({ field: { value, onChange } }) => (

<SwitchLabel

checked={value}

onChange={() => {

onChange(!value);

}}

label="Важная задача"

/>

)}

/>

<Modal.Controls>

<Button

type="submit"

disabled={isSubmitting}

isLoading={isSubmitting}

>

Создать

</Button>

</Modal.Controls>

</Modal.Form>

</Modal.Layout>

);

};

П.1.3. Column.tsx

import { ReactNode } from "react";

import cn from "classnames";

import { Draggable, Droppable } from "react-beautiful-dnd";

import { KanbanType } from "src/FSD/entities/kanban";

import styles from "./Column.module.scss";

import ColumnTask from "./ColumnTask";

import ColumnSortFilter from "./ColumnSortFilter";

import ColumnHeader from "./ColumnHeader";

import ColumnTitle from "./ColumnTitle";

import ColumnRange from "./ColumnRange";

import ColumnAddTask from "./ColumnAddTask";

import { useColumn } from "../lib/hooks/useColumn";

import { NoTasksFound } from "./NoTasksFound";

type ColumnProps = {

position?: string;

isEmpty?: boolean;

isStatic?: boolean;

isAllTasks?: boolean;

header?: ReactNode;

tasks?: KanbanType.Task[];

};

const Column = ({

position = "",

header,

isStatic,

isEmpty,

isAllTasks,

tasks = [],

}: ColumnProps) => {

const { handleShowTaskInfo, noGroupping } = useColumn();

if (isEmpty) {

return (

<div className={cn(styles.column, styles.column\_empty)}>{header}</div>

);

}

if (isStatic) {

return (

<div className={styles.column}>

<div className={cn(styles.tasksList, styles.tasksList\_static)}>

{tasks.length ? (

tasks.map((task: KanbanType.Task) => {

return (

<div

onClick={() => handleShowTaskInfo(task)}

className={styles.taskCard}

>

<Column.Task key={task.id} position={position} {...task} />

</div>

);

})

) : (

<NoTasksFound isStatic={isStatic} />

)}

</div>

</div>

);

}

return (

<div

className={cn(

styles.column,

isAllTasks && styles.column\_allTasks,

isAllTasks && noGroupping && styles.column\_allTasks\_noGroups

)}

>

{header}

<Droppable droppableId={position}>

{(provided) => (

<div

id={position}

key={position}

className={cn(styles.tasksList, styles.tasksList\_dynamic)}

{...provided.droppableProps}

ref={provided.innerRef}

>

{tasks.length ? (

tasks.map((task: KanbanType.Task, index: number) => {

return (

<Draggable

key={task.id}

draggableId={String(task.id)}

index={index}

>

{(provided) => (

<div

onClick={() => handleShowTaskInfo(task)}

className={styles.taskCard}

ref={provided.innerRef}

{...provided.draggableProps}

{...provided.dragHandleProps}

>

<Column.Task

key={task.id}

position={position}

{...task}

/>

</div>

)}

</Draggable>

);

})

) : (

<NoTasksFound isStatic={isStatic} />

)}

{provided.placeholder}

</div>

)}

</Droppable>

</div>

);

};

Column.Header = ColumnHeader;

Column.SortFilter = ColumnSortFilter;

Column.Task = ColumnTask;

Column.Title = ColumnTitle;

Column.AddTask = ColumnAddTask;

Column.Range = ColumnRange;

export default Column;

П.1.4. SectionEditing.tsx

export const Filter = () => {

const dispatch = useAppDispatch();

const selectOptions = useAppSelector(KanbanModel.getSelectOptions);

const filterPending = useAppSelector(KanbanModel.getFilterPending);

const groupBy = useAppSelector(KanbanModel.getGroupByValue);

const sort = useAppSelector(KanbanModel.getSortValue);

const datePlaceholders = useAppSelector(KanbanModel.getSetting);

const isFirstlyOpen = useRef<boolean>(true);

const userTypeIdRef = useRef<Nullable<number>>(null);

const {

control,

watch,

trigger,

setValue,

formState: { errors, isValidating, isValid },

} = useForm<KanbanType.FilterValues>({

defaultValues: useMemo(

() => getFilterValues(userTypeId),

[userTypeId, datePlaceholders]

),

resolver: yupResolver(filterSchema),

});

const projectValue = watch("project");

const partGroupValue = watch("partGroup");

const groupValue = watch("group");

useClearUpFields(groupValue, setValue);

const onSubmit = (filterValues: KanbanType.FilterValues) => {

saveFilterValuesInLocalStorage(userTypeId, filterValues);

dispatch(KanbanModel.setGroupByValue(filterValues.groupBy));

dispatch(KanbanModel.setSortValue(filterValues.sort));

dispatch(

KanbanApi.fetchViewByUserTypeId({

userTypeId,

filterValues,

showSpinner: true,

})

);

};

const filterValues = watch();

const filterValuesExtendedRef = useRef<Nullable<any>>(null);

const submitKanbanFilter = () => {

const filterValuesExtended = { ...filterValues, groupBy, sort };

if (

!isDeepEqual(filterValuesExtended, filterValuesExtendedRef.current) &&

!isValidating &&

isValid

) {

onSubmit(filterValuesExtended);

filterValuesExtendedRef.current = filterValuesExtended;

}

};

useEffect(() => {

if (

!selectOptions.projects.length ||

filterPending.getProjects ||

!projectId ||

!isFirstlyOpen.current

) {

return;

}

const project = selectOptions.projects.find(

(project) => project.id === projectId

);

setValue("project", project ?? null);

const filterValues = getFilterValues(userTypeId);

saveFilterValuesInLocalStorage(userTypeId, { ...filterValues, project });

isFirstlyOpen.current = false;

}, [projectId, selectOptions, userTypeId, filterPending]);

const handleClear = () => {

Object.entries(initialFilterParams).forEach(([field, value]) => {

setValue(field as keyof KanbanType.FilterValues, value);

});

};

useEffect(() => {

if (!userTypeId) {

return;

}

dispatch(KanbanApi.fetchWorkFlowList());

dispatch(KanbanApi.fetchProjectsByUserTypeId(userTypeId));

return () => {

dispatch(KanbanModel.clearFilterData());

};

}, [userTypeId]);

useEffect(() => {

if (!firmId) {

dispatch(KanbanModel.setFilterGroupsList([]));

return;

}

dispatch(

KanbanApi.fetchGroupsByFirmIdAndPartGroupId({

firmId,

partGroups: partGroupValue,

})

);

}, [partGroupValue.length, firmId]);

useEffect(() => {

if (!userTypeId) {

return;

}

if (!projectValue?.id && !partGroupValue.length && !groupValue.length) {

const {

data: { surname, name, id },

} = getParsedUserToken();

const currentUser = {

id,

title: `${surname} ${name}`,

};

dispatch(KanbanModel.setFilterUsersList([currentUser]));

return;

}

dispatch(

KanbanApi.fetchUsersListByUserTypeId({

userTypeId,

projectId: projectValue?.id as number,

partGroups: partGroupValue,

groups: groupValue,

})

);

}, [userTypeId, projectValue?.id, partGroupValue.length, groupValue.length]);

return (

<form className={styles.filter}>

<div className={styles.filter\_left}>

<Controller

name="project"

control={control}

render={({ field: { value, onChange } }) => (

<Select

value={value}

changeHandler={(\_, newValue) => {

onChange(newValue);

setValue("executor", []);

}}

placeholder="Все проекты"

isLoading={filterPending.getProjects}

label="Проект"

options={selectOptions.projects}

/>

)}

/>

</div>

<div

className={cn(

styles.filter\_right,

hasFirm && styles.filter\_right\_hasFirm

)}

>

<div className={styles.filter\_right\_\_dates}>

<Controller

name="dateStart"

control={control}

render={({ field: { value, onChange } }) => {

const dateValue = value ? moment(value).utc().toDate() : null;

return (

<DateField

label="От даты"

changeDateHandler={(date) => {

onChange(date);

trigger("dateLimit");

}}

startDateProp={dateValue}

error={Boolean(errors.dateLimit)}

placeholderText={datePlaceholders.start}

/>

);

}}

/>

<Controller

name="dateLimit"

control={control}

render={({ field: { value, onChange } }) => {

const dateValue = value ? moment(value).utc().toDate() : null;

return (

<DateField

label="До даты"

changeDateHandler={(date) => {

onChange(date);

trigger("dateLimit");

}}

startDateProp={dateValue}

error={Boolean(errors.dateLimit)}

placeholderText={datePlaceholders.end}

/>

);

}}

/>

</div>

<Controller

name="workFlow"

control={control}

render={({ field: { value, onChange } }) => (

<SelectMultipleLight

value={value}

changeHandler={(newValue) => {

onChange(newValue);

}}

onBlur={() => {

submitKanbanFilter();

}}

placeholder="Все типы задач"

isLoading={filterPending.getWfList}

label="Тип задачи"

options={selectOptions.workFlows}

/>

)}

/>

<Controller

name="partGroup"

control={control}

render={({ field: { value, onChange } }) => (

<SpecializationMultipleLight

value={value}

changeHandler={(newValue) => {

onChange(newValue);

setValue("executor", []);

}}

onBlur={() => {

submitKanbanFilter();

}}

label="Специализация"

placeholder="Все специализации"

/>

)}

/>

{hasFirm && (

<Controller

name="group"

control={control}

render={({ field: { value, onChange } }) => {

return (

<SelectMultipleLight

value={value}

changeHandler={(newValue) => {

onChange(newValue);

setValue("executor", []);

}}

isLoading={filterPending.getGroups}

onBlur={() => {

submitKanbanFilter();

}}

placeholder="Все группы"

label="Группа"

options={selectOptions.groups}

optionsClassName={styles.optionsClassName}

/>

);

}}

/>

)}

<Controller

name="executor"

control={control}

render={({ field: { value, onChange } }) => {

return (

<SelectMultipleLight

value={value}

changeHandler={(newValue) => {

onChange(newValue);

}}

onBlur={() => {

submitKanbanFilter();

}}

isLoading={filterPending.getUsersList}

label="Исполнитель"

options={selectOptions.users}

placeholder="Все исполнители"

/>

);

}}

/>

<div className={styles.filter\_\_controls}>

<CustomButton

onClick={handleClear}

className={styles.filter\_\_controls\_\_button}

>

Сбросить

</CustomButton>

</div>

</div>

</form>

);

};

# ПРИЛОЖЕНИЕ 2

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ (СПРАВОЧНАЯ ИНФОРМАЦИЯ ПО РАБОТЕ С ИНТЕРАКТИВНОЙ ПАНЕЛЬЮ УПРАВЛЕНИЯ ПРОЕКТИРОВАНИЕМ ОБЪЕКТОВ КАПИТАЛЬНОГО СТРОИТЕЛЬСТВА)

П.2.1. Назначение модуля

Назначение интерактивной панели управления проектированием объектов капитального строительства заключается в автоматизации процессов управления проектированием и обеспечении эффективной координации всех участников проектного процесса — проектировщиков, заказчиков и исполнителей.

Данный модуль является частью клиентского приложения цифровой платформы «ПИРС» и служит инструментом для централизованного управления проектами. Интерактивная панель обеспечивает прозрачность, оперативность и согласованность действий всех участников, снижая риски несвоевременного выполнения проектных этапов и повышая общую управляемость процесса проектирования.

П.2.2. Условия применения модуля

Интерактивная панель доступна авторизованным пользователям цифровой платформы «ПИРС» по адресу: https://pirs.online/user/dashboard.

Для использования панели необходимо наличие учетной записи на платформе. Доступ предоставляется только зарегистрированным пользователям — исполнителям, заказчикам и другим участникам проектного процесса.

Доступ к функциональности модуля зависит от роли пользователя (исполнитель, главный инженер проекта, заказчик). Некоторые действия (например, согласование задач) доступны только главному инженеру проекта.

Модуль предназначен для работы в современных браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge). Корректная работа не гарантируется в устаревших версиях.

Полноценное использование панели возможно при наличии добавленных проектов и задач. Без данных некоторые разделы модуля будут неактивны или отображать уведомление об отсутствии информации.

Модуль работает в режиме реального времени и требует стабильного подключения к интернету для корректной загрузки, обновления данных и взаимодействия с сервером.

П.2.3. Подготовка к работе

Для доступа к интерактивной панели пользователь должен пройти регистрацию на цифровой платформе по адресу https://pirs.online, указав актуальные данные и выбрав соответствующую роль.

После успешной регистрации необходимо войти в личный кабинет по адресу. Для начала работы необходимо создать новый проект или подключиться к уже существующему. Проект должен содержать базовые элементы: разделы, документы, задачи.

Убедитесь, что используется современный веб-браузер (Google Chrome, Mozilla Firefox, Microsoft Edge) и имеется стабильное подключение к интернету. Необходима поддержка JavaScript.

П.2.4. Описание операций

Пользователь просматривает список проектов, сгруппированных по форме собственности (заказчик или проектировщик). Каждый проект содержит сведения о количестве договоров, разделов и задач. Для комплексных проектов предусмотрена динамическая загрузка подпроектов и постраничная пагинация, как показано на рис. П.2.1.

Просмотр списка проектов

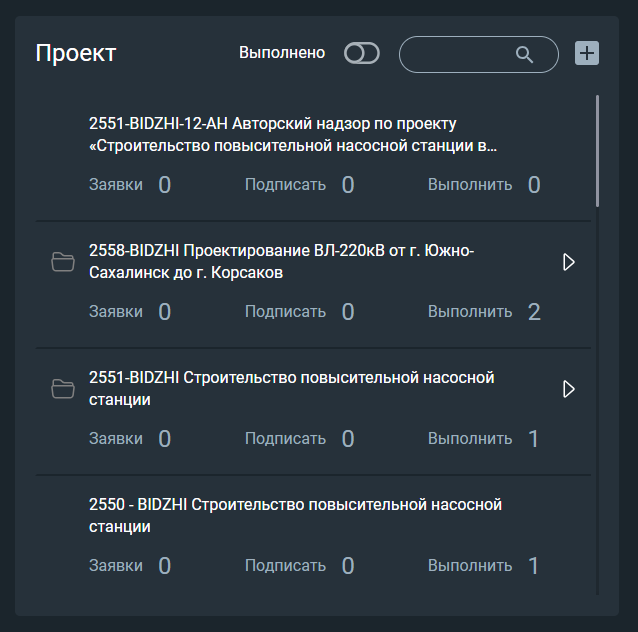


Рис. П.2.1

Разделы и подразделы проекта представлены в иерархической списке с указанием шифра, количества задач и текущего статуса.

Для упрощения навигации реализованы поиск и фильтрация по названию, статусу и шифру, как показано на рис. П.2.2.

Просмотр списка разделов проекта

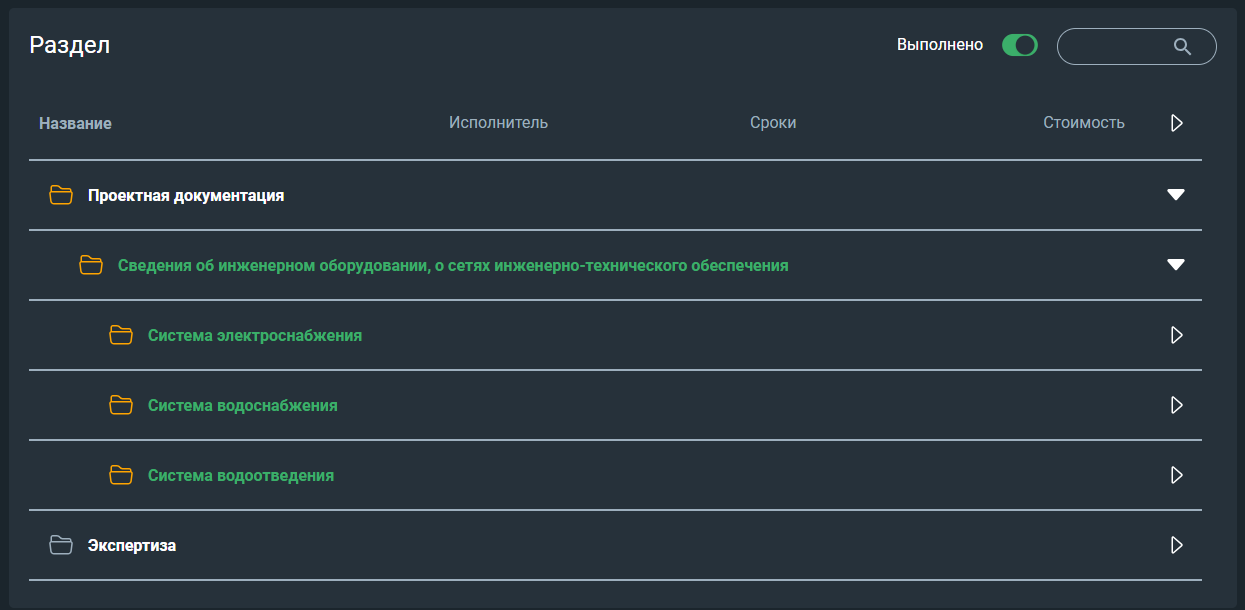


Рис. П.2.2

Пользователь получает доступ к документам (акты, отчёты, дополнительные соглашения), связанным с выбранным разделом проекта.

Интерфейс позволяет фильтровать документы по статусу подписания, названию и шифру, как показано на рис. П.2.3.

Просмотр списка документов раздела

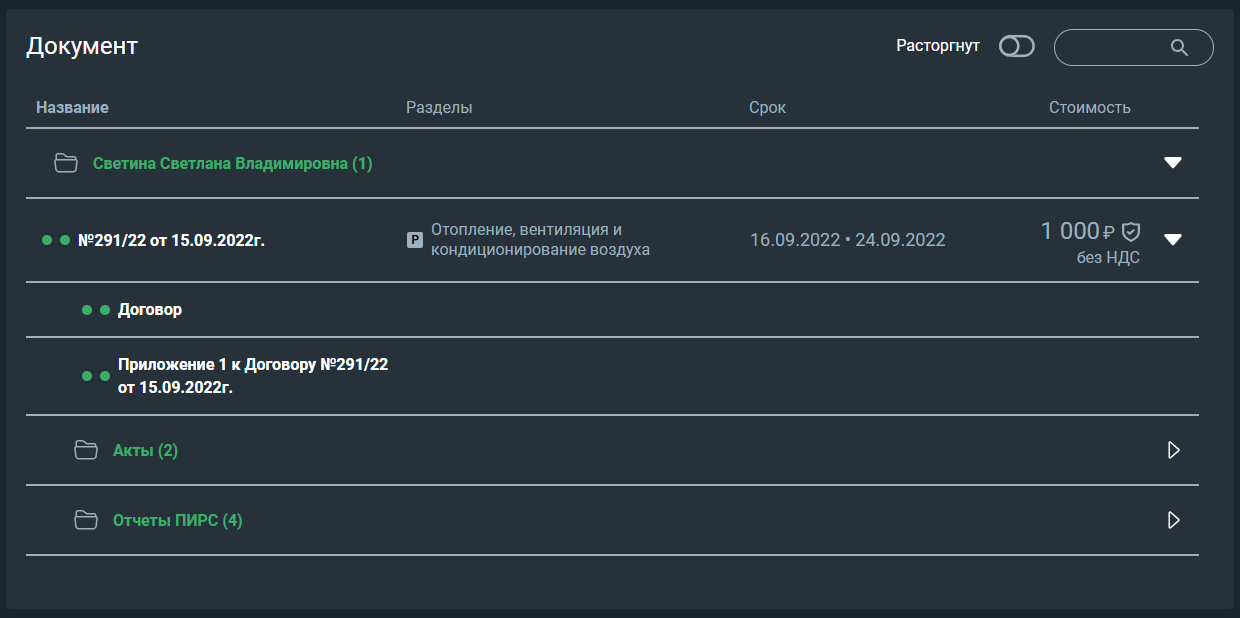


Рис. П.2.3

Задачи канбан-доски можно перетаскивать между колонками, что позволяет отслеживать процесс выполнения, как показано на рис. П.2.4.

Управление задачами на канбан-доске

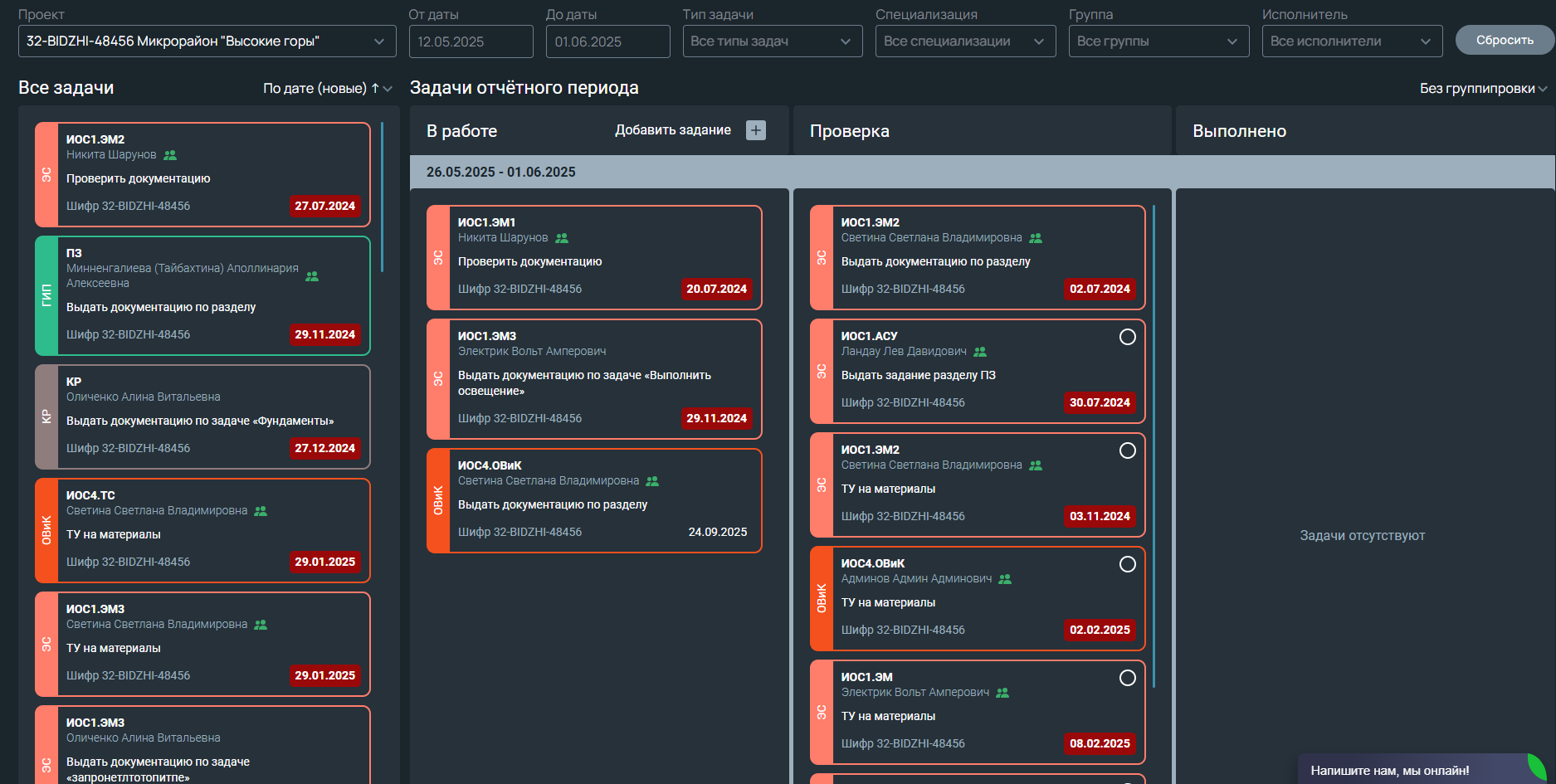


Рис. П.2.4

Свободная задача создается без прохождения формальных этапов согласования и сразу попадает в работу. Пример заполненной формы показан на рис. П.2.5.

Форма создания «свободной задачи»

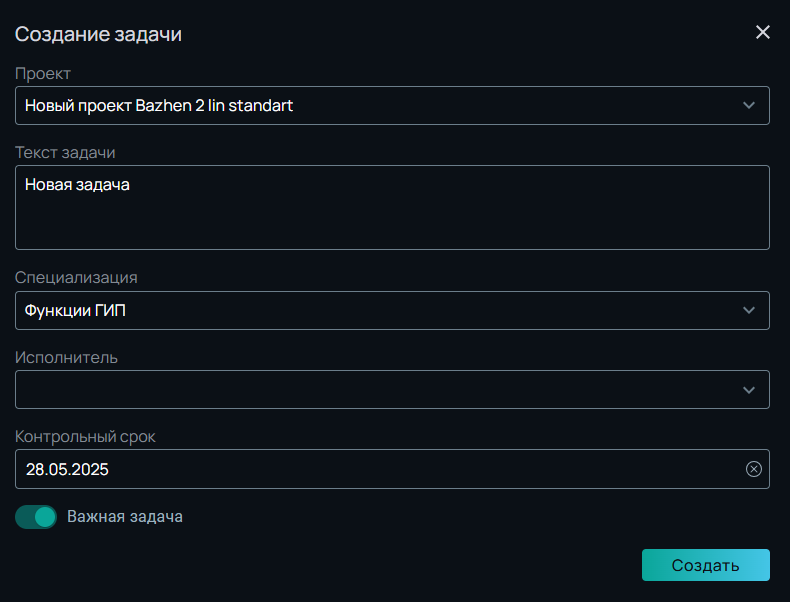


Рис. П.2.5

Фильтр задач помогает пользователю сузить список отображаемых задач. Изменения в фильтре моментально обновляют отображение канбан-доски, как показано на рис. П.2.6.

Фильтр задач канбан-доски

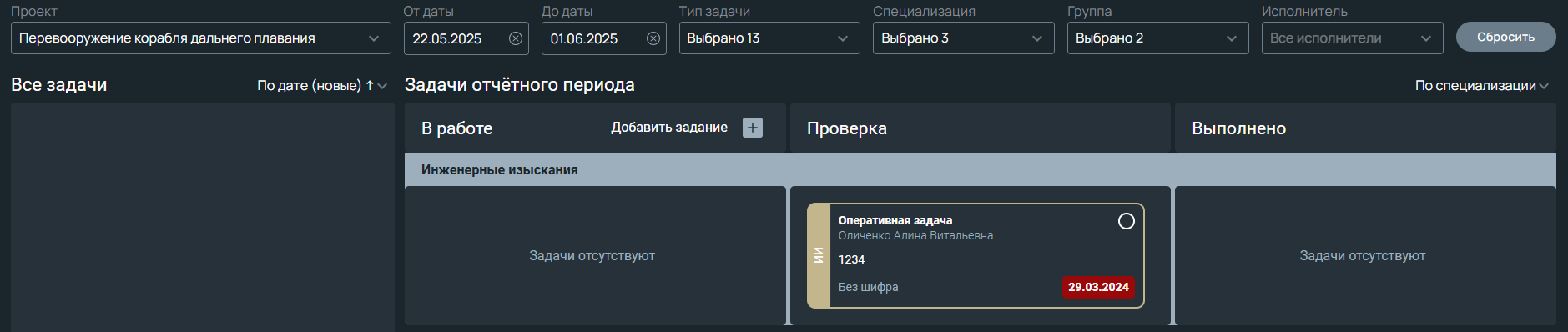


Рис. П.2.6

П.2.5. Аварийные ситуации

В случае отказа или сбоя интерактивной панели рекомендуется обратиться в чат поддержки цифровой платформы «ПИРС», который находится в правом нижнем углу веб-страницы.